Developer Guide

AWS Key Management Service

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Key Management Service: Developer Guide

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

AWS Key Management Service

AWS Key Management Service (AWS KMS) is a managed service that makes it easy for you to create and control the cryptographic keys that are used to protect your data. AWS KMS uses hardware security modules (HSM) to protect and validate your AWS KMS keys under the FIPS 140-2 Cryptographic Module Validation Program. China (Beijing) and China (Ningxia) Regions do not support the FIPS 140-2 Cryptographic Module Validation Program. AWS KMS uses OSCCA certified HSMs to protect KMS keys in China Regions.

AWS KMS integrates with most other AWS services that encrypt your data. AWS KMS also integrates with AWS CloudTrail to log use of your KMS keys for auditing, regulatory, and compliance needs.

You can use the AWS KMS API to create and manage KMS keys and special features, such as custom key stores, and use KMS keys in cryptographic operations. For detailed information, see the *AWS Key Management Service API Reference*.

You can create and manage your AWS KMS keys:

- Create, edit, and view symmetric and asymmetric KMS keys, including HMAC keys.
- Control access to your KMS keys by using key policies, IAM policies, and grants. AWS KMS supports attribute-based access control (ABAC). You can also refine policies by using condition keys.
- Create, delete, list, and update *aliases*, friendly names for your KMS keys. You can also use aliases to control access to your KMS keys.
- Tag your KMS keys for identification, automation, and cost tracking. You can also use tags to control access to your KMS keys.
- Enable and disable KMS keys.
- Enable and disable automatic rotation of the cryptographic material in a KMS key.
- Delete KMS keys to complete the key lifecycle.

You can use your KMS keys in cryptographic operations. For examples, see Programming the AWS KMS API.

- Encrypt, decrypt, and re-encrypt data with symmetric or asymmetric KMS keys.
- Sign and verify messages with asymmetric KMS keys.

- Generate exportable symmetric data keys and asymmetric data key pairs.
- Generate and verify HMAC codes.
- Derive shared secrets with asymmetric KMS keys.
- Generate random numbers suitable for cryptographic applications.

You can use the advanced features of AWS KMS.

- Create multi-Region keys, which act like copies of the same KMS key in different AWS Regions.
- Import cryptographic material into a KMS key.
- Create KMS keys in an AWS CloudHSM key store backed by your AWS CloudHSM cluster.
- Create KMS keys in an external key store backed by your cryptographic keys outside of AWS.
- Connect directly to AWS KMS through a private endpoint in your VPC.
- Use hybrid post-quantum TLS to provide forward-looking encryption in transit for the data that you send to AWS KMS.

By using AWS KMS, you gain more control over access to data you encrypt. You can use the key management and cryptographic features directly in your applications or through AWS services integrated with AWS KMS. Whether you write applications for AWS or use AWS services, AWS KMS enables you to maintain control over who can use your AWS KMS keys and gain access to your encrypted data.

AWS KMS integrates with AWS CloudTrail, a service that delivers log files to your designated Amazon S3 bucket. By using CloudTrail you can monitor and investigate how and when your KMS keys have been used and who used them.

AWS KMS in AWS Regions

The AWS Regions in which AWS KMS is supported are listed in AWS Key Management Service Endpoints and Quotas. If an AWS KMS feature is not supported in an AWS Region that AWS KMS supports, the regional difference is described in the topic about the feature.

AWS KMS pricing

As with other AWS products, using AWS KMS does not require contracts or minimum purchases. For more information about AWS KMS pricing, see AWS Key Management Service Pricing.

Service level agreement

AWS Key Management Service is backed by a service level agreement that defines our service availability policy.

Learn more

- To learn about the terms and concepts used in AWS KMS, see AWS KMS Concepts.
- For information about the AWS KMS API, see the AWS Key Management Service API Reference. For examples in different programming languages, see Programming the AWS KMS API.
- To learn how to use AWS CloudFormation templates to create and manage keys and aliases, see Creating AWS KMS resources with AWS CloudFormation and AWS Key Management Service resource type reference in the AWS CloudFormation User Guide.
- For detailed technical information about how AWS KMS uses cryptography and secures KMS keys, see AWS Key Management Service Cryptographic Details. The Cryptographic Details documentation does not describe how AWS KMS works in the China (Beijing) and China (Ningxia) Regions.
- For a list of AWS KMS endpoints, including FIPS endpoints, in each AWS Region, see Service endpoints in the AWS Key Management Service topic of the AWS General Reference.
- For help with questions about AWS KMS, see the AWS Key Management Service Discussion Forum.

AWS KMS in the AWS SDKs

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP
- AWS SDK for Python (Boto3)
- AWS SDK for Ruby

AWS KMS concepts

Learn the basic terms and concepts used in AWS Key Management Service (AWS KMS) and how they work together to help protect your data.

Topics

AWS KMS keys

AWS KMS keys (KMS keys) are the primary resource in AWS KMS. You can use a KMS key to encrypt, decrypt, and re-encrypt data. It can also generate data keys that you can use outside of AWS KMS.

Typically, you'll use symmetric encryption KMS keys, but you can create and use asymmetric KMS keys for encryption or signing, and create and use HMAC KMS keys to generate and verify HMAC tags.

Note

AWS KMS is replacing the term *customer master key (CMK)* with *AWS KMS key* and *KMS key*. The concept has not changed. To prevent breaking changes, AWS KMS is keeping some variations of this term.

An *AWS KMS key* is a logical representation of a cryptographic key. A KMS key contains metadata, such as the key ID, key spec, key usage, creation date, description, and key state. Most importantly, it contains a reference to the key material that is used when you perform cryptographic operations with the KMS key.

You can create a KMS key with cryptographic key material generated in AWS KMS FIPS validated hardware security modules. The key material for symmetric KMS keys and the private keys of asymmetric KMS key never leaves AWS KMS unencrypted. To use or manage your KMS keys, you must use AWS KMS. For information about creating and managing KMS keys, see *Managing keys*. For information about using KMS keys, see the AWS Key Management Service API Reference.

By default, AWS KMS creates the key material for a KMS key. You cannot extract, export, view, or manage this key material. The only exception is the public key of an asymmetric key pair, which you can export for use outside of AWS. Also, you cannot delete this key material; you must delete the KMS key. However, you can import your own key material into a KMS key, or use a custom key store to create KMS keys that use key material in your AWS CloudHSM cluster, or key material in an external key manager that you own and manage outside of AWS.

AWS KMS also supports multi-Region keys, which let you encrypt data in one AWS Region and decrypt it in a different AWS Region.

For information about creating and managing KMS keys, see *Managing keys*. For information about using KMS keys, see the AWS Key Management Service API Reference.

Customer keys and AWS keys

The KMS keys that you create are customer managed keys. AWS services that use KMS keys to encrypt your service resources often create keys for you. KMS keys that AWS services create in your

AWS account are AWS managed keys. KMS keys that AWS services create in a service account are AWS owned keys.

Type of KMS key	Can view KMS key metadata	Can manage KMS key	Used only for my AWS account	Automatic rotation	Pricing
Customer managed key	Yes	Yes	Yes	Optional. Every year (approximately 365 days)	Monthly fee (pro-rated hourly) Per-use fee
AWS managed key	Yes	No	Yes	Required. Every year (approximately 365 days)	No monthly fee Per-use fee (some AWS services pay this fee for you)
AWS owned key	No	No	No	Varies	No fees

AWS services that integrate with AWS KMS differ in their support for KMS keys. Some AWS services encrypt your data by default with an AWS owned key or an AWS managed key. Some AWS services support customer managed keys. Other AWS services support all types of KMS keys to allow you the ease of an AWS owned key, the visibility of an AWS managed key, or the control of a customer managed key. For detailed information about the encryption options that an AWS service offers, see the *Encryption at Rest* topic in the user guide or the developer guide for the service.

Customer managed keys

The KMS keys that you create are *customer managed keys*. Customer managed keys are KMS keys in your AWS account that you create, own, and manage. You have full control over these KMS keys, including establishing and maintaining their key policies, IAM policies, and grants, enabling and disabling them, rotating their cryptographic material, adding tags, creating aliases that refer to the KMS keys, and scheduling the KMS keys for deletion.

Customer managed keys appear on the **Customer managed keys** page of the AWS Management Console for AWS KMS. To definitively identify a customer managed key, use the DescribeKey operation. For customer managed keys, the value of the `KeyManager` field of the `DescribeKey` response is `CUSTOMER`.

You can use your customer managed key in cryptographic operations and audit usage in AWS CloudTrail logs. In addition, many AWS services that integrate with AWS KMS let you specify a customer managed key to protect the data stored and managed for you.

Customer managed keys incur a monthly fee and a fee for use in excess of the free tier. They are counted against the AWS KMS quotas for your account. For details, see AWS Key Management Service Pricing and Quotas.

AWS managed keys

AWS managed keys are KMS keys in your account that are created, managed, and used on your behalf by an AWS service integrated with AWS KMS.

Some AWS services let you choose an AWS managed key or a customer managed key to protect your resources in that service. In general, unless you are required to control the encryption key that protects your resources, an AWS managed key is a good choice. You don't have to create or maintain the key or its key policy, and there's never a monthly fee for an AWS managed key.

You have permission to view the AWS managed keys in your account, view their key policies, and audit their use in AWS CloudTrail logs. However, you cannot change any properties of AWS managed keys, rotate them, change their key policies, or schedule them for deletion. And, you cannot use AWS managed keys in cryptographic operations directly; the service that creates them uses them on your behalf.

AWS managed keys appear on the **AWS managed keys** page of the AWS Management Console for AWS KMS. You can also identify AWS managed keys by their aliases, which have the format `aws/`*`service-name`*, such as `aws/redshift`. To definitively identify an AWS managed keys, use the DescribeKey operation. For AWS managed keys, the value of the `KeyManager` field of the `DescribeKey` response is `AWS`.

All AWS managed keys are automatically rotated every year. You cannot change this rotation schedule.

> **Note**
>
> In May 2022, AWS KMS changed the rotation schedule for AWS managed keys from every three years (approximately 1,095 days) to every year (approximately 365 days).
> New AWS managed keys are automatically rotated one year after they are created, and approximately every year thereafter.
> Existing AWS managed keys are automatically rotated one year after their most recent rotation, and every year thereafter.

There is no monthly fee for AWS managed keys. They can be subject to fees for use in excess of the free tier, but some AWS services cover these costs for you. For details, see the *Encryption at Rest* topic in the user guide or developer guide for the service. For details, see AWS Key Management Service Pricing.

AWS managed keys do not count against resource quotas on the number of KMS keys in each Region of your account. But when used on behalf of a principal in your account, the KMS keys count against request quotas. For details, see Quotas.

AWS owned keys

AWS owned keys are a collection of KMS keys that an AWS service owns and manages for use in multiple AWS accounts. Although AWS owned keys are not in your AWS account, an AWS service can use an AWS owned key to protect the resources in your account.

Some AWS services let you choose an AWS owned key or a customer managed key. In general, unless you are required to audit or control the encryption key that protects your resources, an AWS owned key is a good choice. AWS owned keys are completely free of charge (no monthly fees or usage fees), they do not count against the AWS KMS quotas for your account, and they're easy to use. You don't need to create or maintain the key or its key policy.

The rotation of AWS owned keys varies across services. For information about the rotation of a particular AWS owned key, see the *Encryption at Rest* topic in the user guide or developer guide for the service.

Symmetric encryption KMS keys

When you create an AWS KMS key, by default, you get a KMS key for symmetric encryption. This is the basic and most commonly used type of KMS key.

In AWS KMS, a *symmetric encryption KMS key* represents a 256-bit AES-GCM encryption key, except in China Regions, where it represents a 128-bit SM4 encryption key. Symmetric key material never leaves AWS KMS unencrypted. To use a symmetric encryption KMS key, you must call AWS KMS. Symmetric encryption keys are used in symmetric encryption, where the same key is used for encryption and decryption. Unless your task explicitly requires asymmetric encryption, symmetric encryption KMS keys, which never leave AWS KMS unencrypted, are a good choice.

AWS services that are integrated with AWS KMS use only symmetric encryption KMS keys to encrypt your data. These services do not support encryption with asymmetric KMS keys. For help determining whether a KMS key is symmetric or asymmetric, see Identifying asymmetric KMS keys.

Technically, the key spec for a symmetric key is SYMMETRIC_DEFAULT, the key usage is ENCRYPT_DECRYPT, and the encryption algorithm is SYMMETRIC_DEFAULT. For details, see SYMMETRIC_DEFAULT key spec.

You can use a symmetric encryption KMS key in AWS KMS to encrypt, decrypt, and re-encrypt data, and generate data keys and data key pairs. You can create multi-Region symmetric encryption KMS keys, import your own key material into a symmetric encryption KMS key, and create symmetric encryption KMS keys in custom key stores. For a table comparing the operations that you can perform on KMS keys of different types, see Key type reference.

Asymmetric KMS keys

You can create asymmetric KMS keys in AWS KMS. An *asymmetric KMS key* represents a mathematically related public key and private key pair. The private key never leaves AWS KMS unencrypted. To use the private key, you must call AWS KMS. You can use the public key within AWS KMS by calling the AWS KMS API operations, or you can download the public key and use it outside of AWS KMS. You can also create multi-Region asymmetric KMS keys.

You can create asymmetric KMS keys that represent RSA key pairs, elliptic curve key pairs, or SM2 key pairs (China Regions only). KMS keys with RSA key pairs can be used to encrypt or decrypt data or sign and verify messages (but not both). KMS keys with NIST-recommended elliptic curve key pairs can be used to sign and verify messages or derive shared secrets (but not both). KMS keys with ECC_SECG_P256K1 key pairs can be used only to sign and verify messages. KMS keys with SM2 (China Regions only) key pairs can be used to either encrypt and decrypt data, sign and verify messages, or derive shared secrets (you must choose one key usage type).

For more information about creating and using asymmetric KMS keys, see Asymmetric keys in AWS KMS.

HMAC KMS keys

An *HMAC KMS key* represents a symmetric key of varying length that is used to generate and verify hash-based message authentication codes (HMAC). The key material for an HMAC key never leaves AWS KMS unencrypted. To use an HMAC key, call the `GenerateMac` or `VerifyMac` API operations.

You can also create multi-Region HMAC KMS keys.

For more information about creating and using HMAC KMS keys, see HMAC keys in AWS KMS.

Data keys

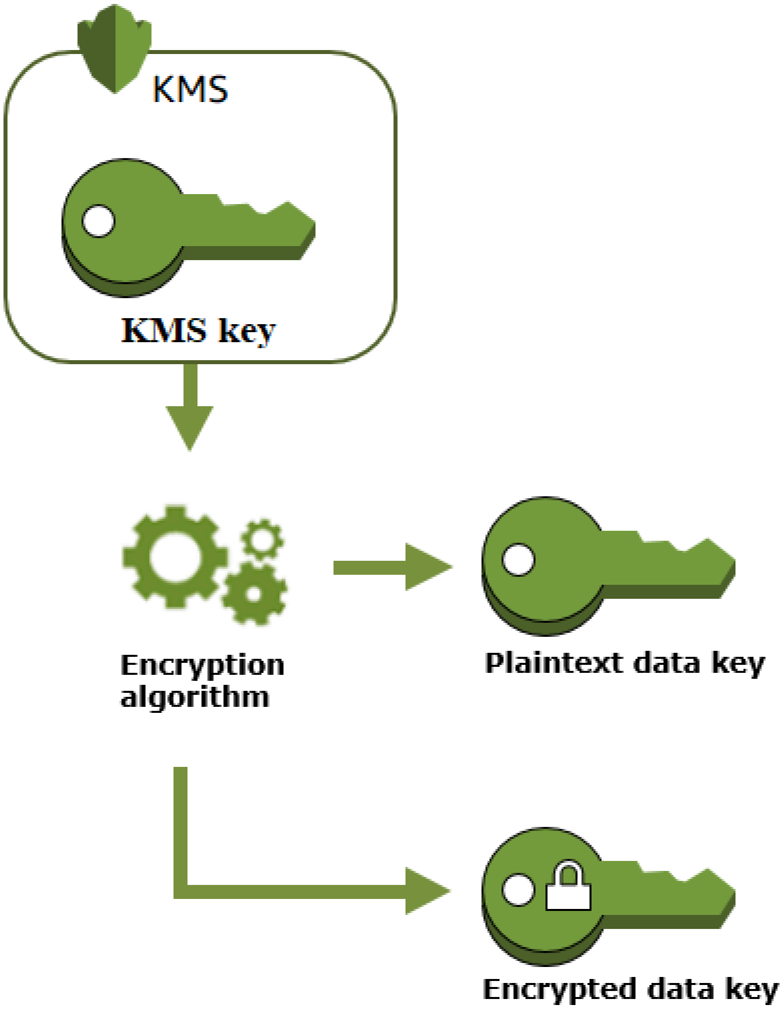
Data keys are symmetric keys you can use to encrypt data, including large amounts of data and other data encryption keys. Unlike symmetric KMS keys, which can't be downloaded, data keys are returned to you for use outside of AWS KMS.

When AWS KMS generates data keys, it returns a plaintext data key for immediate use (optional) and an encrypted copy of the data key that you can safely store with the data. When you are ready to decrypt the data, you first ask AWS KMS to decrypt the encrypted data key.

AWS KMS generates, encrypts, and decrypts data keys. However, AWS KMS does not store, manage, or track your data keys, or perform cryptographic operations with data keys. You must use and manage data keys outside of AWS KMS. For help using the data keys securely, see the AWS Encryption SDK.

Create a data key

To create a data key, call the GenerateDataKey operation. AWS KMS generates the data key. Then it encrypts a copy of the data key under a symmetric encryption KMS key that you specify. The operation returns a plaintext copy of the data key and the copy of the data key encrypted under the KMS key. The following image shows this operation.

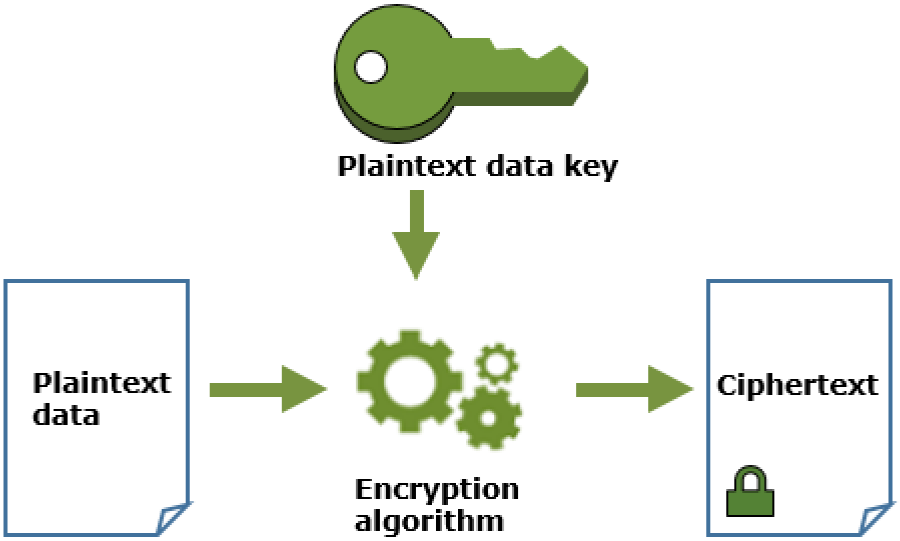


AWS KMS also supports the GenerateDataKeyWithoutPlaintext operation, which returns only an encrypted data key. When you need to use the data key, ask AWS KMS to decrypt it.

Encrypt data with a data key

AWS KMS cannot use a data key to encrypt data. But you can use the data key outside of AWS KMS, such as by using OpenSSL or a cryptographic library like the AWS Encryption SDK.

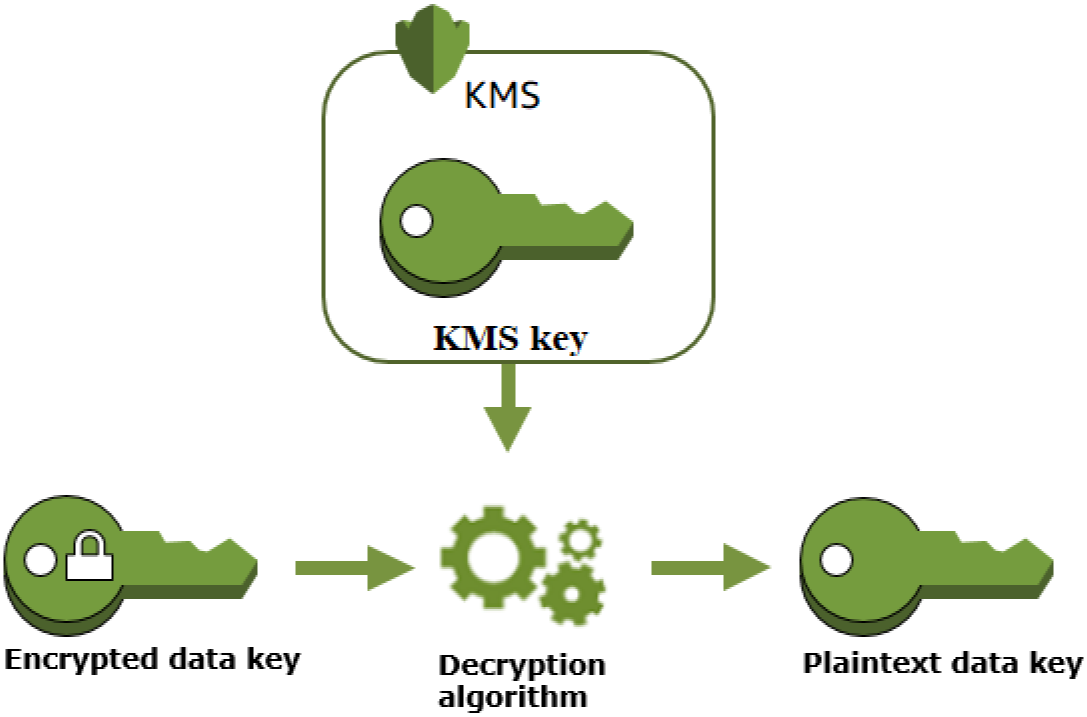
After using the plaintext data key to encrypt data, remove it from memory as soon as possible. You can safely store the encrypted data key with the encrypted data so it is available to decrypt the data.



Decrypt data with a data key

To decrypt your data, pass the encrypted data key to the Decrypt operation. AWS KMS uses your KMS key to decrypt the data key and then returns the plaintext data key. Use the plaintext data key to decrypt your data and then remove the plaintext data key from memory as soon as possible.

The following diagram shows how to use the Decrypt operation to decrypt an encrypted data key.



How unusable KMS keys affect data keys

When a KMS key becomes unusable, the effect is almost immediate (subject to eventual consistency). The key state of the KMS key changes to reflect its new condition, and all requests to use the KMS key in cryptographic operations fail.

However, the effect on data keys encrypted by the KMS key, and on data encrypted by the data key, is delayed until the KMS key is used again, such as to decrypt the data key.

KMS keys can become unusable for a variety of reasons, including the following actions that you might perform.

- Disabling the KMS key
- Scheduling the KMS key for deletion
- Deleting the key material from a KMS key with imported key material, or allowing the imported key material to expire.
- Disconnecting the AWS CloudHSM key store that hosts the KMS key, or deleting the key from the AWS CloudHSM cluster that serves as key material for the KMS key.
- Disconnecting the external key store that hosts the KMS key, or any other action that interferes with encryption and decryption requests to the external key store proxy, including deleting the external key from its external key manager.

This effect is particularly important for the many AWS services that use data keys to protect the resources that the service manages. The following example uses Amazon Elastic Block Store (Amazon EBS) and Amazon Elastic Compute Cloud (Amazon EC2). Different AWS services use data keys in different ways. For details, see the Data protection section of the Security chapter for the AWS service.

For example, consider this scenario:

1. You create an encrypted EBS volume and specify a KMS key to protect it. Amazon EBS asks AWS KMS to use your KMS key to generate an encrypted data key for the volume. Amazon EBS stores the encrypted data key with the volume's metadata.
2. When you attach the EBS volume to an EC2 instance, Amazon EC2 uses your KMS key to decrypt the EBS volume's encrypted data key. Amazon EC2 uses the data key in the Nitro hardware, which is responsible for encrypting all disk I/O to the EBS volume. The data key persists in the Nitro hardware while the EBS volume is attached to the EC2 instance.

3. You perform an action that makes the KMS key unusable. This has no immediate effect on the EC2 instance or the EBS volume. Amazon EC2 uses the data key—not the KMS key—to encrypt all disk I/O while the volume is attached to the instance.
4. However, when the encrypted EBS volume is detached from the EC2 instance, Amazon EBS removes the data key from the Nitro hardware. The next time the encrypted EBS volume is attached to an EC2 instance, the attachment fails, because Amazon EBS cannot use the KMS key to decrypt the volume's encrypted data key. To use the EBS volume again, you must make the KMS key usable again.

Data key pairs

Data key pairs are asymmetric data keys consisting of a mathematically-related public key and private key. They are designed for use in client-side encryption and decryption or signing and verification outside of AWS KMS.

Unlike the data key pairs that tools like OpenSSL generate, AWS KMS protects the private key in each data key pair under a symmetric encryption KMS key in AWS KMS that you specify. However, AWS KMS does not store, manage, or track your data key pairs, or perform cryptographic operations with data key pairs. You must use and manage data key pairs outside of AWS KMS.

AWS KMS supports the following types of data key pairs:

- RSA key pairs: RSA_2048, RSA_3072, and RSA_4096
- Elliptic curve key pairs: ECC_NIST_P256, ECC_NIST_P384, ECC_NIST_P521, and ECC_SECG_P256K1
- SM key pairs (China Regions only): SM2

The type of data key pair that you select usually depends on your use case or regulatory requirements. Most certificates require RSA keys. Elliptic curve keys are often used for digital signatures or deriving shared secrets. ECC_SECG_P256K1 keys are commonly used for cryptocurrencies. AWS KMS recommends that you use ECC key pairs for signing, and use RSA key pairs for either encryption or signing, but not both. However, AWS KMS cannot enforce any restrictions on the use of data key pairs outside of AWS KMS.

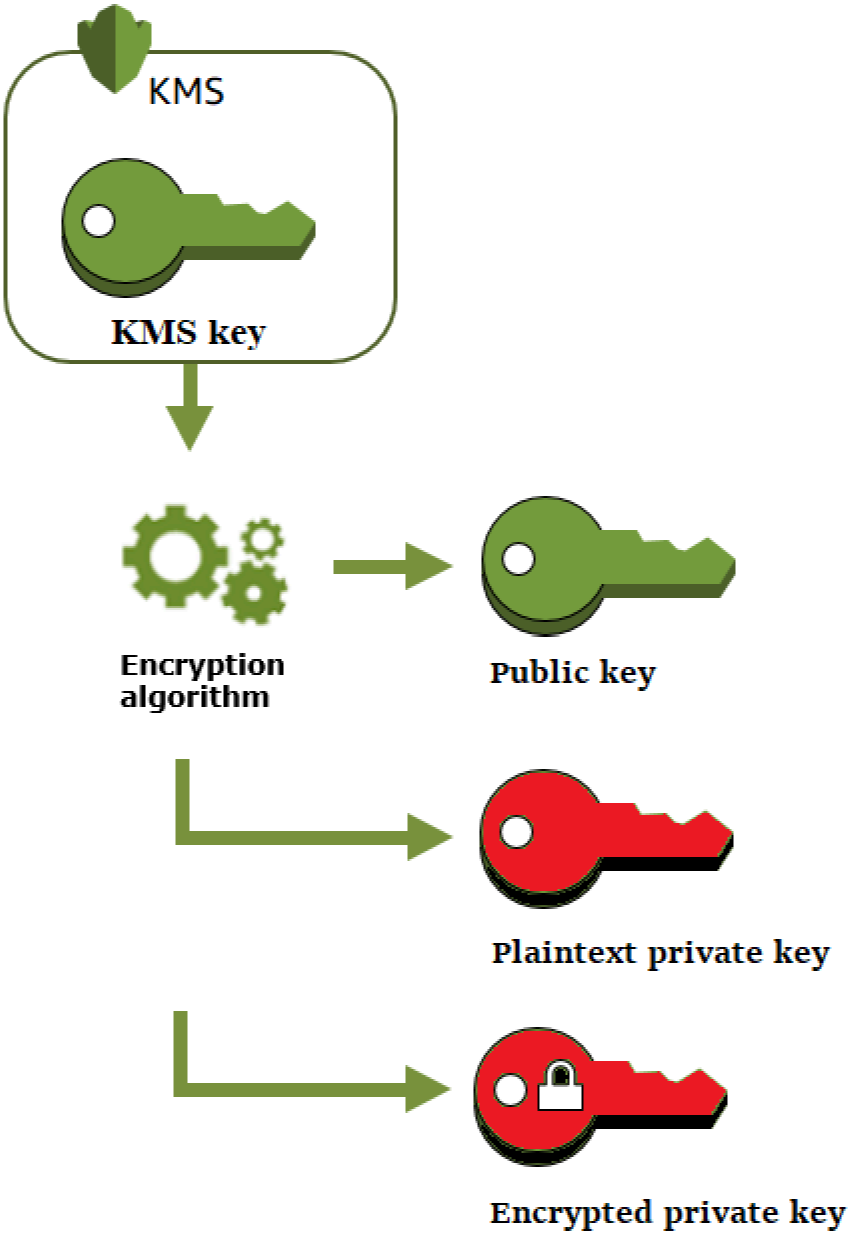
Create a data key pair

To create a data key pair, call the GenerateDataKeyPair or GenerateDataKeyPairWithoutPlaintext operations. Specify the symmetric encryption KMS key you want to use to encrypt the private key.

`GenerateDataKeyPair` returns a plaintext public key, a plaintext private key, and an encrypted private key. Use this operation when you need a plaintext private key immediately, such as to generate a digital signature.

`GenerateDataKeyPairWithoutPlaintext` returns a plaintext public key and an encrypted private key, but not a plaintext private key. Use this operation when you don't need a plaintext private key immediately, such as when you're encrypting with a public key. Later, when you need a plaintext private key to decrypt the data, you can call the Decrypt operation.

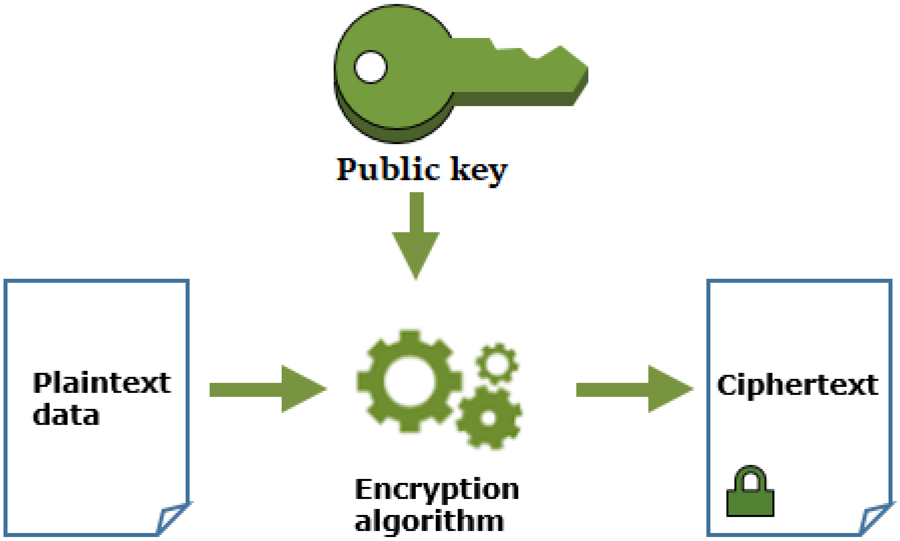
The following image shows the `GenerateDataKeyPair` operation. The `GenerateDataKeyPairWithoutPlaintext` operation omits the plaintext private key.



Encrypt data with a data key pair

When you encrypt with a data key pair, you use the public key of the pair to encrypt the data and the private key of the same pair to decrypt the data. Typically, you use data key pairs when many parties need to encrypt data that only the party with the private key can decrypt.

The parties with the public key use that key to encrypt data, as shown in the following diagram.

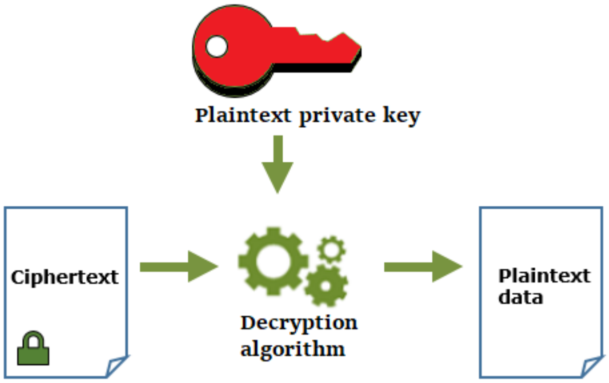


Decrypt data with a data key pair

To decrypt your data, use the private key in the data key pair. For the operation to succeed, the public and private keys must be from the same data key pair, and you must use the same encryption algorithm.

To decrypt the encrypted private key, pass it to the Decrypt operation. Use the plaintext private key to decrypt the data. Then remove the plaintext private key from memory as soon as possible.

The following diagram shows how to use the private key in a data key pair to decrypt ciphertext.



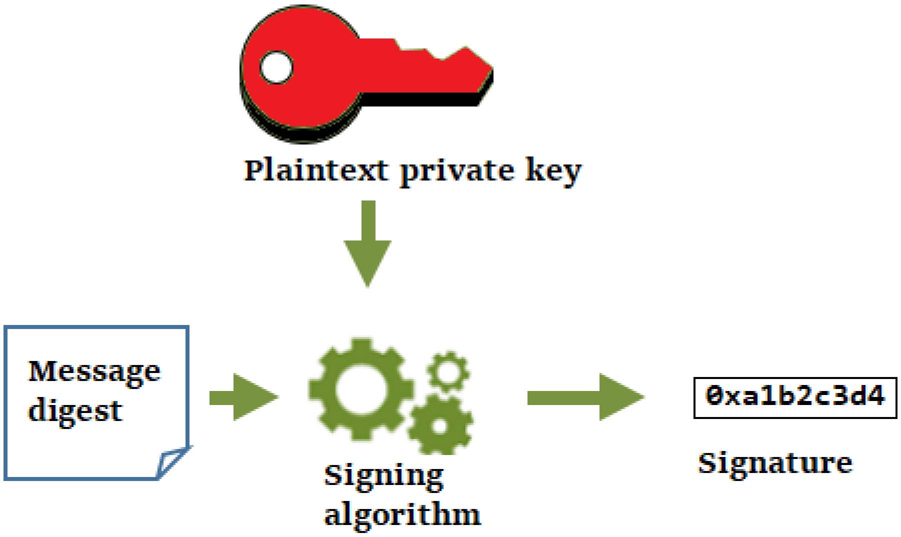
Sign messages with a data key pair

To generate a cryptographic signature for a message, use the private key in the data key pair. Anyone with the public key can use it to verify that the message was signed with your private key and that it has not changed since it was signed.

If you encrypt your private key, pass the encrypted private key to the Decrypt operation. AWS KMS uses your KMS key to decrypt the data key and then it returns the plaintext private key. Use the plaintext private key to generate the signature. Then remove the plaintext private key from memory as soon as possible.

To sign a message, create a message digest using a cryptographic hash function, such as the `dgst` command in OpenSSL. Then, pass your plaintext private key to the signing algorithm. The result is a signature that represents the contents of the message. (You might be able to sign shorter messages without first creating a digest. The maximum message size varies with the signing tool you use.)

The following diagram shows how to use the private key in a data key pair to sign a message.

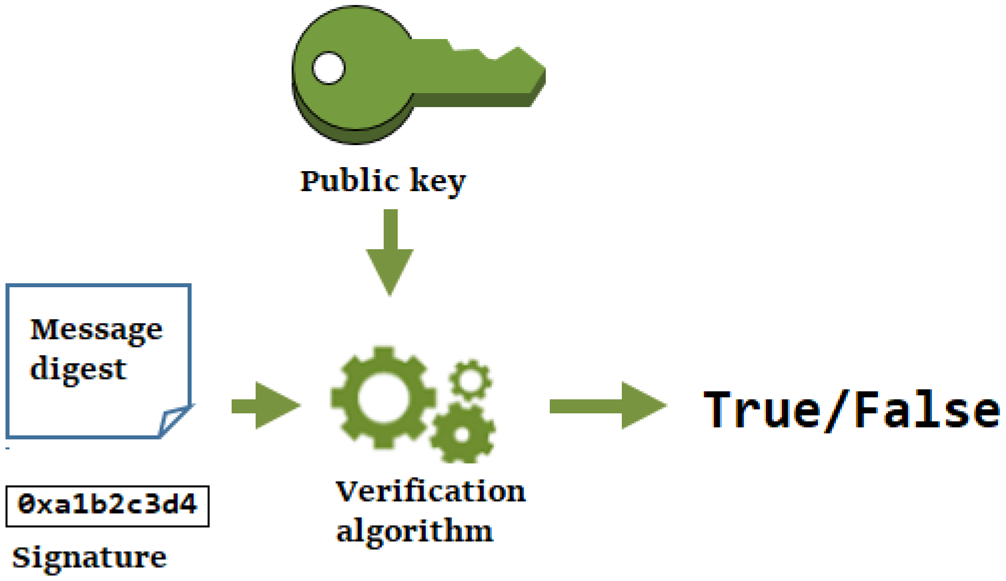


Verify a signature with a data key pair

Anyone who has the public key in your data key pair can use it to verify the signature that you generated with your private key. Verification confirms that an authorized user signed the message with the specified private key and signing algorithm, and the message hasn't changed since it was signed.

To be successful, the party verifying the signature must generate the same type of digest, use the same algorithm, and use the public key that corresponds to the private key used to sign the message.

The following diagram shows how to use the public key in a data key pair to verify a message signature.



Derive a shared secret with data key pairs

Key agreement enables two peers, each having an elliptic-curve public–private key pair, to establish a shared secret over an insecure channel. To derive a shared secret, the two peers must exchange their public keys over the insecure communication channel (like the internet). Then, each party uses their *private key* and their peer's *public key* to calculate the same shared secret using a key agreement algorithm. You can use the shared secret value to derive a symmetric key that can encrypt and decrypt data that is sent between the two peers, or that can generate and verify HMACs.

Note

AWS KMS strongly recommends verifying that the public key you receive came from the expected party before using it to derive a shared secret.

Aliases

Use an *alias* as a friendly name for a KMS key. For example, you can refer to a KMS key as *test-key* instead of 1234abcd-12ab-34cd-56ef-1234567890ab.

Aliases make it easier to identify a KMS key in the AWS Management Console. You can use an alias to identify a KMS key in some AWS KMS operations, including cryptographic operations. In applications, you can use a single alias to refer to different KMS keys in each AWS Region.

You can also allow and deny access to KMS keys based on their aliases without editing policies or managing grants. This feature is part of AWS KMS support for attribute-based access control (ABAC). For details, see ABAC for AWS KMS.

In AWS KMS, aliases are independent resources, not properties of a KMS key. As such, you can add, change, and delete an alias without affecting the associated KMS key.

> ⚠ **Important**
>
> Do not include confidential or sensitive information in an alias name. Aliases may appear in plaintext in CloudTrail logs and other output.

Learn more:

- For detailed information about aliases, see Using aliases.
- For information about the formats of key identifiers, including aliases, see Key identifiers (KeyId).
- For help finding the aliases associated with a KMS key, see Finding the alias name and alias ARN
- For examples of creating and managing aliases in multiple programming languages, see Working with aliases.

Custom key stores

A *custom key store* is an AWS KMS resource backed by a key manager outside of AWS KMS that you own and manage. When you use a KMS key in a custom key store for a cryptographic operation, the cryptographic operation is actually performed in your key manager using its cryptographic keys.

AWS KMS supports AWS CloudHSM key stores backed by an AWS CloudHSM cluster and external key stores that are backed by an external key manager outside of AWS.

For more information, see Custom key stores.

Cryptographic operations

In AWS KMS, *cryptographic operations* are API operations that use KMS keys to protect data. Because KMS keys remain within AWS KMS, you must call AWS KMS to use a KMS key in a cryptographic operation.

To perform cryptographic operations with KMS keys, use the AWS SDKs, AWS Command Line Interface (AWS CLI), or the AWS Tools for PowerShell. You cannot perform cryptographic operations in the AWS KMS console. For examples of calling the cryptographic operations in several programming languages, see Programming the AWS KMS API.

The following table lists the AWS KMS cryptographic operations. It also shows the key type and key usage requirements for KMS keys used in the operation.

Operation	Key type	Key usage
Decrypt	Symmetric or asymmetric	ENCRYPT_DECRYPT
DeriveSharedSecret	Asymmetric	KEY_AGREEMENT
Encrypt	Symmetric or asymmetric	ENCRYPT_DECRYPT
GenerateDataKey	Symmetric	ENCRYPT_DECRYPT
GenerateDataKeyPair	Symmetric [1] Not supported on KMS keys in custom key stores.	ENCRYPT_DECRYPT
GenerateDataKeyPairWithoutPlaintext	Symmetric [1] Not supported on KMS keys in custom key stores.	ENCRYPT_DECRYPT

Operation	Key type	Key usage
GenerateDataKeyWithoutPlaintext	Symmetric	ENCRYPT_DECRYPT
GenerateMac	HMAC	GENERATE_ VERIFY_MAC
GenerateRandom	N/A. This operation doesn't use a KMS key.	N/A
ReEncrypt	Symmetric or asymmetric	ENCRYPT_DECRYPT
Sign	Asymmetric	SIGN_VERIFY
Verify	Asymmetric	SIGN_VERIFY
VerifyMac	HMAC	GENERATE_ VERIFY_MAC

[1] Generates an asymmetric data key pair that is protected by a symmetric encryption KMS key.

For information about the permissions for cryptographic operations, see the the section called "Permissions reference".

To make AWS KMS responsive and highly functional for all users, AWS KMS establishes quotas on number of cryptographic operations called in each second. For details, see the section called "Shared quotas for cryptographic operations".

Key identifiers (KeyId)

Key identifiers act like names for your KMS keys. They help you to recognize your KMS keys in the console. You use them to indicate which KMS keys you want to use in AWS KMS API operations, key policies, IAM policies, and grants. The key identifier values are completely unrelated to the key material associated with the KMS key.

AWS KMS defines several key identifiers. When you create a KMS key, AWS KMS generates a key ARN and key ID, which are properties of the KMS key. When you create an alias, AWS KMS generates an alias ARN based on the alias name that you define. You can view the key and alias identifiers in the AWS Management Console and in the AWS KMS API.

In the AWS KMS console, you can view and filter KMS keys by their key ARN, key ID, or alias name, and sort by key ID and alias name. For help finding the key identifiers in the console, see the section called “Finding the key ID and key ARN”.

In the AWS KMS API, the parameters you use to identify a KMS key are named `KeyId` or a variation, such as `TargetKeyId` or `DestinationKeyId`. However, the values of those parameters are not limited to key IDs. Some can take any valid key identifier. For information about the values for each parameter, see the parameter description in the AWS Key Management Service API Reference.

> **Note**
>
> When using the AWS KMS API, be careful about the key identifier that you use. Different APIs require different key identifiers. In general, use the most complete and practical key identifier for your task.

AWS KMS supports the following key identifiers.

Key ARN

The key ARN is the Amazon Resource Name (ARN) of a KMS key. It is a unique, fully qualified identifier for the KMS key. A key ARN includes the AWS account, Region, and the key ID. For help finding the key ARN of a KMS key, see the section called “Finding the key ID and key ARN”.

The format of a key ARN is as follows:

```
arn:<partition>:kms:<region>:<account-id>:key/<key-id>
```

The following is an example key ARN for a single-Region KMS key.

```
arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

The *`key-id`* element of the key ARNs of multi-Region keys begin with the `mrk-` prefix. The following is an example key ARN for a multi-Region key.

```
arn:aws:kms:us-west-2:111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab
```

Key ID

The key ID uniquely identifies a KMS key within an account and Region. For help finding the key ID of a KMS key, see the section called "Finding the key ID and key ARN".

The following is an example key ID for a single-Region KMS key.

```
1234abcd-12ab-34cd-56ef-1234567890ab
```

The key IDs of multi-Region keys begin with the `mrk-` prefix. The following is an example key ID for a multi-Region key.

```
mrk-1234abcd12ab34cd56ef1234567890ab
```

Alias ARN

The alias ARN is the Amazon Resource Name (ARN) of an AWS KMS alias. It is a unique, fully qualified identifier for the alias, and for the KMS key it represents. An alias ARN includes the AWS account, Region, and the alias name.

At any given time, an alias ARN identifies one particular KMS key. However, because you can change the KMS key associated with the alias, the alias ARN can identify different KMS keys at different times. For help finding the alias ARN of a KMS key, see Finding the alias name and alias ARN.

The format of an alias ARN is as follows:

```
arn:<partition>:kms:<region>:<account-id>:alias/<alias-name>
```

The following is the alias ARN for a fictitious `ExampleAlias`.

```
arn:aws:kms:us-west-2:111122223333:alias/ExampleAlias
```

Alias name

The alias name is a string of up to 256 characters. It uniquely identifies an associated KMS key within an account and Region. In the AWS KMS API, alias names always begin with `alias/`. For help finding the alias name of a KMS key, see Finding the alias name and alias ARN.

The format of an alias name is as follows:

```
alias/<alias-name>
```

For example:

```
alias/ExampleAlias
```

The aws/ prefix for an alias name is reserved for AWS managed keys. You cannot create an alias with this prefix. For example, the alias name of the AWS managed key for Amazon Simple Storage Service (Amazon S3) is the following.

```
alias/aws/s3
```

Key material

Key material is the string of bits used in a cryptographic algorithm. Secret key material must be kept secret to protect the cryptographic operations that use it. Public key material is designed to be shared.

Each KMS key includes a reference to its key material in its metadata. The key material origin of symmetric encryption KMS keys can vary. You can use key material that AWS KMS generates, key material that is generated in the AWS CloudHSM cluster of a custom key store, or import your own key material. If you use AWS KMS key material for your symmetric encryption KMS key, you can enable automatic rotation of your key material.

By default, each KMS key has unique key material. However, you can create a set of multi-Region keys with the same key material.

Key material origin

Key material origin is a KMS key property that identifies the source of the key material in the KMS key. You choose the key material origin when you create the KMS key, and you cannot change it. The source of the key material affects the security, durability, availability, latency, and throughput characteristics of the KMS key.

To find the key material origin of a KMS key, use the DescribeKey operation, or see the **Origin** value on the **Cryptographic configuration** tab of the detail page for a KMS key in the AWS KMS console. For help, see Viewing Keys.

KMS keys can have one of the following key material origin values.

AWS_KMS

AWS KMS creates and manages the key material for the KMS key in its own key store. This is the default and the recommended value for most KMS keys.

For help creating keys with key material from AWS KMS, see Creating keys.

EXTERNAL (Import key material)

The KMS key has imported key material. When you create a KMS key with an `External` key material origin, the KMS key has no key material. Later, you can import key material into the KMS key. When you use imported key material, you need to secure and manage that key material outside of AWS KMS, including replacing the key material if it expires. For details, see About imported key material.

For help creating a KMS key for imported key material, see Step 1: Create a KMS key with no key material.

AWS_CLOUDHSM

AWS KMS creates the key material in the AWS CloudHSM cluster for your AWS CloudHSM key store.

For help creating a KMS key in an AWS CloudHSM key store, see Creating KMS keys in an AWS CloudHSM key store.

EXTERNAL_KEY_STORE

The key material is a cryptographic key in an external key manager outside of AWS. This origin is supported only for KMS keys in an external key store.

For help creating a KMS key in an external key store, see Creating KMS keys in an external key store.

Key spec

Key spec is a property that represents the cryptographic configuration of a key. The meaning of the key spec differs with the key type.

- AWS KMS keys — The *key spec* determines whether the KMS key is symmetric or asymmetric. It also determines the type of its key material, and the algorithms it supports. You choose the key spec when you create the KMS key, and you cannot change it. The default key spec, SYMMETRIC_DEFAULT, represents a 256-bit symmetric encryption key.

> **Note**
>
> The KeySpec for a KMS key was known as a CustomerMasterKeySpec. The CustomerMasterKeySpec parameter of the CreateKey operation is deprecated. Instead, use the KeySpec parameter, which works the same way. To prevent breaking changes, the response of the CreateKey and DescribeKey operations now includes both KeySpec and CustomerMasterKeySpec members with the same values.

For a list of key specs and help with choosing a key spec, see Selecting the key spec. To find the key spec of a KMS key, use the DescribeKey operation, or see the **Cryptographic configuration** tab on the detail page for a KMS key in the AWS KMS console. For help, see Viewing Keys.

To limit the key specs that principals can use when creating KMS keys, use the kms:KeySpec condition key. You can also use the kms:KeySpec condition key to allow principals to call AWS KMS operations only on KMS keys with a particular key spec. For example, you can deny permission to schedule deletion of any KMS key with an RSA_4096 key spec.

- Data keys (GenerateDataKey) — The *key spec* determines the length of an AES data key.
- Data keys pairs (GenerateDataKeyPair) — The *key pair spec* determines the type of key material in the data key pair.

Key usage

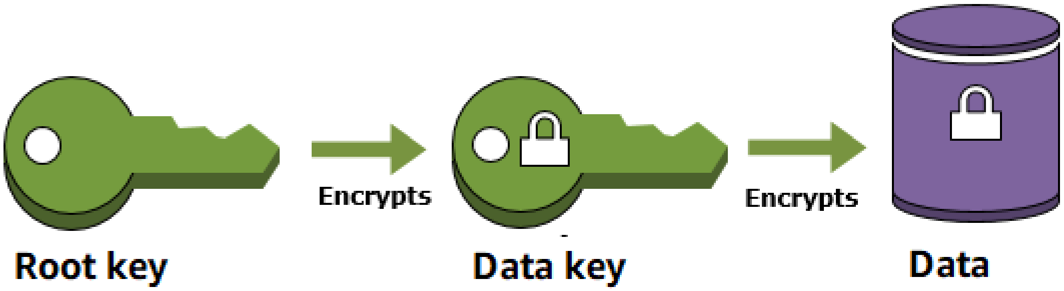
Key usage is a property that determines the cryptographic operations the key supports. KMS keys can have a key usage of ENCRYPT_DECRYPT, SIGN_VERIFY, GENERATE_VERIFY_MAC, or KEY_AGREEMENT. Each KMS key can have only one key usage. Using a KMS key for more than one type of operation makes the product of both operations more vulnerable to attack.

For help choosing the key usage for your KMS key, see Selecting the key usage. To find the key usage of a KMS key, use the DescribeKey operation, or choose the **Cryptographic configuration** tab on the detail page for a KMS key in the AWS KMS console. For help, see Viewing Keys.

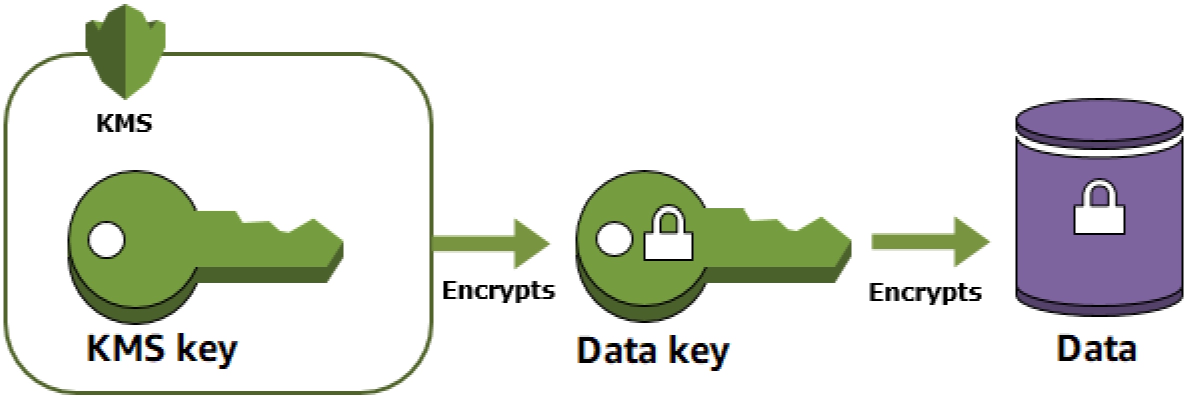
Envelope encryption

When you encrypt your data, your data is protected, but you have to protect your encryption key. One strategy is to encrypt it. *Envelope encryption* is the practice of encrypting plaintext data with a data key, and then encrypting the data key under another key.

You can even encrypt the data encryption key under another encryption key, and encrypt that encryption key under another encryption key. But, eventually, one key must remain in plaintext so you can decrypt the keys and your data. This top-level plaintext key encryption key is known as the *root key*.



AWS KMS helps you to protect your encryption keys by storing and managing them securely. Root keys stored in AWS KMS, known as AWS KMS keys, never leave the AWS KMS FIPS validated hardware security modules unencrypted. To use a KMS key, you must call AWS KMS.



Envelope encryption offers several benefits:

- **Protecting data keys**

When you encrypt a data key, you don't have to worry about storing the encrypted data key, because the data key is inherently protected by encryption. You can safely store the encrypted data key alongside the encrypted data.

- **Encrypting the same data under multiple keys**

  Encryption operations can be time consuming, particularly when the data being encrypted are large objects. Instead of re-encrypting raw data multiple times with different keys, you can re-encrypt only the data keys that protect the raw data.

- **Combining the strengths of multiple algorithms**

  In general, symmetric key algorithms are faster and produce smaller ciphertexts than public key algorithms. But public key algorithms provide inherent separation of roles and easier key management. Envelope encryption lets you combine the strengths of each strategy.

Encryption context

All AWS KMS cryptographic operations with symmetric encryption KMS keys accept an *encryption context*, an optional set of non-secret key–value pairs that can contain additional contextual information about the data. AWS KMS uses the encryption context as additional authenticated data (AAD) to support authenticated encryption.

When you include an encryption context in an encryption request, it is cryptographically bound to the ciphertext such that the same encryption context is required to decrypt (or decrypt and re-encrypt) the data. If the encryption context provided in the decryption request is not an exact, case-sensitive match, the decrypt request fails. Only the order of the key-value pairs in the encryption context can vary.

> **Note**
>
> You cannot specify an encryption context in a cryptographic operation with an asymmetric KMS key or an HMAC KMS key. Asymmetric algorithms and MAC algorithms do not support an encryption context.

The encryption context is not secret and not encrypted. It appears in plaintext in AWS CloudTrail Logs so you can use it to identify and categorize your cryptographic operations. Your encryption context should not include sensitive information. We recommend that your encryption context describe the data being encrypted or decrypted. For example, when you encrypt a file, you might use part of the file path as encryption context.

```
"encryptionContext": {
    "department": "10103.0"
}
```

For example, when encrypting volumes and snapshots created with the Amazon Elastic Block Store (Amazon EBS) CreateSnapshot operation, Amazon EBS uses the volume ID as encryption context value.

```
"encryptionContext": {
  "aws:ebs:id": "vol-abcde12345abc1234"
}
```

You can also use the encryption context to refine or limit access to AWS KMS keys in your account. You can use the encryption context as a constraint in grants and as a condition in policy statements.

To learn how to use encryption context to protect the integrity of encrypted data, see the post How to Protect the Integrity of Your Encrypted Data by Using AWS Key Management Service and EncryptionContext on the AWS Security Blog.

More about encryption context.

Encryption context rules

AWS KMS enforces the following rules for encryption context keys and values.

- The key and value in an encryption context pair must be simple literal strings. If you use a different type, such as an integer or float, AWS KMS interprets it as a string.
- The keys and values in an encryption context can include Unicode characters. If an encryption context includes characters that are not permitted in key policies or IAM policies, you won't be able to specify the encryption context in policy condition keys, such as `kms:EncryptionContext:`*`context-key`* and `kms:EncryptionContextKeys`. For details about key policy document rules, see Key policy format. For details about IAM policy document rules, see IAM name requirements in the *IAM User Guide*.

Encryption context in policies

The encryption context is used primarily to verify integrity and authenticity. But you can also use the encryption context to control access to symmetric encryption AWS KMS keys in key policies and IAM policies.

The kms:EncryptionContext: and kms:EncryptionContextKeys condition keys allow (or deny) a permission only when the request includes particular encryption context keys or key–value pairs.

For example, the following key policy statement allows the `RoleForExampleApp` role to use the KMS key in `Decrypt` operations. It uses the `kms:EncryptionContext:`*`context-key`* condition key to allow this permission only when the encryption context in the request includes an `AppName:ExampleApp` encryption context pair.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
  },
  "Action": "kms:Decrypt",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:EncryptionContext:AppName": "ExampleApp"
    }
  }
}
```

For more information about these encryption context condition keys, see Condition keys for AWS KMS.

Encryption context in grants

When you create a grant, you can include grant constraints that establish conditions for the grant permissions. AWS KMS supports two grant constraints, `EncryptionContextEquals` and `EncryptionContextSubset`, both of which involve the encryption context in a request for a cryptographic operation. When you use these grant constraints, the permissions in the grant are effective only when the encryption context in the request for the cryptographic operation satisfies the requirements of the grant constraints.

For example, you can add an `EncryptionContextEquals` grant constraint to a grant that allows the GenerateDataKey operation. With this constraint, the grant allows the operation only when the encryption context in the request is a case-sensitive match for the encryption context in the grant constraint.

```
$ aws kms create-grant \
    --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
    --grantee-principal arn:aws:iam::111122223333:user/exampleUser \
    --retiring-principal arn:aws:iam::111122223333:role/adminRole \
    --operations GenerateDataKey \
    --constraints EncryptionContextEquals={Purpose=Test}
```

A request like the following from the grantee principal would satisfy the `EncryptionContextEquals` constraint.

```
$ aws kms generate-data-key \
    --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
    --key-spec AES_256 \
    --encryption-context Purpose=Test
```

For details about the grant constraints, see Using grant constraints. For detailed information about grants, see the section called “Grants”.

Logging encryption context

AWS KMS uses AWS CloudTrail to log the encryption context so you can determine which KMS keys and data have been accessed. The log entry shows exactly which KMS keys was used to encrypt or decrypt specific data referenced by the encryption context in the log entry.

> **Important**
>
> Because the encryption context is logged, it must not contain sensitive information.

Storing encryption context

To simplify use of any encryption context when you call the `Decrypt` or `ReEncrypt` operations, you can store the encryption context alongside the encrypted data. We recommend that you store only enough of the encryption context to help you create the full encryption context when you need it for encryption or decryption.

For example, if the encryption context is the fully qualified path to a file, store only part of that path with the encrypted file contents. Then, when you need the full encryption context, reconstruct it from the stored fragment. If someone tampers with the file, such as renaming it or moving it to a different location, the encryption context value changes and the decryption request fails.

Key policy

When you create a KMS key, you determine who can use and manage that KMS key. These permissions are contained in a document called the *key policy*. You can use the key policy to add, remove, or change permissions at any time for a customer managed keys. But you cannot edit the key policy for an AWS managed keys. For more information, see Key policies in AWS KMS.

Grant

A *grant* is a policy instrument that allows AWS principals to use AWS KMS keys in cryptographic operations. It also can let them view a KMS key (DescribeKey) and create and manage grants. When authorizing access to a KMS key, grants are considered along with key policies and IAM policies. Grants are often used for temporary permissions because you can create one, use its permissions, and delete it without changing your key policies or IAM policies. Because grants can be very specific, and are easy to create and revoke, they are often used to provide temporary permissions or more granular permissions.

For detailed information about grants, including grant terminology, see Grants in AWS KMS.

Auditing KMS key usage

You can use AWS CloudTrail to audit key usage. CloudTrail creates log files that contain a history of AWS API calls and related events for your account. These log files include all AWS KMS API requests made with the AWS Management Console, AWS SDKs, and command line tools. The log files also include requests to AWS KMS that AWS services make on your behalf. You can use these log files to find important information, including when the KMS keys was used, the operation that was requested, the identity of the requester, and the source IP address. For more information, see Logging with AWS CloudTrail and the AWS CloudTrail User Guide.

Key management infrastructure

A common practice in cryptography is to encrypt and decrypt with a publicly available and peer-reviewed algorithm such as AES (Advanced Encryption Standard) and a secret key. One of the main problems with cryptography is that it's very hard to keep a key secret. This is typically the job of a key management infrastructure (KMI). AWS KMS operates the key infrastructure for you. AWS KMS creates and securely stores your root keys, called AWS KMS keys. For more information about how AWS KMS operates, see AWS Key Management Service Cryptographic Details.

Managing keys

To get started with AWS KMS, create an AWS KMS key.

The topics in this section explain how to manage the basic KMS key, a symmetric encryption KMS key, from creation to deletion. It includes topics on editing and viewing keys, tagging keys, enabling and disabling keys, rotating key material, and using AWS tools and services to monitor use of your KMS keys. It also includes information about using AWS CloudFormation to create and manage your KMS keys and a key state reference that shows the required key state for each AWS KMS operation.

For information about creating, using, and managing other types of KMS keys, see Special-purpose keys.

Topics

Creating keys

You can create AWS KMS keys in the AWS Management Console, or by using the CreateKey operation or an AWS CloudFormation template. During this process, you pick the type of the KMS key, its regionality (single-Region or multi-Region), and the origin of the key material (by default, AWS KMS creates the key material). You cannot change these properties after the KMS key is created. You also set the key policy for the KMS key, which you can change at any time.

This topic explains how to create the basic KMS key, a symmetric encryption KMS key for a single Region with key material from AWS KMS. You can use this KMS key to protect your resources in an AWS service. For detailed information about symmetric encryption KMS keys, see SYMMETRIC_DEFAULT key spec. For help creating other types of keys, see Special-purpose keys.

If you are creating a KMS key to encrypt data you store or manage in an AWS service, create a symmetric encryption KMS key. AWS services that are integrated with AWS KMS use only symmetric encryption KMS keys to encrypt your data. These services do not support encryption with asymmetric KMS keys. For help deciding which type of KMS key to create, see Choosing a KMS key type.

Note

Symmetric KMS keys are now called *symmetric encryption* KMS keys. AWS KMS supports two kinds of symmetric KMS keys, symmetric encryption KMS keys (the default type) and HMAC KMS keys, which are also symmetric keys.

When you create a KMS key in the AWS KMS console, you are required to give it an alias (friendly name). The `CreateKey` operation does not create an alias for the new KMS key. To create an alias for a new or existing KMS key, use the CreateAlias operation. For detailed information about aliases in AWS KMS, see Using aliases.

This topic explains how to create a symmetric encryption KMS key. Use the following table to find instructions for creating KMS keys of different types.

Instructions for creating a KMS key

KMS key type	Instructions
Symmetric encryption key (SYMMETRIC_DEFAULT)	the section called "Creating symmetric encryption KMS keys"
Asymmetric key	the section called "Creating asymmetric KMS keys"
HMAC key	the section called "Creating HMAC keys"
Multi-Region key (of any type)	the section called "Creating a primary key with imported key material"

KMS key type	Instructions
	the section called “Creating a replica key with imported key material”
Imported key material ("Bring your own key — BYOK")	the section called “Step 1: Create a KMS key with no key material”
AWS CloudHSM key store	the section called “Creating KMS keys in an AWS CloudHSM key store”
External key store ("Hold your own key — HYOK")	the section called “Creating KMS keys in an external key store”

Learn more:

- To create data keys for client-side encryption, use the GenerateDataKey operation.
- To create an asymmetric KMS key for encryption or signing, see Creating asymmetric KMS keys.
- To create an HMAC KMS key, see Creating HMAC KMS keys.
- To create a KMS key with imported key material ("bring your own key"), see Importing key material step 1: Create an AWS KMS key without key material.
- To create a multi-Region primary key or replica key, see Creating multi-Region keys.
- To create a KMS key in a custom key store (key material origin is Custom Key Store (CloudHSM)), see Creating KMS keys in an AWS CloudHSM key store.
- To use an AWS CloudFormation template to create a KMS key, see AWS::KMS::Key in the *AWS CloudFormation User Guide*.
- To determine whether an existing KMS key is symmetric or asymmetric, see Identifying asymmetric KMS keys.
- To use your KMS key programmatically and in command line interface operations, you need a key ID or key ARN. For detailed instructions, see Finding the key ID and key ARN.
- For information about quotas that apply to KMS keys, see Quotas.

Topics

Permissions for creating KMS keys

To create a KMS key in the console or by using the APIs, you must have the following permission in an IAM policy. Whenever possible, use condition keys to limit the permissions. For example, you can use the kms:KeySpec condition key in an IAM policy to allow principals to create only symmetric encryption keys.

For an example of an IAM policy for principals who create keys, see Allow a user to create KMS keys.

> **Note**
>
> Be cautious when giving principals permission to manage tags and aliases. Changing a tag or alias can allow or deny permission to the customer managed key. For details, see ABAC for AWS KMS.

- kms:CreateKey is required.
- kms:CreateAlias is required to create a KMS key in the console where an alias is required for every new KMS key.
- kms:TagResource is required to add tags while creating the KMS key.
- iam:CreateServiceLinkedRole is required to create multi-Region primary keys. For details, see Controlling access to multi-Region keys.

The kms:PutKeyPolicy permission is not required to create the KMS key. The `kms:CreateKey` permission includes permission to set the initial key policy. But you must add this permission to the key policy while creating the KMS key to ensure that you can control access to the KMS key. The alternative is using the BypassLockoutSafetyCheck parameter, which is not recommended.

KMS keys belong to the AWS account in which they were created. The IAM user who creates a KMS key is not considered to be the key owner and they don't automatically have permission to use or manage the KMS key that they created. Like any other principal, the key creator needs to get permission through a key policy, IAM policy, or grant. However, principals who have the `kms:CreateKey` permission can set the initial key policy and give themselves permission to use or manage the key.

Creating symmetric encryption KMS keys

You can create KMS keys in the AWS Management Console or by using the AWS KMS API.

This topic explains how to create the basic KMS key, a symmetric encryption KMS key for a single Region with key material from AWS KMS. You can use this KMS key to protect your resources in an AWS service. For help creating other types of keys, see Special-purpose keys.

Creating symmetric encryption KMS keys (console)

You can use the AWS Management Console to create AWS KMS keys (KMS keys).

> ⚠ **Important**
>
> Do not include confidential or sensitive information in the alias, description, or tags. These fields may appear in plain text in CloudTrail logs and other output.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Choose **Create key**.
5. To create a symmetric encryption KMS key, for **Key type** choose **Symmetric**.

   For information about how to create an asymmetric KMS key in the AWS KMS console, see Creating asymmetric KMS keys (console).
6. In **Key usage**, the **Encrypt and decrypt** option is selected for you.

   For information about how to create KMS keys that generate and verify MAC codes, see Creating HMAC KMS keys.
7. Choose **Next**.

   For information about the **Advanced options**, see Special-purpose keys.
8. Type an alias for the KMS key. The alias name cannot begin with **`aws/`**. The **`aws/`** prefix is reserved by Amazon Web Services to represent AWS managed keys in your account.

> **Note**
>
> Adding, deleting, or updating an alias can allow or deny permission to the KMS key. For details, see ABAC for AWS KMS and Using aliases to control access to KMS keys.

An alias is a display name that you can use to identify the KMS key. We recommend that you choose an alias that indicates the type of data you plan to protect or the application you plan to use with the KMS key.

Aliases are required when you create a KMS key in the AWS Management Console. They are optional when you use the CreateKey operation.

9. (Optional) Type a description for the KMS key.

   You can add a description now or update it any time unless the key state is `Pending Deletion` or `Pending Replica Deletion`. To add, change, or delete the description of an existing customer managed key, edit the description in the AWS Management Console or use the UpdateKeyDescription operation.

10. (Optional) Type a tag key and an optional tag value. To add more than one tag to the KMS key, choose **Add tag**.

    > **Note**
    >
    > Tagging or untagging a KMS key can allow or deny permission to the KMS key. For details, see ABAC for AWS KMS and Using tags to control access to KMS keys.

    When you add tags to your AWS resources, AWS generates a cost allocation report with usage and costs aggregated by tags. Tags can also be used to control access to a KMS key. For information about tagging KMS keys, see Tagging keys and ABAC for AWS KMS.

11. Choose **Next**.

12. Select the IAM users and roles that can administer the KMS key.

> **Note**
>
> This key policy gives the AWS account full control of this KMS key. It allows account administrators to use IAM policies to give other principals permission to manage the KMS key. For details, see the section called "Default key policy".
>
> IAM best practices discourage the use of IAM users with long-term credentials. Whenever possible, use IAM roles, which provide temporary credentials. For details, see Security best practices in IAM in the *IAM User Guide*.

13. (Optional) To prevent the selected IAM users and roles from deleting this KMS key, in the **Key deletion** section at the bottom of the page, clear the **Allow key administrators to delete this key** check box.
14. Choose **Next**.
15. Select the IAM users and roles that can use the key in cryptographic operations

> **Note**
>
> This key policy gives the AWS account full control of this KMS key. It allows account administrators to use IAM policies to give other principals permission to use the KMS key in cryptographic operations. For details, see the section called "Default key policy".
>
> IAM best practices discourage the use of IAM users with long-term credentials. Whenever possible, use IAM roles, which provide temporary credentials. For details, see Security best practices in IAM in the *IAM User Guide*.

16. (Optional) You can allow other AWS accounts to use this KMS key for cryptographic operations. To do so, in the **Other AWS accounts** section at the bottom of the page, choose **Add another AWS account** and enter the AWS account identification number of an external account. To add multiple external accounts, repeat this step.

> **Note**
>
> To allow principals in the external accounts to use the KMS key, Administrators of the external account must create IAM policies that provide these permissions. For more information, see Allowing users in other accounts to use a KMS key.

17. Choose **Next**.
18. Review the key settings that you chose. You can still go back and change all settings.
19. Choose **Finish** to create the KMS key.

Creating symmetric encryption KMS keys (AWS KMS API)

You can use the CreateKey operation to create AWS KMS keys of all types. These examples use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

> **Important**
>
> Do not include confidential or sensitive information in the `Description` or `Tags` fields. These fields may appear in plain text in CloudTrail logs and other output.

The following operation creates the most commonly used KMS key, a symmetric encryption key in a single Region backed by key material generated by AWS KMS. This operation has no required parameters. However, you might also want to use the `Policy` parameter to specify a key policy. You can change the key policy (PutKeyPolicy) and add optional elements, such as a description and tags at any time. You can also create asymmetric keys, multi-Region keys, keys with imported key material, and keys in custom key stores.

The `CreateKey` operation doesn't let you specify an alias, but you can use the CreateAlias operation to create an alias for your new KMS key.

The following is an example of a call to the `CreateKey` operation with no parameters. This command uses all of the default values. It creates a symmetric encryption KMS key with key material generated by AWS KMS.

```
$ aws kms create-key
{
    "KeyMetadata": {
        "Origin": "AWS_KMS",
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "Description": "",
        "KeyManager": "CUSTOMER",
        "Enabled": true,
        "KeySpec": "SYMMETRIC_DEFAULT",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "CreationDate": 1502910355.475,
        "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "AWSAccountId": "111122223333",
        "MultiRegion": false
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ],
    }
}
```

If you do not specify a key policy for your new KMS key, the default key policy that `CreateKey` applies differs from the default key policy that the console applies when you use it to create a new KMS key.

For example, this call to the GetKeyPolicy operation returns the key policy that `CreateKey` applies. It gives the AWS account access to the KMS key and allows it to create AWS Identity and Access Management (IAM) policies for the KMS key. For detailed information about IAM policies and key policies for KMS keys, see Authentication and access control for AWS KMS

```
$ aws kms get-key-policy --key-id 1234abcd-12ab-34cd-56ef-1234567890ab --policy-name
 default --output text
{
  "Version" : "2012-10-17",
  "Id" : "key-default-1",
  "Statement" : [ {
    "Sid" : "Enable IAM User Permissions",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::111122223333:root"
    },
    "Action" : "kms:*",
    "Resource" : "*"
  } ]
}
```

Using aliases

An *alias* is a friendly name for a AWS KMS key. For example, an alias lets you refer to a KMS key as `test-key` instead of `1234abcd-12ab-34cd-56ef-1234567890ab`.

You can use an alias to identify a KMS key in the AWS KMS console, in the DescribeKey operation, and in cryptographic operations, such as Encrypt and GenerateDataKey. Aliases also make it easy to recognize an AWS managed key. Aliases for these KMS keys always have the form `aws/<service-name>`. For example, the alias for the AWS managed key for Amazon DynamoDB is `aws/dynamodb`. You can establish similar alias standards for your projects, such as prefacing your aliases with the name of a project or category.

You can also allow and deny access to KMS keys based on their aliases without editing policies or managing grants. This feature is part of AWS KMS support for attribute-based access control (ABAC). For details, see Using aliases to control access to KMS keys.

Much of the power of aliases come from your ability to change the KMS key associated with an alias at any time. Aliases can make your code easier to write and maintain. For example, suppose you use an alias to refer to a particular KMS key and you want to change the KMS key. In that case, just associate the alias with a different KMS key. You don't need to change your code.

Aliases also make it easier to reuse the same code in different AWS Regions. Create aliases with the same name in multiple Regions and associate each alias with a KMS key in its Region. When the code runs in each Region, the alias refers to the associated KMS key in that Region. For an example, see Using aliases in your applications.

You can create an alias for a KMS key in the AWS KMS console, by using the CreateAlias API, or by using an AWS CloudFormation template.

The AWS KMS API provides full control of aliases in each account and Region. The API includes operations to create an alias (CreateAlias), view alias names and alias ARNs (ListAliases), change the KMS key associated with an alias (UpdateAlias), and delete an alias (DeleteAlias). For examples of managing aliases multiple programming languages, see the section called “Working with aliases”.

The following resources can help you learn more:

- For information about KMS key identifiers, including aliases, see Key identifiers (KeyId).
- For help using a AWS CloudFormation template to create an alias for a KMS key, see AWS::KMS::Alias in the *AWS CloudFormation User Guide*.

- For help finding the aliases associated with a KMS key, see Finding the alias name and alias ARN
- For information about resource quotas for aliases and rate quotas for API operations related to aliases, see Quotas.
- For examples of creating and managing aliases in multiple programming languages, see Working with aliases.

Topics

About aliases

Learn how aliases work in AWS KMS.

An alias is an independent AWS resource

An alias is not a property of a KMS key. The actions that you take on the alias don't affect its associated KMS key. You can create an alias for a KMS key and then update the alias so it's associated with a different KMS key. You can even delete the alias without any effect on the associated KMS key. However, if you delete a KMS key, all aliases associated with that KMS key are deleted.

If you specify an alias as the resource in an IAM policy, the policy refers to the alias, not to the associated KMS key.

Each alias has two formats

When you create an alias, you specify the alias name. AWS KMS creates the alias ARN for you.

- An alias ARN is an Amazon Resource Name (ARN) that uniquely identifies the alias.

```
# Alias ARN
arn:aws:kms:us-west-2:111122223333:alias/<alias-name>
```

- An alias name that is unique in the account and Region. In the AWS KMS API, the alias name is always prefixed by `alias/`. That prefix is omitted in the AWS KMS console.

  ```
  # Alias name
  alias/<alias-name>
  ```

Aliases are not secret

Aliases may be displayed in plaintext in CloudTrail logs and other output. Do not include confidential or sensitive information in the alias name.

Each alias is associated with one KMS key at a time

The alias and its KMS key must be in the same account and Region.

You can associate an alias with any customer managed key in the same AWS account and Region. However, you do not have permission to associate an alias with an AWS managed key.

For example, this ListAliases output shows that the `test-key` alias is associated with exactly one target KMS key, which is represented by the `TargetKeyId` property.

```
{
     "AliasName": "alias/test-key",
     "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/test-key",
     "TargetKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
     "CreationDate": 1593622000.191,
     "LastUpdatedDate": 1593622000.191
}
```

Multiple aliases can be associated with the same KMS key

For example, you can associate the `test-key` and `project-key` aliases with the same KMS key.

```
{
     "AliasName": "alias/test-key",
     "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/test-key",
     "TargetKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
     "CreationDate": 1593622000.191,
     "LastUpdatedDate": 1593622000.191
},
{
```

```
        "AliasName": "alias/project-key",
        "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/project-key",
        "TargetKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "CreationDate": 1516435200.399,
        "LastUpdatedDate": 1516435200.399
}
```

An alias must be unique in an account and Region

For example, you can have only one `test-key` alias in each account and Region. Aliases are case-sensitive, but aliases that differ only in their capitalization are very prone to error. You cannot change an alias name. However, you can delete the alias and create a new alias with the desired name.

You can create an alias with the same name in different Regions

For example, you can have a `finance-key` alias in US East (N. Virginia) and a `finance-key` alias in Europe (Frankfurt). Each alias would be associated with a KMS key in its Region. If your code refers to an alias name like `alias/finance-key`, you can run it in multiple Regions. In each Region, it uses a different KMS key. For details, see Using aliases in your applications.

You can change the KMS key associated with an alias

You can use the UpdateAlias operation to associate an alias with a different KMS key. For example, if the `finance-key` alias is associated with the `1234abcd-12ab-34cd-56ef-1234567890ab` KMS key, you can update it so it is associated with the `0987dcba-09fe-87dc-65ba-ab0987654321` KMS key.

However, the current and new KMS key must be the same type (both symmetric or both asymmetric or both HMAC), and they must have the same key usage (ENCRYPT_DECRYPT or SIGN_VERIFY or GENERATE_VERIFY_MAC). This restriction prevents errors in code that uses aliases. If you must associate an alias with a different type of key, and you have mitigated the risks, you can delete and recreate the alias.

Some KMS keys don't have aliases

When you create a KMS key in the AWS KMS console, you must give it a new alias. But an alias is not required when you use the CreateKey operation to create a KMS key. Also, you can use the UpdateAlias operation to change the KMS key associated with an alias and the DeleteAlias operation to delete an alias. As a result, some KMS keys might have several aliases, and some might have none.

AWS creates aliases in your account

AWS creates aliases in your account for AWS managed keys. These aliases have names of the form `alias/aws/<service-name>`, such as `alias/aws/s3`.

Some AWS aliases have no KMS key. These predefined aliases are usually associated with an AWS managed key when you start using the service.

Use aliases to identify KMS keys

You can use an alias name or alias ARN to identify a KMS key in cryptographic operations, DescribeKey, and GetPublicKey. (If the KMS key is in a different AWS account, you must use its key ARN or alias ARN.) Aliases are not valid identifiers for KMS keys in other AWS KMS operations. For information about the valid key identifiers for each AWS KMS API operation, see the descriptions of the `KeyId` parameters in the *AWS Key Management Service API Reference*.

You cannot use an alias name or alias ARN to identify a KMS key in an IAM policy. To control access to a KMS key based on its aliases, use the kms:RequestAlias or kms:ResourceAliases condition keys. For details, see ABAC for AWS KMS.

Managing aliases

Authorized users can create, view, and delete aliases. You can also update an alias, that is, associate an existing alias with a different KMS key.

Topics

Creating an alias

You can create aliases in the AWS KMS console or by using AWS KMS API operations.

The alias must be string of 1–256 characters. It can contain only alphanumeric characters, forward slashes (/), underscores (_), and dashes (-). The alias name for a customer managed key cannot begin with `alias/aws/`. The `alias/aws/` prefix is reserved for AWS managed key.

You can create an alias for a new KMS key or for an existing KMS key. You might add an alias so that a particular KMS key is used in a project or application.

Create an alias (console)

When you create a KMS key in the AWS KMS console, you must create an alias for the new KMS key. To create an alias for an existing KMS key, use the **Aliases** tab on the detail page for the KMS key.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**. You cannot manage aliases for AWS managed keys or AWS owned keys.
4. In the table, choose the key ID or alias of the KMS key. Then, on the KMS key detail page, choose the **Aliases** tab.

   If a KMS key has multiple aliases, the **Aliases** column in the table displays one alias and an alias summary, such as **(+*n* more)**. Choosing the alias summary takes you directly to the **Aliases** tab on the KMS key detail page.
5. On the **Aliases** tab, choose **Create alias**. Enter an alias name and choose **Create alias**.

   > ⚠ **Important**
   >
   > Do not include confidential or sensitive information in this field. This field may be displayed in plaintext in CloudTrail logs and other output.

   > ⓘ **Note**
   >
   > Do not add the `alias/` prefix. The console automatically adds it for you. If you enter `alias/ExampleAlias`, the actual alias name will be `alias/alias/ExampleAlias`.

Create an alias (AWS KMS API)

To create an alias, use the CreateAlias operation. Unlike the process of creating KMS keys in the console, the CreateKey operation doesn't create an alias for a new KMS key.

> **Important**
>
> Do not include confidential or sensitive information in this field. This field may be displayed in plaintext in CloudTrail logs and other output.

You can use the `CreateAlias` operation to create an alias for a new KMS key with no alias. You can also use the `CreateAlias` operation to add an alias to any existing KMS key or to recreate an alias that was accidentally deleted.

In the AWS KMS API operations, the alias name must begin with `alias/` followed by a name, such as `alias/ExampleAlias`. The alias must be unique in the account and Region. To find the alias names that are already in use, use the ListAliases operation. The alias name is case sensitive.

The `TargetKeyId` can be any customer managed key in the same AWS Region. To identify the KMS key, use its key ID or key ARN. You cannot use another alias.

The following example creates the `example-key` alias and associates it with the specified KMS key. These examples use the AWS Command Line Interface (AWS CLI). For examples in multiple programming languages, see Working with aliases.

```
$ aws kms create-alias \
    --alias-name alias/example-key \
    --target-key-id 1234abcd-12ab-34cd-56ef-1234567890ab
```

`CreateAlias` does not return any output. To see the new alias, use the `ListAliases` operation. For details, see Viewing aliases (AWS KMS API).

Viewing aliases

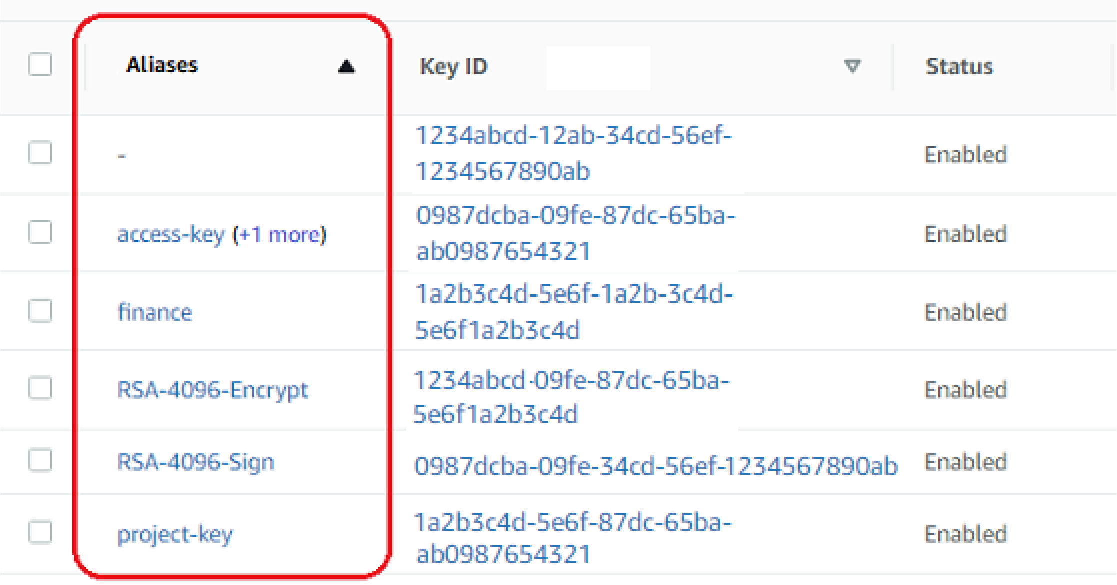
Aliases make it easy to recognize KMS keys in the AWS KMS console. You can view the aliases for a KMS key in the AWS KMS console or by using the ListAliases operation. The DescribeKey operation, which returns the properties of a KMS key, does not include aliases.

Viewing aliases (console)

The **Customer managed keys** and **AWS managed keys** pages in the AWS KMS console display the alias associated with each KMS key. You can also search, sort, and filter KMS keys based on their aliases.

The following image of the AWS KMS console shows the aliases on the **Customer managed keys** page of an example account. As shown in the image, some KMS keys do not have an alias.

When a KMS key has multiple aliases, the **Aliases** column displays one alias and an *alias summary* **(+*n* more)**. The alias summary shows how many additional aliases are associated with the KMS key and links to the display of all aliases for the KMS key on the **Aliases** tab.



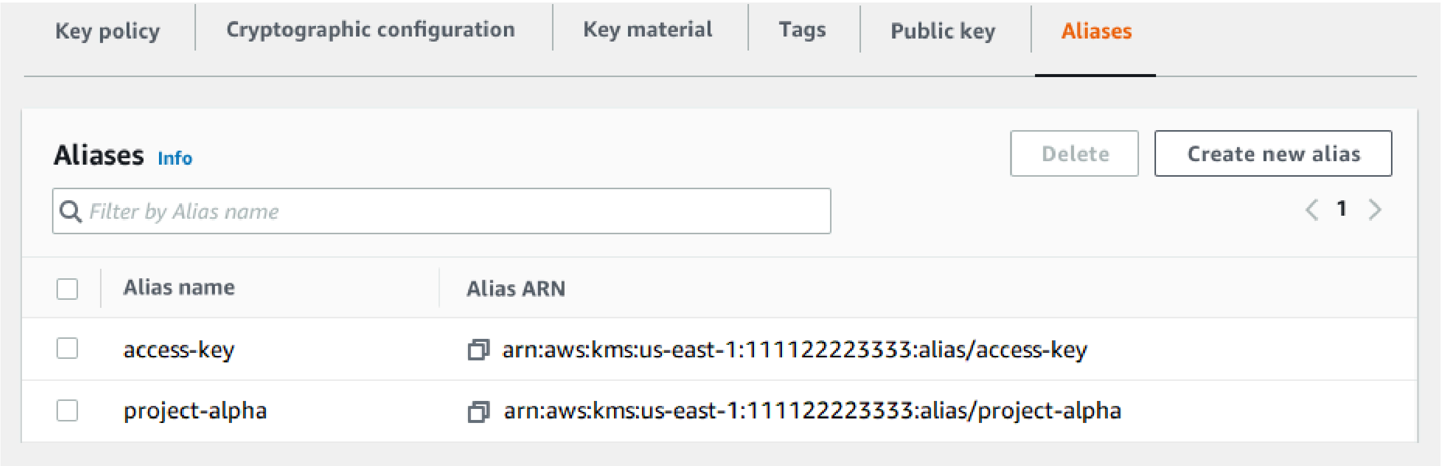
□	Aliases ▲	Key ID ▽	Status
□	-	1234abcd-12ab-34cd-56ef-1234567890ab	Enabled
□	access-key (+1 more)	0987dcba-09fe-87dc-65ba-ab0987654321	Enabled
□	finance	1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d	Enabled
□	RSA-4096-Encrypt	1234abcd-09fe-87dc-65ba-5e6f1a2b3c4d	Enabled
□	RSA-4096-Sign	0987dcba-09fe-34cd-56ef-1234567890ab	Enabled
□	project-key	1a2b3c4d-5e6f-87dc-65ba-ab0987654321	Enabled

The **Aliases** tab on the details page for each KMS key displays the alias name and alias ARN of all aliases for the KMS key in the AWS account and Region. You can also use the **Aliases** tab to [create aliases](#) and [delete aliases](#).

To find the alias name and alias ARN of all aliases for the KMS key, use the **Aliases** tab.

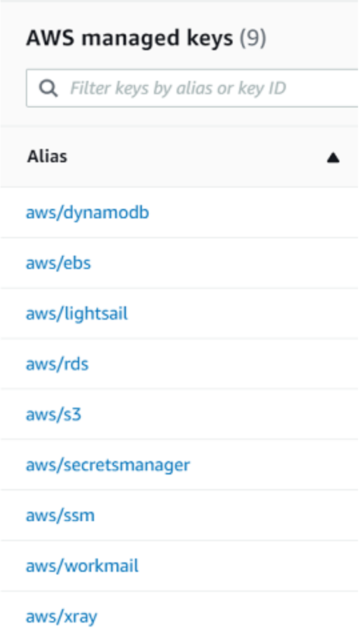
- To go directly to the **Aliases** tab, in the **Aliases** column, choose the alias summary (**+*n* more**). An alias summary appears only if the KMS key has more than one alias.
- Or, choose the alias or key ID of the KMS key (which opens the detail page for the KMS key) and then choose the **Aliases** tab. The tabs are under the **General configuration** section.

The following image shows the **Aliases** tab for an example KMS key.



□	Alias name	Alias ARN
□	access-key	arn:aws:kms:us-east-1:111122223333:alias/access-key
□	project-alpha	arn:aws:kms:us-east-1:111122223333:alias/project-alpha

You can use the alias to recognize an AWS managed key, as shown in this example **AWS managed keys** page. The aliases for AWS managed keys always have the format: `aws/<service-name>`. For example, the alias for the AWS managed key for Amazon DynamoDB is `aws/dynamodb`.



Alias ▲
aws/dynamodb
aws/ebs
aws/lightsail
aws/rds
aws/s3
aws/secretsmanager
aws/ssm
aws/workmail
aws/xray

Viewing aliases (AWS KMS API)

The ListAliases operation returns the alias name and alias ARN of aliases in the account and Region. The output includes aliases for AWS managed keys and for customer managed keys. The aliases for AWS managed keys have the format `aws/<service-name>`, such as `aws/dynamodb`.

The response might also include aliases that have no `TargetKeyId` field. These are predefined aliases that AWS has created but has not yet associated with a KMS key.

```
$ aws kms list-aliases
{
    "Aliases": [
        {
            "AliasName": "alias/access-key",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/access-key",
            "TargetKeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
            "CreationDate": 1516435200.399,
            "LastUpdatedDate": 1516435200.399
        },
        {
            "AliasName": "alias/ECC-P521-Sign",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/ECC-P521-Sign",
            "TargetKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
            "CreationDate": 1693622000.704,
            "LastUpdatedDate": 1693622000.704
        },
        {
            "AliasName": "alias/ImportedKey",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/ImportedKey",
            "TargetKeyId": "1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d",
            "CreationDate": 1493622000.704,
            "LastUpdatedDate": 1521097200.235
        },
        {
            "AliasName": "alias/finance-project",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/finance-project",
            "TargetKeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
            "CreationDate": 1604958290.014,
            "LastUpdatedDate": 1604958290.014
        },
        {
            "AliasName": "alias/aws/dynamodb",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/aws/dynamodb",
            "TargetKeyId": "0987ab65-43cd-21ef-09ab-87654321cdef",
            "CreationDate": 1521097200.454,
            "LastUpdatedDate": 1521097200.454
        },
        {
            "AliasName": "alias/aws/ebs",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/aws/ebs",
            "TargetKeyId": "abcd1234-09fe-ef90-09fe-ab0987654321",
            "CreationDate": 1466518990.200,
            "LastUpdatedDate": 1466518990.200
        }
    ]
}
```

To get all aliases that are associated with a particular KMS key, use the optional `KeyId` parameter of the `ListAliases` operation. The `KeyId` parameter takes the key ID or key ARN of the KMS key.

This example gets all aliases associated with the `0987dcba-09fe-87dc-65ba-ab0987654321` KMS key.

```
$ aws kms list-aliases --key-id 0987dcba-09fe-87dc-65ba-ab0987654321
{
    "Aliases": [
        {
            "AliasName": "alias/access-key",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/access-key",
            "TargetKeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
            "CreationDate": "2018-01-20T15:23:10.194000-07:00",
            "LastUpdatedDate": "2018-01-20T15:23:10.194000-07:00"
        },
        {
            "AliasName": "alias/finance-project",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/finance-project",
            "TargetKeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
            "CreationDate": 1604958290.014,
            "LastUpdatedDate": 1604958290.014
        }
    ]
}
```

The `KeyId` parameter doesn't take wildcard characters, but you can use the features of your programming language to filter the response.

For example, the following AWS CLI command gets only the aliases for AWS managed keys.

```
$ aws kms list-aliases --query 'Aliases[?starts_with(AliasName, `alias/aws/`)]'
```

The following command gets only the `access-key` alias. The alias name is case-sensitive.

```
$ aws kms list-aliases --query 'Aliases[?AliasName==`alias/access-key`]'
[
    {
        "AliasName": "alias/access-key",
        "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/access-key",
        "TargetKeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
        "CreationDate": "2018-01-20T15:23:10.194000-07:00",
        "LastUpdatedDate": "2018-01-20T15:23:10.194000-07:00"
    }
]
```

Updating aliases

Because an alias is an independent resource, you can change the KMS key associated with an alias. For example, if the `test-key` alias is associated with one KMS key, you can use the UpdateAlias operation to associate it with a different KMS key. This is one of several ways to manually rotate a KMS key without changing its key material. You might also update a KMS key so that an application that was using one KMS key for new resources is now using a different KMS key.

You cannot update an alias in the AWS KMS console. Also, you cannot use `UpdateAlias` (or any other operation) to change an alias name. To change an alias name, delete the current alias and then create a new alias for the KMS key.

When you update an alias, the current KMS key and the new KMS key must be the same type (both symmetric or asymmetric or HMAC). They must also have the same key usage (ENCRYPT_DECRYPT or `SIGN_VERIFY` or GENERATE_VERIFY_MAC). This restriction prevents cryptographic errors in code that uses aliases.

The following example begins by using the ListAliases operation to show that the `test-key` alias is currently associated with KMS key `1234abcd-12ab-34cd-56ef-1234567890ab`.

```
$ aws kms list-aliases --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
{
    "Aliases": [
        {
            "AliasName": "alias/test-key",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/test-key",
            "TargetKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
            "CreationDate": 1593622000.191,
            "LastUpdatedDate": 1593622000.191
        }
    ]
}
```

Next, it uses the `UpdateAlias` operation to change the KMS key that is associated with the `test-key` alias to KMS key `0987dcba-09fe-87dc-65ba-ab0987654321`. You don't need to specify the currently associated KMS key, only the new ("target") KMS key. The alias name is case sensitive.

```
$ aws kms update-alias --alias-name 'alias/test-key' --target-key-id
 0987dcba-09fe-87dc-65ba-ab0987654321
```

To verify that the alias is now associated with the target KMS key, use the `ListAliases` operation again. This AWS CLI command uses the `--query` parameter to get only the `test-key` alias. The `TargetKeyId` and `LastUpdatedDate` fields are updated.

```
$ aws kms list-aliases --query 'Aliases[?AliasName==`alias/test-key`]'
[
    {
        "AliasName": "alias/test-key",
        "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/test-key",
        "TargetKeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
        "CreationDate": 1593622000.191,
        "LastUpdatedDate": 1604958290.154
    }
]
```

Deleting an alias

You can delete an alias in the AWS KMS console or by using the DeleteAlias operation. Before deleting an alias, make sure that it's not in use. Although deleting an alias doesn't affect the associated KMS key, it might create problems for any application that uses the alias. If you delete an alias by mistake, you can create a new alias with the same name and associate it with the same or a different KMS key.

If you delete a KMS key, all aliases associated with that KMS key are deleted.

Delete aliases (console)

To delete an alias in the AWS KMS console, use the **Aliases** tab on the detail page for the KMS key. You can delete multiple aliases for a KMS key at one time.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.

3. In the navigation pane, choose **Customer managed keys**. You cannot manage aliases for AWS managed keys or AWS owned keys.
4. In the table, choose the key ID or alias of the KMS key. Then, on the KMS key detail page, choose the **Aliases** tab.

   If a KMS key has multiple aliases, the **Aliases** column in the table displays one alias and an alias summary, such as **(+*n* more)**. Choosing the alias summary takes you directly to the **Aliases** tab on the KMS key detail page.
5. On the **Aliases** tab, select the check box next to the aliases that you want to delete. Then choose **Delete**.

Delete an alias (AWS KMS API)

To delete an alias, use the DeleteAlias operation. This operation deletes one alias at a time. The alias name is case-sensitive and it must be preceded by the `alias/` prefix.

For example, the following command deletes the `test-key` alias. This command does not return any output.

```
$ aws kms delete-alias --alias-name alias/test-key
```

To verify that the alias is deleted, use the ListAliases operation. The following command uses the `--query` parameter in the AWS CLI to get only the `test-key` alias. The empty brackets in the response indicate that the `ListAliases` response didn't include a `test-key` alias. To eliminate the brackets, use the `--output text` parameter and value.

```
$ aws kms list-aliases --query 'Aliases[?AliasName==`alias/test-key`]'
[]
```

Using aliases in your applications

You can use an alias to represent a KMS key in your application code. The `KeyId` parameter in AWS KMS cryptographic operations, DescribeKey, and GetPublicKey accepts an alias name or alias ARN.

For example, the following `GenerateDataKey` command uses an alias name (`alias/finance`) to identify a KMS key. The alias name is the value of the `KeyId` parameter.

```
$ aws kms generate-data-key --key-id alias/finance --key-spec AES_256
```

If the KMS key is in a different AWS account, you must use a key ARN or alias ARN in these operations. When using an alias ARN, remember that the alias for a KMS key is defined in the account that owns the KMS key and might differ in each Region. For help finding the alias ARN, see Finding the alias name and alias ARN.

For example, the following `GenerateDataKey` command uses a KMS key that's not in the caller's account. The `ExampleAlias` alias is associated with the KMS key in the specified account and Region.

```
$ aws kms generate-data-key --key-id arn:aws:kms:us-west-2:444455556666:alias/
ExampleAlias --key-spec AES_256
```

One of the most powerful uses of aliases is in applications that run in multiple AWS Regions. For example, you might have a global application that uses an RSA asymmetric KMS key for signing and verification.

- In US West (Oregon) (us-west-2), you want to use `arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab`.
- In Europe (Frankfurt) (eu-central-1), you want to use `arn:aws:kms:eu-central-1:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321`
- In Asia Pacific (Singapore) (ap-southeast-1), you want to use `arn:aws:kms:ap-southeast-1:111122223333:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d`.

You could create a different version of your application in each Region or use a dictionary or switch statement to select the right KMS key for each Region. But it's much easier to create an alias with the same alias name in each Region. Remember that the alias name is case-sensitive.

```
aws --region us-west-2 kms create-alias \
    --alias-name alias/new-app \
    --key-id arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

aws --region eu-central-1 kms create-alias \
    --alias-name alias/new-app \
    --key-id arn:aws:kms:eu-central-1:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321

aws --region ap-southeast-1 kms create-alias \
    --alias-name alias/new-app \
    --key-id arn:aws:kms:ap-
southeast-1:111122223333:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d
```

Then, use the alias in your code. When your code runs in each Region, the alias will refer to its associated KMS key in that Region. For example, this code calls the Sign operation with an alias name.

```
aws kms sign --key-id alias/new-app \
    --message $message \
    --message-type RAW \
    --signing-algorithm RSASSA_PSS_SHA_384
```

However, there is a risk that the alias might be deleted or updated to be associated with a different KMS key. In that case, the application's attempts to verify signatures using the alias name will fail, and you might need to recreate or update the alias.

To mitigate this risk, be cautious about giving principals permission to manage the aliases that you use in your application. For details, see Controlling access to aliases.

There are several other solutions for applications that encrypt data in multiple AWS Regions, including the AWS Encryption SDK.

Controlling access to aliases

When you create or change an alias, you affect the alias and its associated KMS key. Therefore, principals who manage aliases must have permission to call the alias operation on the alias and on all affected KMS keys. You can provide these permissions by using key policies, IAM policies and grants.

> **Note**
>
> Be cautious when giving principals permission to manage tags and aliases. Changing a tag or alias can allow or deny permission to the customer managed key. For details, see ABAC for AWS KMS and Using aliases to control access to KMS keys.

For information about controlling access to all AWS KMS operations, see Permissions reference.

Permissions to create and manage aliases work as follows.

kms:CreateAlias

To create an alias, the principal needs the following permissions for both the alias and for the associated KMS key.

- `kms:CreateAlias` for the alias. Provide this permission in an IAM policy that is attached to the principal who is allowed to create the alias.

  The following example policy statement specifies a particular alias in a `Resource` element. But you can list multiple alias ARNs or specify an alias pattern, such as "test*". You can also specify a `Resource` value of `"*"` to allow the principal to create any alias in the account and Region. Permission to create an alias can also be included in a `kms:Create*` permission for all resources in an account and Region.

  ```
  {
    "Sid": "IAMPolicyForAnAlias",
    "Effect": "Allow",
    "Action": [
      "kms:CreateAlias",
      "kms:UpdateAlias",
      "kms:DeleteAlias"
    ],
    "Resource": "arn:aws:kms:us-west-2:111122223333:alias/test-key"
  }
  ```

- `kms:CreateAlias` for the KMS key. This permission must be provided in a key policy or in an IAM policy that is delegated from the key policy.

  ```
  {
    "Sid": "Key policy for 1234abcd-12ab-34cd-56ef-1234567890ab",
    "Effect": "Allow",
    "Principal": {"AWS": "arn:aws:iam::111122223333:user/KMSAdminUser"},
    "Action": [
      "kms:CreateAlias",
      "kms:DescribeKey"
    ],
    "Resource": "*"
  }
  ```

You can use condition keys to limit the KMS keys that you can associate with an alias. For example, you can use the kms:KeySpec condition key to allow the principal to create aliases only on asymmetric KMS keys. For a full list of conditions keys that you can use to limit the `kms:CreateAlias` permission on KMS key resources, see AWS KMS permissions.

kms:ListAliases

To list aliases in the account and Region, the principal must have `kms:ListAliases` permission in an IAM policy. Because this policy isn't related to any particular KMS key or alias resource, the value of the resource element in the policy must be `"*"`.

For example, the following IAM policy statement gives the principal permission to list all KMS keys and aliases in the account and Region.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:ListKeys",
      "kms:ListAliases"
    ],
    "Resource": "*"
  }
}
```

kms:UpdateAlias

To change the KMS key that is associated with an alias, the principal needs three permission elements: one for the alias, one for the current KMS key, and one for the new KMS key.

For example, suppose you want to change the `test-key` alias from the KMS key with key ID 1234abcd-12ab-34cd-56ef-1234567890ab to the KMS key with key ID 0987dcba-09fe-87dc-65ba-ab0987654321. In that case, include policy statements similar to the examples in this section.

- `kms:UpdateAlias` for the alias. You provide this permission in an IAM policy that is attached to the principal. The following IAM policy specifies a particular alias. But you can list multiple alias ARNs or specify an alias pattern, such as `"test*"`. You can also specify a `Resource` value of `"*"` to allow the principal to update any alias in the account and Region.

  ```
  {
  ```

```
  "Sid": "IAMPolicyForAnAlias",
  "Effect": "Allow",
  "Action": [
    "kms:UpdateAlias",
    "kms:ListAliases",
    "kms:ListKeys"
  ],
  "Resource": "arn:aws:kms:us-west-2:111122223333:alias/test-key"
}
```

- `kms:UpdateAlias` for the KMS key that is currently associated with the alias. This permission must be provided in a key policy or in an IAM policy that is delegated from the key policy.

```
{
  "Sid": "Key policy for 1234abcd-12ab-34cd-56ef-1234567890ab",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::111122223333:user/KMSAdminUser"},
  "Action": [
    "kms:UpdateAlias",
    "kms:DescribeKey"
  ],
  "Resource": "*"
}
```

- `kms:UpdateAlias` for the KMS key that the operation associates with the alias. This permission must be provided in a key policy or in an IAM policy that is delegated from the key policy.

```
{
  "Sid": "Key policy for 0987dcba-09fe-87dc-65ba-ab0987654321",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::111122223333:user/KMSAdminUser"},
  "Action": [
    "kms:UpdateAlias",
    "kms:DescribeKey"
  ],
  "Resource": "*"
}
```

You can use condition keys to limit either or both of KMS keys in an `UpdateAlias` operation. For example, you can use a kms:ResourceAliases condition key to allow the principal to update aliases only when the target KMS key already has a particular alias. For a full list of conditions keys that you can use to limit the `kms:UpdateAlias` permission on a KMS key resource, see [AWS KMS permissions](#).

kms:DeleteAlias

To delete an alias, the principal needs permission for the alias and for the associated KMS key.

As always, you should exercise caution when giving principals permission to delete a resource. However, deleting an alias has no effect on the associated KMS key. Although it might cause a failure in an application that relies on the alias, if you mistakenly delete an alias, you can recreate it.

- `kms:DeleteAlias` for the alias. Provide this permission in an IAM policy attached to the principal who is allowed to delete the alias.

  The following example policy statement specifies the alias in a `Resource` element. But you can list multiple alias ARNs or specify an alias pattern, such as `"test*"`, You can also specify a `Resource` value of `"*"` to allow the principal to delete any alias in the account and Region.

  ```
  {
    "Sid": "IAMPolicyForAnAlias",
    "Effect": "Allow",
    "Action": [
      "kms:CreateAlias",
      "kms:UpdateAlias",
      "kms:DeleteAlias"
    ],
    "Resource": "arn:aws:kms:us-west-2:111122223333:alias/test-key"
  }
  ```

- `kms:DeleteAlias` for the associated KMS key. This permission must be provided in a key policy or in an IAM policy that is delegated from the key policy.

  ```
  {
    "Sid": "Key policy for 1234abcd-12ab-34cd-56ef-1234567890ab",
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::111122223333:user/KMSAdminUser"
    },
    "Action": [
      "kms:CreateAlias",
      "kms:UpdateAlias",
  ```

```
      "kms:DeleteAlias",
      "kms:DescribeKey"
    ],
    "Resource": "*"
  }
```

Limiting alias permissions

You can use condition keys to limit alias permissions when the resource is a KMS key. For example, the following IAM policy allows the alias operations on KMS keys in a particular account and Region. However, it uses the kms:KeyOrigin condition key to further limit the permissions to KMS keys with key material from AWS KMS.

For a full list of conditions keys that you can use to limit alias permission on a KMS key resource, see AWS KMS permissions.

```
{
  "Sid": "IAMPolicyKeyPermissions",
  "Effect": "Allow",
  "Resource": "arn:aws:kms:us-west-2:111122223333:key/*",
  "Action": [
    "kms:CreateAlias",
    "kms:UpdateAlias",
    "kms:DeleteAlias"
  ],
  "Condition": {
    "StringEquals": {
      "kms:KeyOrigin": "AWS_KMS"
    }
  }
}
```

You can't use condition keys in a policy statement where the resource is an alias. To limit the aliases that a principal can manage, use the value of the `Resource` element of the IAM policy statement that controls access to the alias. For example, the following policy statements allow the principal to create, update, or delete any alias in the AWS account and Region unless the alias begins with `Restricted`.

```
{
  "Sid": "IAMPolicyForAnAliasAllow",
  "Effect": "Allow",
  "Action": [
    "kms:CreateAlias",
    "kms:UpdateAlias",
    "kms:DeleteAlias"
  ],
  "Resource": "arn:aws:kms:us-west-2:111122223333:alias/*"
},
{
  "Sid": "IAMPolicyForAnAliasDeny",
  "Effect": "Deny",
  "Action": [
    "kms:CreateAlias",
    "kms:UpdateAlias",
    "kms:DeleteAlias"
  ],
  "Resource": "arn:aws:kms:us-west-2:111122223333:alias/Restricted*"
}
```

Using aliases to control access to KMS keys

You can control access to KMS keys based on the aliases that are associated with the KMS key. To do so, use the kms:RequestAlias and kms:ResourceAliases condition keys. This feature is part of AWS KMS support for attribute-based access control (ABAC).

The `kms:RequestAlias` condition key allows or denies access to a KMS key based on the alias in a request. The `kms:ResourceAliases` condition key allows or denies access to a KMS key based on the aliases associated with the KMS key.

These features do not allow you to identify a KMS key by using an alias in the `resource` element of a policy statement. When an alias is the value of a `resource` element, the policy applies to the alias resource, not to any KMS key that might be associated with it.

Note

It might take up to five minutes for tag and alias changes to affect KMS key authorization. Recent changes might be visible in API operations before they affect authorization.

When using aliases to control access to KMS keys, consider the following:

- Use aliases to reinforce the best practice of least privileged access. Give IAM principals only the permissions that they need for only the KMS keys that they must use or manage. For example, use aliases to identify the KMS keys used for a project. Then give the project team permission to use only KMS keys with the project aliases.
- Be cautious about giving principals the `kms:CreateAlias`, `kms:UpdateAlias`, or `kms:DeleteAlias` permissions that let them add, edit, and delete aliases. When you use aliases to control access to KMS keys, changing an alias can give principals permission to use KMS keys that they didn't otherwise have permission to use. It can also deny access to KMS keys that other principals require to do their jobs.
- Review the principals in your AWS account that currently have permission to manage aliases and adjust the permissions, if necessary. Key administrators who don't have permission to change key policies or create grants can control access to KMS keys if they have permission to manage aliases.

  For example, the console default key policy for key administrators includes `kms:CreateAlias`, `kms:DeleteAlias`, and `kms:UpdateAlias` permission. IAM policies might give alias permissions for all KMS keys in your AWS account. For example, the AWSKeyManagementServicePowerUser managed policy allows principals to create, delete, and list aliases for all KMS keys but not update them.
- Before setting a policy that depends on an alias, review the aliases on the KMS keys in your AWS account. Make sure that your policy applies only to the aliases that you intend to include. Use CloudTrail logs and CloudWatch alarms to alert you to alias changes that might affect access to your KMS keys. Also, the ListAliases response includes the creation date and last updated date for each alias.
- The alias policy conditions use pattern matching; they aren't tied to a particular instance of an alias. A policy that uses alias-based condition keys affects all new and existing aliases that match the pattern. If you delete and recreate an alias that matches a policy condition, the condition applies to the new alias, just as it did to the old one.

The `kms:RequestAlias` condition key relies on the alias specified explicitly in an operation request. The `kms:ResourceAliases` condition key depends on the aliases that are associated with a KMS key, even if they don't appear in the request.

kms:RequestAlias

Allow or deny access to a KMS key based on the alias that identifies the KMS key in a request. You can use the kms:RequestAlias condition key in a key policy or IAM policy. It applies to operations that use an alias to identify a KMS key in a request, namely cryptographic operations, DescribeKey, and GetPublicKey. It is not valid for alias operations, such as CreateAlias or DeleteAlias.

In the condition key, specify an alias name or alias name pattern. You cannot specify an alias ARN.

For example, the following key policy statement allows principals to use the specified operations on the KMS key. The permission is effective only when the request uses an alias that includes `alpha` to identify the KMS key.

```
{
  "Sid": "Key policy using a request alias condition",
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/alpha-developer"
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey*",
    "kms:DescribeKey"
  ],
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "kms:RequestAlias": "alias/*alpha*"
    }
  }
}
```

The following example request from an authorized principal would fulfill the condition. However, a request that used a key ID, a key ARN, or a different alias would not fulfill the condition, even if these values identified the same KMS key.

```
$ aws kms describe-key --key-id "arn:aws:kms:us-west-2:111122223333:alias/project-
alpha"
```

kms:ResourceAliases

Allow or deny access to a KMS key based on the aliases associated with the KMS key, even if the alias isn't used in a request. The kms:ResourceAliases condition key lets you specify an alias or alias pattern, such as `alias/test*`, so you can use it in an IAM policy to control access to several KMS keys in the same Region. It's valid for any AWS KMS operation that uses a KMS key.

For example, the following IAM policy lets the principals manage automatic key rotation on the KMS keys in two AWS accounts. However, the permission applies only to KMS keys associated with aliases that begin with `restricted`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AliasBasedIAMPolicy",
      "Effect": "Allow",
      "Action": [
        "kms:EnableKeyRotation",
        "kms:DisableKeyRotation",
        "kms:GetKeyRotationStatus"
      ],
      "Resource": [
        "arn:aws:kms:*:111122223333:key/*",
        "arn:aws:kms:*:444455556666:key/*"
      ],
      "Condition": {
        "ForAnyValue:StringLike": {
          "kms:ResourceAliases": "alias/restricted*"
        }
      }
    }
  ]
}
```

The `kms:ResourceAliases` condition is a condition of the resource, not the request. As such, a request that doesn't specify the alias can still satisfy the condition.

The following example request, which specifies a matching alias, satisfies the condition.

```
$ aws kms enable-key-rotation --key-id "alias/restricted-project"
```

However, the following example request also satisfies the condition, provided that the specified KMS key has an alias that begins with `restricted`, even if that alias isn't used in the request.

```
$ aws kms enable-key-rotation --key-id "1234abcd-12ab-34cd-56ef-1234567890ab"
```

Finding aliases in AWS CloudTrail logs

You can use an alias to represent an AWS KMS key in an AWS KMS API operation. When you do, the alias and the key ARN of the KMS key are recorded in the AWS CloudTrail log entry for the event. The alias appears in the `requestParameters` field. The key ARN appears in the `resources` field. This is true even when an AWS service uses an AWS managed key in your account.

For example, the following GenerateDataKey request uses the `project-key` alias to represent a KMS key.

```
$ aws kms generate-data-key --key-id alias/project-key --key-spec AES_256
```

When this request is recorded in the CloudTrail log, the log entry includes both the alias and the key ARN of the actual KMS key that was used.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "ABCDE",
        "arn": "arn:aws:iam::111122223333:role/ProjectDev",
        "accountId": "111122223333",
        "accessKeyId": "FFHIJ",
        "userName": "example-dev"
    },
    "eventTime": "2020-06-29T23:36:41Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "205.205.123.000",
    "userAgent": "aws-cli/1.18.89 Python/3.6.10
 Linux/4.9.217-0.1.ac.205.84.332.metal1.x86_64 botocore/1.17.12",
    "requestParameters": {
        "keyId": "alias/project-key",
        "keySpec": "AES_256"
    },
    "responseElements": null,
    "requestID": "d93f57f5-d4c5-4bab-8139-5a1f7824a363",
    "eventID": "d63001e2-dbc6-4aae-90cb-e5370aca7125",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

For details about logging AWS KMS operations in CloudTrail logs, see Logging AWS KMS API calls with AWS CloudTrail.

Viewing keys

You can use AWS Management Console or the AWS Key Management Service (AWS KMS) API to view AWS KMS keys in each account and Region, including KMS keys that you manage and KMS keys that are managed by AWS.

Topics

Viewing KMS keys in the console

In the AWS Management Console, you can view lists of your KMS keys in the account and Region and details about each KMS key.

Note

The AWS KMS console displays the KMS keys that you have permission to view in your account and Region. KMS keys in other AWS accounts do not appear in the console, even if you have permission to view, manage, and use them. To view KMS keys in other accounts, use the DescribeKey operation.

Topics

Navigating to the key tables

The AWS KMS keys in each account and Region are displayed in tables. There are separate tables for the KMS keys that you create and the KMS keys that AWS services create for you.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. To view the keys in your account that you create and manage, in the navigation pane choose **Customer managed keys**. To view the keys in your account that AWS creates and manages for you, in the navigation pane, choose **AWS managed keys**. For information about the different types of KMS keys, see AWS KMS keys.

   Tip

   To view AWS managed keys that are missing an alias, use the **Customer managed keys** page.
   The AWS KMS console also displays the custom key stores in the account and Region. KMS keys that you create in custom key stores appear on the **Customer managed keys** page. For information about custom key stores, see Custom key stores.

Navigating to key details

There is a details page for every AWS KMS key in the account and Region. The details page displays the **General configuration** section for the KMS key and includes tabs that let authorized users view and manage the **Cryptographic configuration** and **Key policy** for the key. Depending on the type of key, the detail page might also include **Aliases**, **Key material**, **Key rotation**, **Public key**, **Regionality** and **Tags** tabs.

To navigate to the key details page for a KMS key.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. To view the keys in your account that you create and manage, in the navigation pane choose **Customer managed keys**. To view the keys in your account that AWS creates and manages for you, in the navigation pane, choose **AWS managed keys**. For information about the different types of KMS keys, see AWS KMS key.
4. To open the key details page, in the key table, choose the key ID or alias of the KMS key.

   If the KMS key has multiple aliases, an alias summary (**+*n* more**) appears beside the name of the one of the aliases. Choosing the alias summary takes you directly to the **Aliases** tab on the key details page.

Sorting and filtering your KMS keys

To make it easier to find your KMS keys in the console, you can sort and filter the key tables.

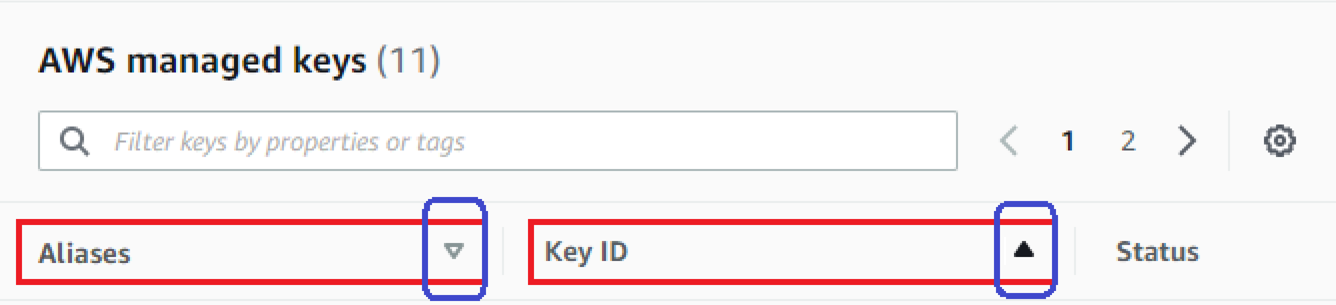
Sort

You can sort KMS keys in ascending or descending order by their column values. This feature sorts all KMS keys in the table, even if they don't appear on the current table page.

Sortable columns are indicated by an arrow beside the column name. On the **AWS managed keys** page, you can sort by **Aliases** or **Key ID**. On the **Customer managed keys** page, you can sort by **Aliases**, **Key ID**, or **Key type**.

To sort in ascending order, choose the column heading until the arrow points upward. To sort in descending order, choose the column heading until the arrow points downward. You can sort by only one column at a time.

For example, you can sort KMS keys in ascending order by key ID, instead of aliases, which is the default.



When you sort KMS keys on the **Customer managed keys** page in ascending order by **Key type**, all asymmetric keys are displayed before all symmetric keys.

Filter

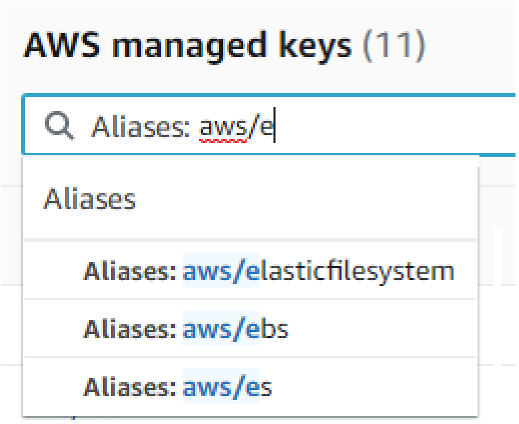
You can filter KMS keys by their property values or tags. The filter applies to all KMS keys in the table, even if they don't appear on the current table page. The filter is not case-sensitive.

Filterable properties are listed in the filter box. On the **AWS managed keys** page, you can filter by alias and key ID. On the **Customer managed keys** page, you can filter by the alias, key ID, and key type properties, and by tags.

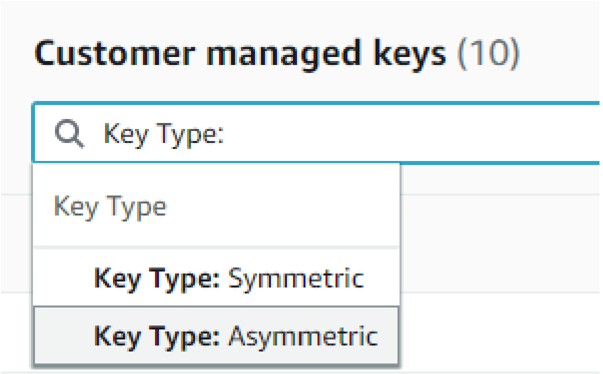
- On the **AWS managed keys** page, you can filter by alias and key ID.
- On the **Customer managed keys** page, you can filter by tags, or by the alias, key ID, key type, or regionality properties.

To filter by a property value, choose the filter, choose the property name, and then choose from the list of actual property values. To filter by a tag, choose the tag key, and then choose from the list of actual tag values. After choosing a property or tag key, you can also type all or part of the property value or tag value. You'll see a preview of the results before you make your choice.

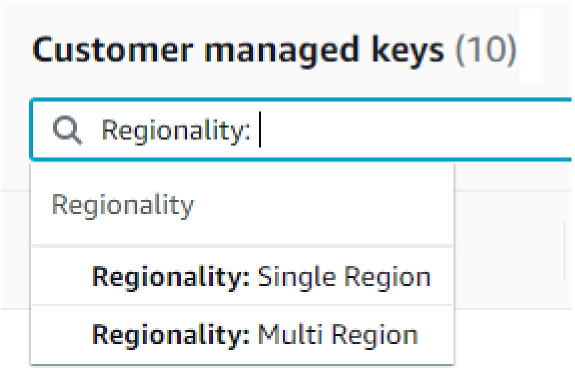
For example, to display KMS keys with an alias name that contains `aws/e`, choose the filter box, choose **Alias**, type `aws/e`, and then press `Enter` or `Return` to add the filter.



To display only asymmetric KMS keys on the **Customer managed keys** page, click the filter box, choose **Key type** and then choose **Key type: Asymmetric**. The **Asymmetric** option appears only when you have asymmetric KMS keys in the table. For more information about identifying asymmetric KMS keys, see Identifying asymmetric KMS keys.



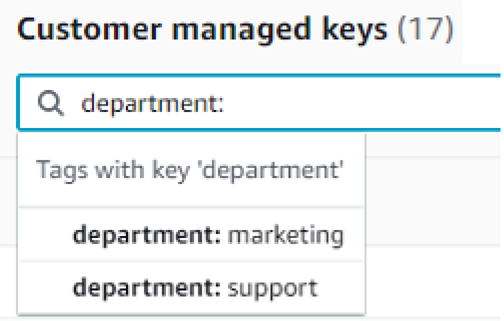
To display only multi-Region keys, on the **Customer managed keys** page, choose the filter box, choose **Regionality** and then choose **Regionality: Multi-Region**. The **Multi-Region** option appears only when you have multi-Region keys in the table. For more information about identifying multi-Region keys, see Viewing multi-Region keys.


Customer managed keys (10)
Regionality:
Regionality
Regionality: Single Region
Regionality: Multi Region

Tag filtering is a bit different. To display only KMS keys with a particular tag, choose the filter box, choose the tag key, and then choose from among the actual tag values. You can also type all or part of the tag value.

The resulting table displays all KMS keys with the chosen tag. However, it doesn't display the tag. To see the tag, choose the key ID or alias of the KMS key and on its detail page, choose the **Tags** tab. The tabs appear below the **General configuration** section.

This filter requires both the tag key and tag value. It won't find KMS keys by typing only the tag key or only its value. To filter tags by all or part of the tag key or value, use the ListResourceTags operation to get tagged KMS keys, then use the filtering features of your programming language. For an example, see ListResourceTags: Get the tags on KMS keys.



Customer managed keys (17)

department:

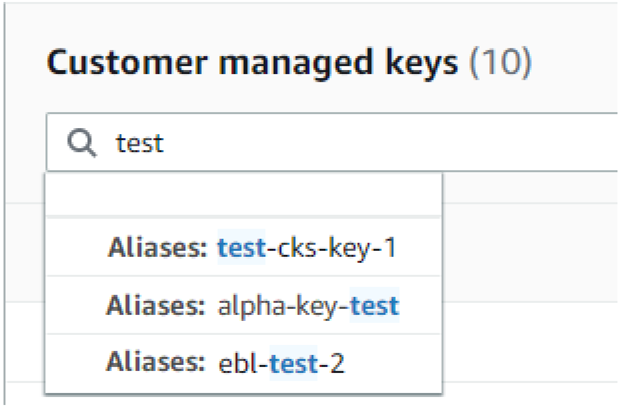
Tags with key 'department'

department: marketing

department: support

To search for text, in the filter box, type all or part of an alias, key ID, key type, or tag key. (After you select the tag key, you can search for a tag value). You'll see a preview of the results before you make your choice.

For example, to display KMS keys with `test` in its tag keys or filterable properties, type `test` in the filter box. The preview shows the KMS keys that the filter will select. In this case, `test` appears only in the **Alias** property.



Customer managed keys (10)

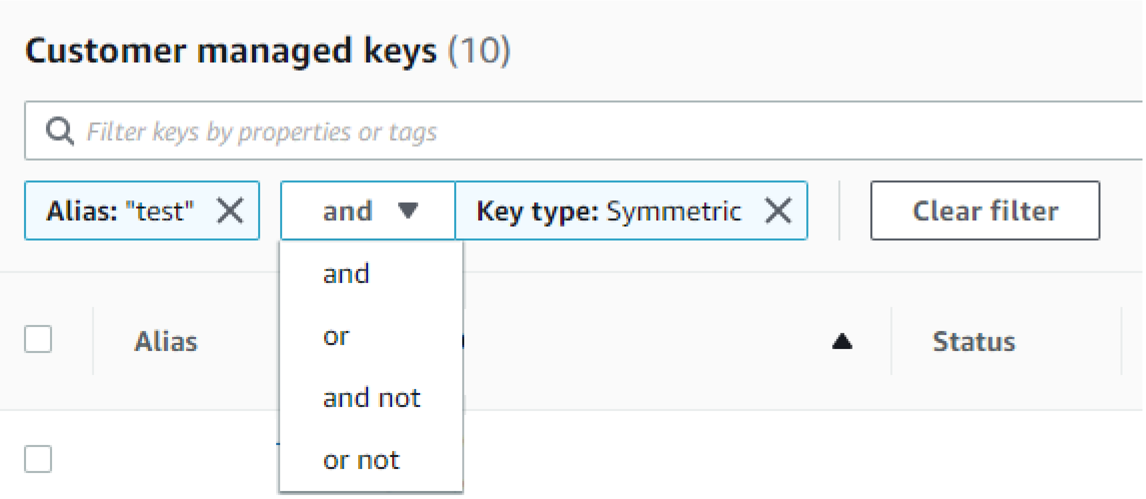
test

Aliases: test-cks-key-1

Aliases: alpha-key-test

Aliases: ebl-test-2

You can use multiple filters at the same time. When you add additional filters, you can also select a logical operator.

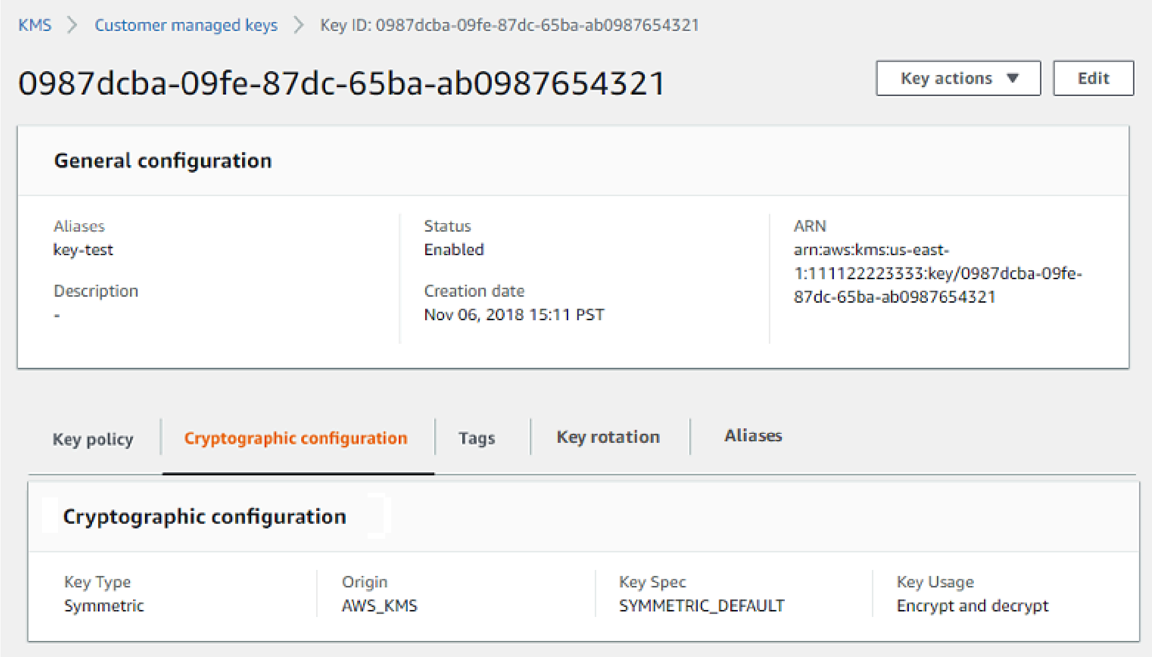


Displaying KMS key details

The details page for each KMS key displays the properties of the KMS key. It differs slightly for the different types of KMS keys.

To display detailed information about a KMS key, on the **AWS managed keys** or **Customer managed keys** page, choose the alias or key ID of the KMS key.

The details page for a KMS key includes a **General Configuration** section that displays the basic properties of the KMS key. It also includes tabs on which you can view and edit properties of the KMS key, such as **Key policy**, **Cryptographic configuration**, **Tags**, **Key material** (for KMS keys with imported key material), **Key rotation** (for symmetric encryption KMS keys), **Regionality** (for multi-Region keys), and **Public key** (for asymmetric KMS keys).


KMS > Customer managed keys > Key ID: 0987dcba-09fe-87dc-65ba-ab0987654321
Key actions ▼
Edit
Key policy
Cryptographic configuration
Tags
Key rotation
Aliases

0987dcba-09fe-87dc-65ba-ab0987654321

General configuration

Aliases
key-test

Description
-

Status
Enabled

Creation date
Nov 06, 2018 15:11 PST

ARN
arn:aws:kms:us-east-1:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321

Cryptographic configuration

Key Type	Origin	Key Spec	Key Usage
Symmetric	AWS_KMS	SYMMETRIC_DEFAULT	Encrypt and decrypt

The following list describes the fields in the detailed display, including field in the tabs. Some of these fields are also available as columns in the table display.

Aliases

Where: Aliases tab

A friendly name for the KMS key. You can use an alias to identify the KMS key in the console and in some AWS KMS APIs. For details, see Using aliases.

The **Aliases** tab displays all aliases associated with the KMS key in the AWS account and Region.

ARN

Where: General configuration section

The Amazon Resource Name (ARN) of the KMS key. This value uniquely identifies the KMS key. You can use it to identify the KMS key in AWS KMS API operations.

Connection state

Indicates whether a custom key store is connected to its backing key store. This field appears only when the KMS key is created in a custom key store.

For information about the values in this field, see ConnectionState in the *AWS KMS API Reference*.

Creation date

Where: General configuration section

The date and time that the KMS key was created. This value is displayed in local time for the device. The time zone does not depend on the Region.

Unlike **Expiration**, the creation refers only to the KMS key, not its key material.

CloudHSM cluster ID

Where: Cryptographic configuration tab

The cluster ID of the AWS CloudHSM cluster that contains the key material for the KMS key. This field appears only when the KMS key is created in a custom key store.

If you choose the CloudHSM cluster ID, it opens the **Clusters** page in the AWS CloudHSM console.

Custom key store ID

Where: Cryptographic configuration tab

The ID of the custom key store that contains the KMS key. This field appears only when the KMS key is created in a custom key store.

If you choose the custom key store ID, it opens the **Custom key stores** page in the AWS KMS console.

Custom key store name

Where: Cryptographic configuration tab

The name of the custom key store that contains the KMS key. This field appears only when the KMS key is created in a custom key store.

Custom key store type

Where: Cryptographic configuration tab

Indicates whether the custom key store is an AWS CloudHSM key store or an external key store. This field appears only when the KMS key is created in a custom key store.

Description

Where: General configuration section

A brief, optional description of the KMS key that you can write and edit. To add or update the description of a customer managed key, above **General Configuration**, choose **Edit**.

Encryption algorithms

Where: Cryptographic configuration tab

Lists the encryption algorithms that can be used with the KMS key in AWS KMS. This field appears only when the **Key type** is **Asymmetric** and the **Key usage** is **Encrypt and decrypt**. For information about the encryption algorithms that AWS KMS supports, see SYMMETRIC_DEFAULT key spec and RSA key specs for encryption and decryption.

Expiration date

Where: Key material tab

The date and time when the key material for the KMS key expires. This field appears only for KMS keys with imported key material, that is, when the **Origin** is **External** and the KMS key has key material that expires.

External key ID

Where: Cryptographic configuration tab

The ID of the external key that is associated with a KMS key in an external key store. This field appears only for KMS keys in an external key store.

External key status

Where: Cryptographic configuration tab

The most recent status that the external key store proxy reported for the external key associated with the KMS key. This field appears only for KMS keys in an external key store.

External key usage

Where: Cryptographic configuration tab

The cryptographic operations that are enabled on the external key associated with the KMS key. This field appears only for KMS keys in an external key store.

Key policy

Where: Key policy tab

Controls access to the KMS key along with IAM policies and grants. Every KMS key has one key policy. It is the only mandatory authorization element. To change the key policy of a customer managed key, on the **Key policy** tab, choose **Edit**. For details, see the section called "Key policies".

Key rotation

Where: Key rotation tab

Enables and disables automatic rotation of the key material in a customer managed KMS key. To change the key rotation status of a customer managed key, use the check box on the **Key rotation** tab.

You can't enable or disable rotation of the key material in an AWS managed key. AWS managed keys are automatically rotated every year.

Key spec

Where: Cryptographic configuration tab

The type of key material in the KMS key. AWS KMS supports symmetric encryption KMS keys (SYMMETRIC_DEFAULT), HMAC KMS keys of different lengths, KMS keys for RSA keys of different lengths, and elliptic curve keys with different curves. For details, see Key spec.

Key type

Where: Cryptographic configuration tab

Indicates whether the KMS key is **Symmetric** or **Asymmetric**.

Key usage

Where: Cryptographic configuration tab

Indicates whether a KMS key can be used for **Encrypt and decrypt**, **Sign and verify** or **Generate and verify MAC**. For details, see Key usage.

Origin

Where: Cryptographic configuration tab

The source of the key material for the KMS key. Valid values are:

- **AWS KMS** for key material that AWS KMS generates
- **AWS CloudHSM** for KMS keys in AWS CloudHSM key store
- **External** for imported key material (BYOK)
- **External key store** for KMS keys in an external key store

MAC algorithms

Where: Cryptographic configuration tab

Lists the MAC algorithms that can be used with an HMAC KMS key in AWS KMS. This field appears only when the **Key spec** is an HMAC key spec (HMAC_*). For information about the MAC algorithms that AWS KMS supports, see Key specs for HMAC KMS keys.

Primary key

Where: Regionality tab

Indicates that this KMS key is a multi-Region primary key. Authorized users can use this section to change the primary key to a different related multi-Region key. This field appears only when the KMS key is a multi-Region primary key.

Public key

Where: Public key tab

Displays the public key of an asymmetric KMS key. Authorized users can use this tab to copy and download the public key.

Regionality

Where: General configuration section and Regionality tabs

Indicates whether a KMS key is a single-Region key, a multi-Region primary key, or a multi-Region replica key. This field appears only when the KMS key is a multi-Region key.

Related multi-Region keys

Where: Regionality tab

Displays all related multi-Region primary and replica keys, except for the current KMS key. This field appears only when the KMS key is a multi-Region key.

In the **Related multi-Region keys** section of a primary key, authorized users can create new replica keys.

Replica key

Where: Regionality tab

Indicates that this KMS key is a multi-Region replica key. This field appears only when the KMS key is a multi-Region replica key.

Signing algorithms

Where: Cryptographic configuration tab

Lists the signing algorithms that can be used with the KMS key in AWS KMS. This field appears only when the **Key type** is **Asymmetric** and the **Key usage** is **Sign and verify**. For information about the signing algorithms that AWS KMS supports, see RSA key specs for signing and verification and Elliptic curve key specs.

Status

Where: General configuration section

The key state of the KMS key. You can use the KMS key in cryptographic operations only when the status is **Enabled**. For a detailed description of each KMS key status and its effect on the operations that you can run on the KMS key, see Key states of AWS KMS keys.

Tags

Where: Tags tab

Optional key-value pairs that describe the KMS key. To add or change the tags for a KMS key, on the **Tags** tab, choose **Edit**.

When you add tags to your AWS resources, AWS generates a cost allocation report with usage and costs aggregated by tags. Tags can also be used to control access to a KMS key. For information about tagging KMS keys, see Tagging keys and ABAC for AWS KMS.

Customizing your KMS key tables

You can customize the tables that appear on the **AWS managed keys** and **Customer managed keys** pages in the AWS Management Console to suit your needs. You can choose the table columns, the number of AWS KMS keys on each page (**Page size**), and the text wrap. The configuration you choose is saved when you confirm it and reapplied whenever you open the pages.

To customize your KMS key tables

1. On the **AWS managed keys** or **Customer managed keys** page, choose the settings icon (⚙) in the upper-right corner of the page.
2. On the **Preferences** page, choose your preferred settings, and then choose **Confirm**.

Consider using the **Page size** setting to increase the number of KMS keys displayed on each page, especially if you typically use a device that's easy to scroll.

The data columns that you display might vary depending on the table, your job role, and the types of KMS keys in the account and Region. The following table offers some suggested configurations. For descriptions of the columns, see Displaying KMS key details.

Suggested KMS key table configurations

You can customize the columns that appear in your KMS key table to display the information you need about your KMS keys.

AWS managed keys

By default, the **AWS managed key** table displays the **Aliases**, **Key ID**, and **Status** columns. These columns are ideal for most use cases.

Symmetric encryption KMS keys

If you use only symmetric encryption KMS keys with key material generated by AWS KMS, the **Aliases**, **Key ID**, **Status**, and **Creation date** columns are likely to be the most useful.

Asymmetric KMS keys

If you use asymmetric KMS keys, in addition to the **Aliases**, **Key ID**, and **Status** columns, consider adding the **Key type**, **Key spec**, and **Key usage** columns. These columns will show you whether a KMS key is symmetric or asymmetric, the type of key material, and whether the KMS key can be used for encryption or signing.

HMAC KMS keys

If you use HMAC KMS keys, in addition to the **Aliases**, **Key ID**, and **Status** columns, consider adding the **Key spec** and **Key usage** columns. These columns will show you whether a KMS key is an HMAC key. Because you can't sort KMS keys by key spec or key usage, use aliases and tags to identify your HMAC keys and then use the filter features of the AWS KMS console to filter by aliases or tags.

Imported key material

If you have KMS keys with imported key material, consider adding the **Origin** and **Expiration date** columns. These columns will show you whether the key material in a KMS key is imported or generated by AWS KMS and when the key material expires, if at all. The **Creation date** field displays the date that the KMS key was created (without key material). It doesn't reflect any characteristic of the key material.

Keys in custom key stores

If you have KMS keys in custom key stores, consider adding the **Origin** and **Custom key store ID** columns. These columns show that the KMS key is in a custom key store, display the custom key store type, and identify the custom key store.

Multi-Region keys

If you have multi-Region keys, consider adding the **Regionality** column. This shows whether a KMS key is a single-Region key, a multi-Region primary key or a multi-Region replica key.

Viewing KMS keys with the API

You can use the AWS Key Management Service (AWS KMS) API to view your KMS keys. This section demonstrates several operations that return details about existing KMS keys. The examples use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

Topics

ListKeys: Get the ID and ARN of all KMS keys

The ListKeys operation returns the ID and Amazon Resource Name (ARN) of all KMS keys in the account and Region.

For example, this call to the `ListKeys` operation returns the ID and ARN of each KMS key in this fictitious account. For examples in multiple programming languages, see Getting key IDs and key ARNs of KMS keys.

```
$ aws kms list-keys

{
  "Keys": [
    {
      "KeyArn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    {
      "KeyArn": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321",
      "KeyId": "0987dcba-09fe-87dc-65ba-ab0987654321"
    },
    {
      "KeyArn": "arn:aws:kms:us-
east-2:111122223333:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d",
      "KeyId": "1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d"
    }
}
```

DescribeKey: Get detailed information about a KMS key

The DescribeKey operation returns details about the specified KMS key. To identify the KMS key, use the key ID, key ARN, alias name, or alias ARN.

Unlike the ListKeys operation, which displays only KMS keys in the caller's account and Region, authorized users can use the `DescribeKey` operation to get details about KMS keys in other accounts.

> **Note**
>
> The `DescribeKey` response includes both `KeySpec` and `CustomerMasterKeySpec` members with the same values. The `CustomerMasterKeySpec` member is deprecated.

For example, this call to `DescribeKey` returns information about a symmetric encryption KMS key. The fields in the response vary with the AWS KMS key spec, key state, and the key material origin. For examples in multiple programming languages, see Viewing an AWS KMS key.

```
$ aws kms describe-key --key-id 1234abcd-12ab-34cd-56ef-1234567890ab

{
    "KeyMetadata": {
        "Origin": "AWS_KMS",
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "Description": "",
        "KeyManager": "CUSTOMER",
        "Enabled": true,
        "KeySpec": "SYMMETRIC_DEFAULT",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "CreationDate": 1499988169.234,
        "MultiRegion": false,
        "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "AWSAccountId": "111122223333",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ]
    }
}
```

This example calls `DescribeKey` operation on an asymmetric KMS key used for signing and verification. The response includes the signing algorithms that AWS KMS supports for this KMS key.

```
$ aws kms describe-key --key-id 0987dcba-09fe-87dc-65ba-ab0987654321

{
    "KeyMetadata": {
        "KeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
        "Origin": "AWS_KMS",
        "Arn": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321",
        "KeyState": "Enabled",
        "KeyUsage": "SIGN_VERIFY",
        "CreationDate": 1569973196.214,
        "Description": "",
        "KeySpec": "ECC_NIST_P521",
        "CustomerMasterKeySpec": "ECC_NIST_P521",
        "AWSAccountId": "111122223333",
        "Enabled": true,
        "MultiRegion": false,
        "KeyManager": "CUSTOMER",
        "SigningAlgorithms": [
            "ECDSA_SHA_512"
        ]
    }
}
```

GetKeyPolicy: Get the key policy attached to a KMS key

The GetKeyPolicy operation gets the key policy that is attached to the KMS key. To identify the KMS key, use its key ID or key ARN. You must also specify the policy name, which is always `default`. (If your output is difficult to read, add the `--output text` option to your command.) `GetKeyPolicy` works only on KMS keys in the caller's account and Region.

For examples in multiple programming languages, see Getting a key policy.

```
$ aws kms get-key-policy --key-id 1234abcd-12ab-34cd-56ef-1234567890ab --policy-name
 default

{
  "Version" : "2012-10-17",
  "Id" : "key-default-1",
  "Statement" : [ {
    "Sid" : "Enable IAM User Permissions",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::111122223333:root"
    },
    "Action" : "kms:*",
    "Resource" : "*"
  } ]
}
```

ListAliases: Get alias names and ARNs for KMS keys

The ListAliases operation returns aliases in the account and Region. The `TargetKeyId` in the response displays the key ID of the KMS key that the alias refers to, if any.

By default, the **ListAliases** command returns all aliases in the account and region. This includes aliases that you created and associated with your customer managed keys, and aliases that AWS created and associated with AWS managed key in your account. You can recognize AWS aliases because their names have the format `aws/`*`<service-name>`*, such as `aws/dynamodb`.

The response might also include aliases without the `TargetKeyId` field, such as the `aws/redshift` alias in this example. These are predefined aliases that AWS has created but has not yet associated with a KMS key.

For examples in multiple programming languages, see Listing aliases.

```
$ aws kms list-aliases

{
    "Aliases": [
        {
            "AliasName": "alias/access-key",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/access-key",
            "TargetKeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
            "CreationDate": 1516435200.399,
            "LastUpdatedDate": 1516435200.399
        },
        {
            "AliasName": "alias/financeKey",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/financeKey",
            "TargetKeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
            "CreationDate": 1604958290.014,
            "LastUpdatedDate": 1604958290.014
        },
        {
            "AliasName": "alias/ECC-P521-Sign",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/ECC-P521-Sign",
            "TargetKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
            "CreationDate": 1693622000.704,
            "LastUpdatedDate": 1693622000.704
        },
        {
            "AliasName": "alias/ImportedKey",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/ImportedKey",
            "TargetKeyId": "1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d",
            "CreationDate": 1493622000.704,
            "LastUpdatedDate": 1521097200.235
        },
        {
            "AliasName": "alias/aws/dynamodb",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/aws/dynamodb",
            "TargetKeyId": "0987ab65-43cd-21ef-09ab-87654321cdef",
            "CreationDate": 1521097200.454,
            "LastUpdatedDate": 1521097200.454
        },
        {
            "AliasName": "alias/aws/ebs",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/aws/ebs",
            "TargetKeyId": "abcd1234-09fe-ef90-09fe-ab0987654321",
            "CreationDate": 1466518990.200,
            "LastUpdatedDate": 1466518990.200
        },
        {
            "AliasName": "alias/aws/redshift",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/aws/redshift"
        },
    ]
}
```

To get the aliases that refer to a particular KMS key, use the KeyId parameter. The parameter value can be the key ID or key ARN. You cannot specify an alias name or alias ARN.

The command in the following example gets the aliases that refer to a customer managed key. But you can use a command like this one to find the aliases that refer to AWS managed keys, too.

```
$ aws kms list-aliases --key-id arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321
{
    "Aliases": [
        {
            "AliasName": "alias/access-key",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/access-key",
            "TargetKeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
            "CreationDate": 1516435200.399,
            "LastUpdatedDate": 1516435200.399
        },
        {
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/financeKey",
            "TargetKeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
            "AliasName": "alias/financeKey",
            "CreationDate": 1604958290.014,
            "LastUpdatedDate": 1604958290.014
        },
    ]
}
```

To get only the aliases for AWS managed keys, use the features of your programming language to filter the response.

```
$ aws kms list-aliases --query 'Aliases[?starts_with(AliasName, `alias/aws/`)]'
```

ListResourceTags: Get the tags on KMS keys

The ListResourceTags operation returns the tags on the specified KMS key. The API returns tags for one KMS key, but you can run the command in a loop to get tags for all KMS keys in the account and Region, or for a set of KMS keys you select. This API returns one page at a time, so if you have numerous tags on numerous KMS keys, you might have to use the paginator in your programming language to get all of the tags you want.

The `ListResourceTags` operation returns tags for all KMS keys, but AWS managed key are not tagged. It works only on KMS keys in the caller's account and Region.

To find the tags for a KMS key, use the `ListResourceTags` operation. The `KeyId` parameter is required. It accepts a key ID or key ARN. Before running this example, replace the example key ARN with a valid one.

```
$  aws kms list-resource-tags --key-id arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
      {
    "Tags": [
        {
            "TagKey": "Department",
            "TagValue": "IT"
        },
        {
            "TagKey": "Purpose",
            "TagValue": "Test"
        }
    ],
    "Truncated": false
}
```

You might want to use the `ListResourceTags` operation to get all KMS keys in the account and Region with a particular tag, tag key, or tag value. To do this, use the filtering features of your programming language.

For example, the following Bash script uses the ListKeys and `ListResourceTags` operations to get all KMS keys in the account and Region with a `Project` tag key. Both of these operations get only the first page of results. If you have numerous KMS keys or numerous tags, use the pagination features of your language to get the entire result from each operation. Before running this example, replace the example key IDs with valid ones.

```
TARGET_TAG_KEY='Project'

for key in $(aws kms list-keys --query 'Keys[*].KeyId' --output text); do
  key_tags=$(aws kms list-resource-tags --key-id "$key" --query "Tags[?TagKey==\`
$TARGET_TAG_KEY\`]")
  if [ "$key_tags" != "[]" ]; then
    echo "Key: $key"
    echo "$key_tags"
  fi
done
```

The output is formatted like the following example output.

```
Key: 0987dcba-09fe-87dc-65ba-ab0987654321
[
    {
        "TagKey": "Project",
        "TagValue": "Gamma"
    }
]
Key: 1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d
[
    {
        "TagKey": "Project",
        "TagValue": "Alpha"
    }
]
Key: 0987ab65-43cd-21ef-09ab-87654321cdef
[
    {
        "TagKey": "Project",
        "TagValue": "Alpha"
    }
]
```

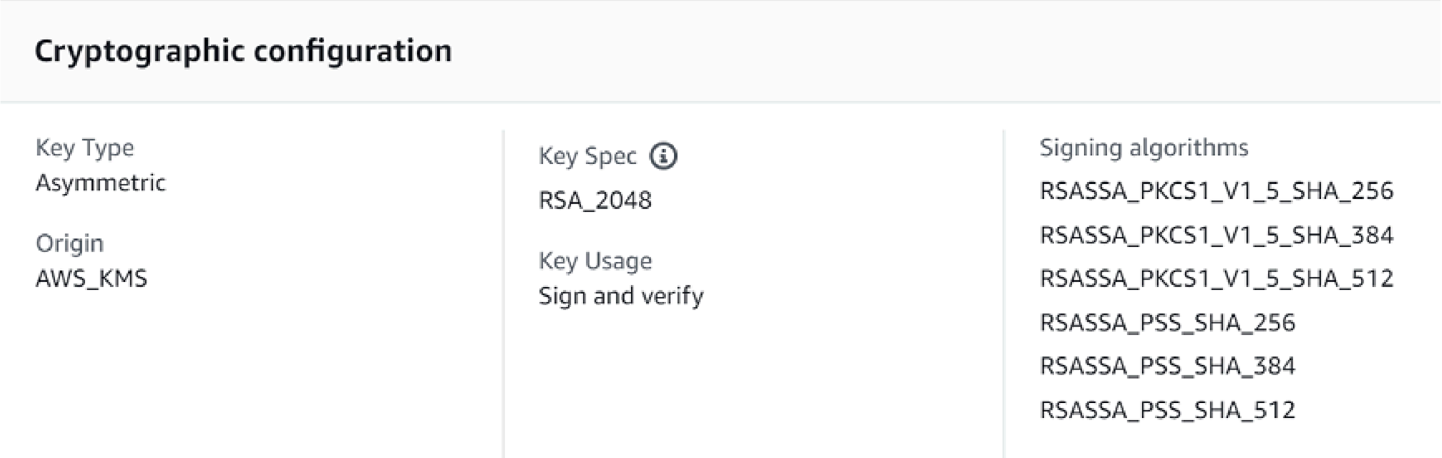
Viewing the cryptographic configuration of KMS keys

After you create your KMS key, you can view its cryptographic configuration. You cannot change the configuration of a KMS key after it is created. If you prefer a different configuration, delete the KMS key and create it again.

You can find the cryptographic configuration of your KMS keys, include the key spec, key usage, and supported encryption or signing algorithms, in the AWS KMS console or by using the AWS KMS API. For details, see Identifying asymmetric KMS keys.

In the AWS KMS console, the details page for each KMS key includes a **Cryptographic configuration** tab that displays cryptographic details about your KMS keys. For example, the following image shows the **Cryptographic configuration** tab for an RSA KMS key used for signing and verification.

The **Cryptographic configuration** tab for some special purpose KMS keys has additional specialized sections. For example, the **Cryptographic configuration** tab for a KMS key in a custom key store has a **Custom key stores** section. The **Cryptographic configuration** tab for a KMS key in an external key store has an **External key** section.



Cryptographic configuration

Key Type
Asymmetric

Origin
AWS_KMS

Key Spec
RSA_2048

Key Usage
Sign and verify

Signing algorithms
RSASSA_PKCS1_V1_5_SHA_256
RSASSA_PKCS1_V1_5_SHA_384
RSASSA_PKCS1_V1_5_SHA_512
RSASSA_PSS_SHA_256
RSASSA_PSS_SHA_384
RSASSA_PSS_SHA_512

In the AWS KMS API, use the DescribeKey operation. The `KeyMetadata` structure in the response includes the cryptographic configuration of the KMS key. For example, `DescribeKey` returns the following response for an RSA KMS key used for signing and verification.

```
{
  "KeyMetadata": {

    "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "AWSAccountId": "111122223333",
    "CreationDate": 1571767572.317,
    "CustomerMasterKeySpec": "RSA_2048",
    "Description": "",
    "Enabled": true,
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "KeyManager": "CUSTOMER",
    "KeyState": "Enabled",
    "MultiRegion": false,
    "Origin": "AWS_KMS",
    "KeySpec": "RSA_2048",
    "KeyUsage": "SIGN_VERIFY",
    "SigningAlgorithms": [
        "RSASSA_PKCS1_V1_5_SHA_256",
        "RSASSA_PKCS1_V1_5_SHA_384",
        "RSASSA_PKCS1_V1_5_SHA_512",
        "RSASSA_PSS_SHA_256",
        "RSASSA_PSS_SHA_384",
        "RSASSA_PSS_SHA_512"
    ]
  }
}
```

Finding the key ID and key ARN

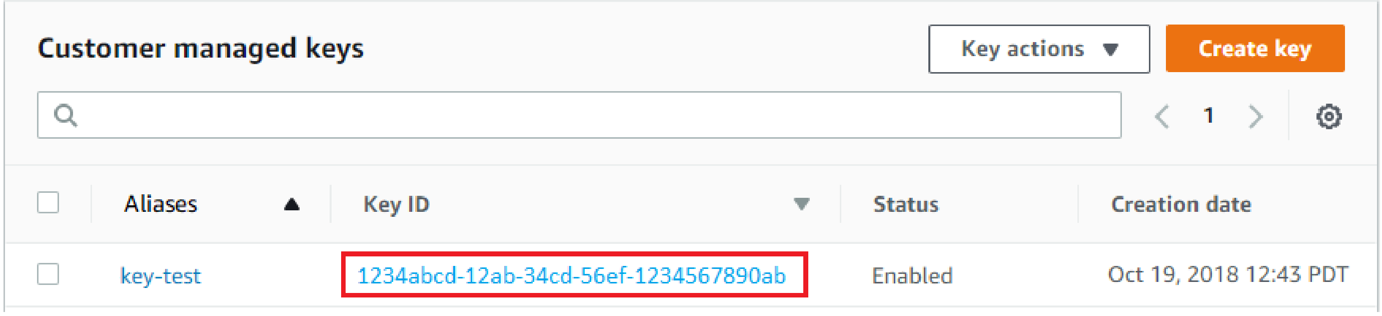
To identify an AWS KMS key, you can use the key ID or the Amazon Resource Name (key ARN). In cryptographic operations, you can also use the alias name or alias ARN.

For detailed information about the KMS key identifiers supported by AWS KMS, see Key identifiers (KeyId). For help finding an alias name and alias ARN, see Finding the alias name and alias ARN.

To find the key ID and ARN (console)

1. Open the AWS KMS console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. To view the keys in your account that you create and manage, in the navigation pane choose **Customer managed keys**. To view the keys in your account that AWS creates and manages for you, in the navigation pane, choose **AWS managed keys**.
4. To find the key ID for a KMS key, see the row that begins with the KMS key alias.

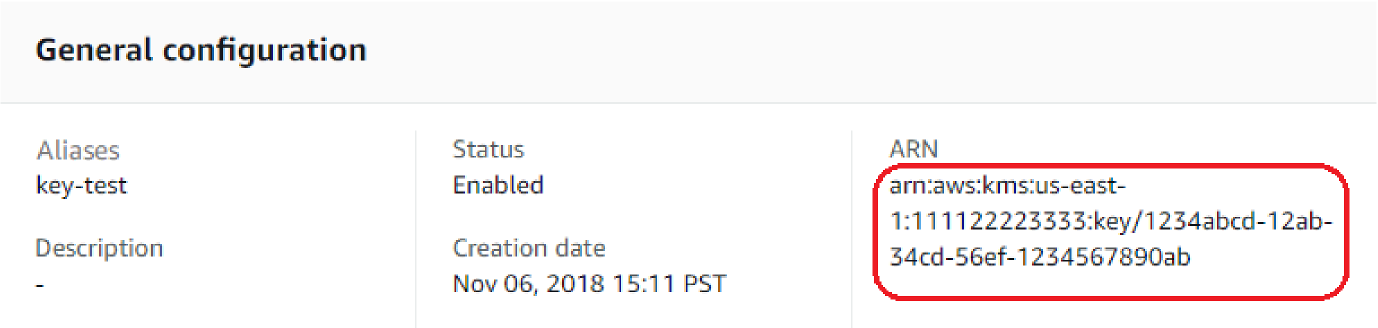
   The **Key ID** column appears in the tables by default. If the Key ID column doesn't appear in your table, use the procedure described in the section called “Customizing your KMS key tables” to restore it. You can also view the key ID of a KMS key on its details page.



Customer managed keys

□	Aliases ▲	Key ID ▼	Status	Creation date
□	key-test	1234abcd-12ab-34cd-56ef-1234567890ab	Enabled	Oct 19, 2018 12:43 PDT

5. To find the Amazon Resource Name (ARN) of the KMS key, choose the key ID or alias. The key ARN appears in the **General Configuration** section.



General configuration

Aliases
key-test

Description
-

Status
Enabled

Creation date
Nov 06, 2018 15:11 PST

ARN
arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

To find the key ID and key ARN (AWS KMS API)

To find the key ID and key ARN of an AWS KMS key, use the ListKeys operation. For examples in multiple programming languages, see Getting key IDs and ARNs and Get key IDs and ARNs.

The `ListKeys` response includes the key ID and key ARN for every KMS key in the account and Region.

```
$ aws kms list-keys
{
    "Keys": [
        {
            "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
            "KeyArn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        },
        {
            "KeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
            "KeyArn": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321"
        }
    ]
}
```

Finding the alias name and alias ARN

An alias is a friendly name for an AWS KMS AWS KMS keys (KMS key). You can find the alias name and alias ARN in the AWS KMS console or AWS KMS API.

For detailed information about the KMS key identifiers that AWS KMS supports, see Key identifiers (KeyId). For help finding the key ID and key ARN, see Finding the key ID and key ARN.

Topics

- To find the alias name and alias ARN (console)
- To find the alias name and alias ARN (AWS KMS API)

To find the alias name and alias ARN (console)

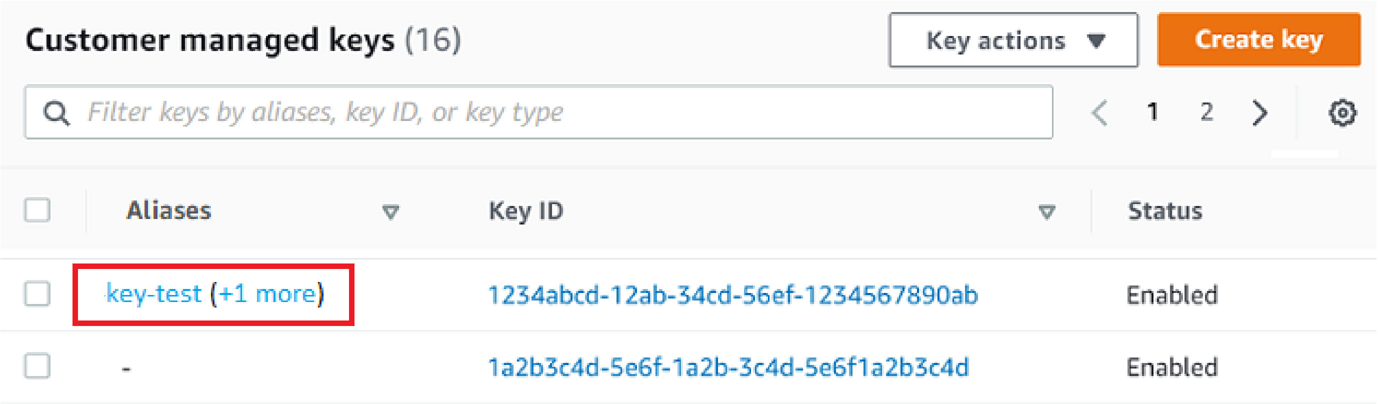
The AWS KMS console displays the aliases associated with the KMS key.

1. Open the AWS KMS console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. To view the keys in your account that you create and manage, in the navigation pane choose **Customer managed keys**. To view the keys in your account that AWS creates and manages for you, in the navigation pane, choose **AWS managed keys**.
4. The **Aliases** column displays the alias for each KMS key. If a KMS key does not have an alias, a dash (**-**) appears in the **Aliases** column.

   If a KMS key has multiple aliases, the **Aliases** column also has an alias summary, such as **(+*n* more)**. For example, the following KMS key has two aliases, one of which is `key-test`.

   To find the alias name and alias ARN of all aliases for the KMS key, use the **Aliases** tab.

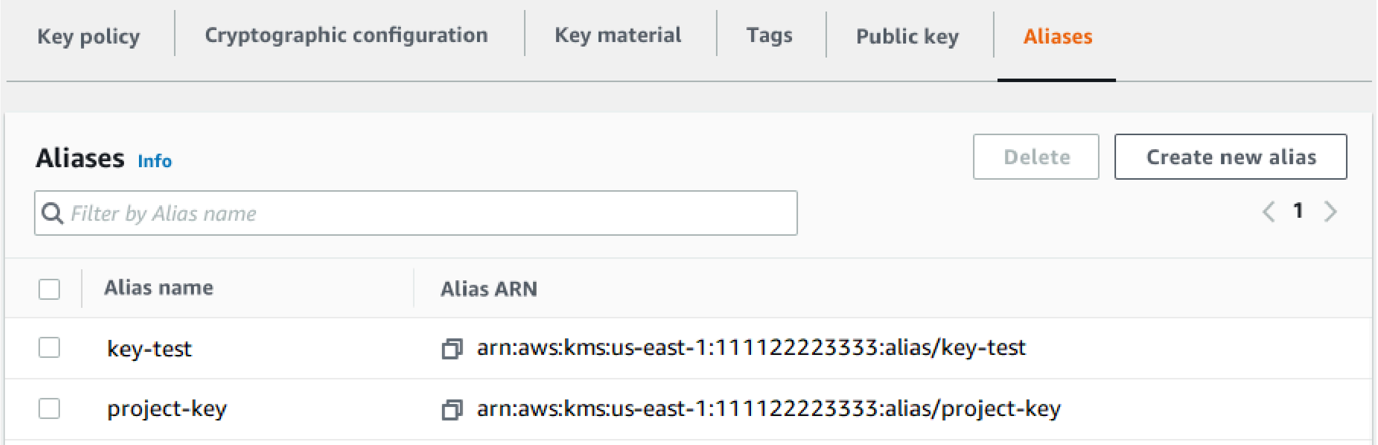
   - To go directly to the **Aliases** tab, in the **Aliases** column, choose the alias summary (**+*n* more**). An alias summary appears only if the KMS key has more than one alias.
   - Or, choose the alias or key ID of the KMS key (which opens the detail page for the KMS key) and then choose the **Aliases** tab. The tabs are under the **General configuration** section.



Customer managed keys (16)

□	Aliases	Key ID	Status
□	key-test (+1 more)	1234abcd-12ab-34cd-56ef-1234567890ab	Enabled
□	-	1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d	Enabled

5. The **Aliases** tab displays the alias name and alias ARN of all aliases for a KMS key. You can also create and delete aliases for the KMS key on this tab.



Aliases

□	Alias name	Alias ARN
□	key-test	arn:aws:kms:us-east-1:111122223333:alias/key-test
□	project-key	arn:aws:kms:us-east-1:111122223333:alias/project-key

To find the alias name and alias ARN (AWS KMS API)

To find the alias name and alias ARN of an AWS KMS key, use the ListAliases operation. For examples in multiple programming languages, see Listing aliases and Get alias names and ARNs.

By default, the response includes the alias name and alias ARN for every alias in the account and Region. To get only the aliases for a particular KMS key, use the `KeyId` parameter.

For example, the following command gets only the aliases for an example KMS key with key ID `1234abcd-12ab-34cd-56ef-1234567890ab`.

```
$ aws kms list-aliases --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
{
    "Aliases": [
        {
            "AliasName": "alias/key-test",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/key-test",
            "TargetKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
            "CreationDate": 1593622000.191,
            "LastUpdatedDate": 1593622000.191
        },
        {
            "AliasName": "alias/project-key",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/project-key",
            "TargetKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
            "CreationDate": 1516435200.399,
            "LastUpdatedDate": 1516435200.399
        }
    ]
}
```

Editing keys

You can change the following properties of your customer managed keys in the AWS KMS console and by using AWS KMS API.

You cannot edit any properties of AWS managed keys or AWS owned keys. These keys are managed by the AWS services that created them.

Description

You can change the description of your customer managed key on the details page for the KMS key or by using the UpdateKeyDescription operation.

To edit the key description in the console, in the upper right corner of the details page for the KMS key, choose **Edit**.

Key policy

You can change the key policy on the **Key policy** tab of the details page for the customer managed key or by using the PutKeyPolicy operation.

For details, see Changing a key policy.

Tags

You can create and delete tags on the **Customer managed keys** page of the AWS KMS console, or on the **Tags** tab of the details page for the customer managed key. Or you can use the TagResource and UntagResource operations.

For details, see Tagging keys.

Enable and disable

You can enable and disable KMS keys on the **Customer managed keys** page of the AWS KMS console, or on the details page for the customer managed key. Or you can use the EnableKey and DisableKey operations.

For details, see Enabling and disabling keys.

Automatic key rotation

You can enable and disable automatic key rotation on the **Key rotation** tab of the details page for the customer managed key or by using the EnableKeyRotation and DisableKeyRotation operations.

For details, see Rotating AWS KMS keys.

See also

Updating aliases

Tagging keys

In AWS KMS, you can add tags to a customer managed key when you create the KMS key, and tag or untag existing KMS keys unless they are pending deletion. You cannot tag aliases, custom key stores, AWS managed keys, AWS owned keys, or KMS keys in other AWS accounts. Tags are optional, but they can be very useful.

For more information, see Creating keys and Editing keys. For general information about tags, including best practices, tagging strategies, and the format and syntax of tags, see Tagging AWS resources in the *Amazon Web Services General Reference*.

Topics

- About tags in AWS KMS
- Managing KMS key tags in the console
- Managing KMS key tags with API operations
- Controlling access to tags
- Using tags to control access to KMS keys

About tags in AWS KMS

A *tag* is an optional metadata label that you can assign (or AWS can assign) to an AWS resource. Each tag consists of a *tag key* and a *tag value*, both of which are case-sensitive strings. The tag value can be an empty (null) string. Each tag on a resource must have a different tag key, but you can add the same tag to multiple AWS resources. Each resource can have up to 50 user-created tags.

Do not include confidential or sensitive information in the tag key or tag value. Tags are accessible to many AWS services, including billing.

In AWS KMS, you can add tags to a customer managed key when you create the KMS key, and tag or untag existing KMS keys unless they are pending deletion. You cannot tag aliases, custom key stores, AWS managed keys, AWS owned keys, or KMS keys in other AWS accounts. Tags are optional, but they can be very useful.

For example, you can add a `"Project"="Alpha"` tag to all KMS keys and Amazon S3 buckets that you use for the Alpha project.

```
TagKey   = "Project"
TagValue = "Alpha"
```

For general information about tags, including the format and syntax, see Tagging AWS resources in the *Amazon Web Services General Reference*.

Tags help you do the following:

- Identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related. For example, you can assign the same tag to an KMS key and an Amazon Elastic Block Store (Amazon EBS) volume or AWS Secrets Manager secret. You can also use tags to identify KMS keys for automation.

- Track your AWS costs. When you add tags to your AWS resources, AWS generates a cost allocation report with usage and costs aggregated by tags. You can use this feature to track AWS KMS costs for a project, application, or cost center.

  For more information about using tags for cost allocation, see Using Cost Allocation Tags in the *AWS Billing User Guide*. For information about the rules for tag keys and tag values, see User-Defined Tag Restrictions in the *AWS Billing User Guide*.
- Control access to your AWS resources. Allowing and denying access to KMS keys based on their tags is part of AWS KMS support for attribute-based access control (ABAC). For information about controlling access to AWS KMS keys based on their tags, see Using tags to control access to KMS keys. For more general information about using tags to control access to AWS resources, see Controlling Access to AWS Resources Using Resource Tags in the *IAM User Guide*.

AWS KMS writes an entry to your AWS CloudTrail log when you use the TagResource, UntagResource, or ListResourceTags operations.

Managing KMS key tags in the console

You can add tags to a KMS key when you create the KMS key in the AWS KMS console. You can also use the **Tags** tab in the console to add, edit, and delete tags on customer managed keys To add, edit, view, and delete tags for a KMS key, you must have the required permissions. For details, see Controlling access to tags.

Add tags while creating a KMS key

To add tags when creating a KMS key in the console, you must have `kms:TagResource` permission in an IAM policy in addition to the permissions required to create KMS keys and view KMS keys in the console. At a minimum, the permission must cover all KMS keys in the account and Region.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**. (You cannot manage the tags of an AWS managed key)
4. Choose the key type, then choose **Next**.
5. Enter an alias and optional description.

6. Enter a tag key and, optionally, a tag value. To add additional tags, choose **Add tag**. To delete a tag, choose **Remove**. When you're done tagging your new KMS key, choose **Next**.
7. Finish creating your KMS key.

View and manage tags on existing KMS keys

To add, view, edit, and delete tags in the console, you need tagging permission on the KMS key. You can get this permission from the key policy for the KMS key or, if the key policy allows it, from an IAM policy that includes the KMS key. You need these permissions in addition to the permissions to view KMS keys in the console.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**. (You cannot manage the tags of an AWS managed key)
4. You can use the table filter to display only KMS keys with particular tags. For details, see Sorting and filtering your KMS keys.
5. Select the check box next to the alias of a KMS key.
6. Choose **Key actions**, **Add or edit tags**.
7. On the details page for KMS key, choose the **Tags** tab.

   - To create your first tag, choose **Create tag**, type a tag key (required) and tag value (optional), and then choose **Save**.

     If you leave the tag value blank, the actual tag value is a null or empty string.
   - To add a tag, choose **Edit**, choose **Add tag**, type a tag key and tag value, and then choose **Save**.
   - To change the name or value of a tag, choose **Edit**, make your changes, and then choose **Save**.
   - To delete a tag, choose **Edit**. On the tag row, choose **Remove**, and then choose **Save**.
8. To save your changes, choose **Save changes**.

Managing KMS key tags with API operations

You can use the AWS Key Management Service (AWS KMS) API to add, delete, and list tags for the KMS keys that you manage. These examples use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language. You cannot tag AWS managed keys.

To add, edit, view, and delete tags for a KMS key, you must have the required permissions. For details, see Controlling access to tags.

Topics

- CreateKey: Add tags to a new KMS key
- TagResource: Add or change tags for a KMS key
- ListResourceTags: Get the tags for a KMS key
- UntagResource: Delete tags from a KMS key

CreateKey: Add tags to a new KMS key

You can add tags when you create a customer managed key To specify the tags, use the `Tags` parameter of the CreateKey operation.

To add tags when creating a KMS key, the caller must have `kms:TagResource` permission in an IAM policy. At a minimum, the permission must cover all KMS keys in the account and Region. For details, see Controlling access to tags.

The value of the `Tags` parameter of `CreateKey` is a collection of case-sensitive tag key and tag value pairs. Each tag on a KMS key must have a different tag name. The tag value can be a null or empty string.

For example, the following AWS CLI command creates a symmetric encryption KMS key with a `Project:Alpha` tag. When specifying more than one key-value pair, use a space to separate each pair.

```
$ aws kms create-key --tags TagKey=Project,TagValue=Alpha
```

When this command is successful, it returns a `KeyMetadata` object with information about the new KMS key. However, the `KeyMetadata` does not include tags. To get the tags, use the ListResourceTags operation.

TagResource: Add or change tags for a KMS key

The TagResource operation adds one or more tags to a KMS key. You cannot use this operation to add or edit tags in a different AWS account.

To add a tag, specify a new tag key and a tag value. To edit a tag, specify an existing tag key and a new tag value. Each tag on a KMS key must have a different tag key. The tag value can be a null or empty string.

For example, the following command adds **Purpose** and **Department** tags to an example KMS key.

```
$ aws kms tag-resource \
         --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
         --tags TagKey=Purpose,TagValue=Pretest TagKey=Department,TagValue=Finance
```

When this command is successful, it does not return any output. To view the tags on a KMS key, use the ListResourceTags operation.

You can also use **TagResource** to change the tag value of an existing tag. To replace a tag value, specify the same tag key with a different value.

For example, this command changes the value of the Purpose tag from Pretest to Test.

```
$ aws kms tag-resource \
         --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
         --tags TagKey=Purpose,TagValue=Test
```

ListResourceTags: Get the tags for a KMS key

The ListResourceTags operation gets the tags for a KMS key. The KeyId parameter is required. You cannot use this operation to view the tags on KMS keys in a different AWS account.

For example, the following command gets the tags for an example KMS key.

```
$ aws kms list-resource-tags --key-id 1234abcd-12ab-34cd-56ef-1234567890ab

  "Truncated": false,
  "Tags": [
      {
        "TagKey": "Project",
        "TagValue": "Alpha"
     },
     {
       "TagKey": "Purpose",
       "TagValue": "Test"
     },
     {
       "TagKey": "Department",
       "TagValue": "Finance"
     }
  ]
}
```

UntagResource: Delete tags from a KMS key

The UntagResource operation deletes tags from a KMS key. To identify the tags to delete, specify the tag keys. You cannot use this operation to delete tags from KMS keys a different AWS account.

When it succeeds, the `UntagResource` operation doesn't return any output. Also, if the specified tag key isn't found on the KMS key, it doesn't throw an exception or return a response. To confirm that the operation worked, use the ListResourceTags operation.

For example, this command deletes the **`Purpose`** tag and its value from the specified KMS key.

```
$ aws kms untag-resource --key-id 1234abcd-12ab-34cd-56ef-1234567890ab --tag-keys
 Purpose
```

Controlling access to tags

To add, view, and delete tags, either in the AWS KMS console or by using the API, principals need tagging permissions. You can provide these permissions in key policies. You can also provide them in IAM policies (including VPC endpoint policies), but only if the key policy allows it. The AWSKeyManagementServicePowerUser managed policy allows principals to tag, untag, and list tags on all KMS keys the account can access.

You can also limit these permissions by using AWS global condition keys for tags. In AWS KMS, these conditions can control access to tagging operations, such as TagResource and UntagResource.

Note

Be cautious when giving principals permission to manage tags and aliases. Changing a tag or alias can allow or deny permission to the customer managed key. For details, see ABAC for AWS KMS and Using tags to control access to KMS keys.

For example policies and more information, see Controlling Access Based on Tag Keys in the *IAM User Guide*.

Permissions to create and manage tags work as follows.

kms:TagResource

Allows principals to add or edit tags. To add tags while creating a KMS key, the principal must have permission in an IAM policy that isn't restricted to particular KMS keys.

kms:ListResourceTags

Allows principals to view tags on KMS keys.

kms:UntagResource

Allows principals to delete tags from KMS keys.

Tag permissions in policies

You can provide tagging permissions in a key policy or IAM policy. For example, the following example key policy gives select users tagging permission on the KMS key. It gives all users who can assume the example Administrator or Developer roles permission to view tags.

```
{
  "Version": "2012-10-17",
  "Id": "example-key-policy",
  "Statement": [
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::111122223333:root"},
      "Action": "kms:*",
      "Resource": "*"
    },
    {
      "Sid": "Allow all tagging permissions",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::111122223333:user/LeadAdmin",
        "arn:aws:iam::111122223333:user/SupportLead"
      ]},
      "Action": [
          "kms:TagResource",
          "kms:ListResourceTags",
          "kms:UntagResource"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow roles to view tags",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::111122223333:role/Administrator",
        "arn:aws:iam::111122223333:role/Developer"
      ]},
      "Action": "kms:ListResourceTags",
      "Resource": "*"
    }
  ]
}
```

To give principals tagging permission on multiple KMS keys, you can use an IAM policy. For this policy to be effective, the key policy for each KMS key must allow the account to use IAM policies to control access to the KMS key.

For example, the following IAM policy allows the principals to create KMS keys. It also allows them to create and manage tags on all KMS keys in the specified account. This combination allows the principals to use the Tags parameter of the CreateKey operation to add tags to a KMS key while they are creating it.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "IAMPolicyCreateKeys",
      "Effect": "Allow",
      "Action": "kms:CreateKey",
      "Resource": "*"
    },
    {
      "Sid": "IAMPolicyTags",
      "Effect": "Allow",
      "Action": [
        "kms:TagResource",
        "kms:UntagResource",
        "kms:ListResourceTags"
      ],
      "Resource": "arn:aws:kms:*:111122223333:key/*"
    }
  ]
}
```

Limiting tag permissions

You can limit tagging permissions by using policy conditions. The following policy conditions can be applied to the `kms:TagResource` and `kms:UntagResource` permissions. For example, you can use the `aws:RequestTag/`*`tag-key`* condition to allow a principal to add only particular tags, or prevent a principal from adding tags with particular tag keys. Or, you can use the `kms:KeyOrigin` condition to prevent principals from tagging or untagging KMS keys with imported key material.

- aws:RequestTag
- aws:ResourceTag/*tag-key* (IAM policies only)
- aws:TagKeys
- kms:CallerAccount
- kms:KeySpec
- kms:KeyUsage
- kms:KeyOrigin
- kms:ViaService

As a best practice when you use tags to control access to KMS keys, use the `aws:RequestTag/`*`tag-key`* or `aws:TagKeys` condition key to determine which tags (or tag keys) are allowed.

For example, the following IAM policy is similar to the previous one. However, this policy allows the principals to create tags (`TagResource`) and delete tags `UntagResource` only for tags with a `Project` tag key.

Because `TagResource` and `UntagResource` requests can include multiple tags, you must specify a `ForAllValues` or `ForAnyValue` set operator with the aws:TagKeys condition. The `ForAnyValue` operator requires that at least one of the tag keys in the request matches one of the tag keys in the policy. The `ForAllValues` operator requires that all of the tag keys in the request match one of the tag keys in the policy. The `ForAllValues` operator also returns `true` if there are no tags in the request, but TagResource and UntagResource fail when no tags are specified. For details about the set operators, see Use multiple keys and values in the *IAM User Guide*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "IAMPolicyCreateKey",
      "Effect": "Allow",
      "Action": "kms:CreateKey",
      "Resource": "*"
    },
    {
      "Sid": "IAMPolicyViewAllTags",
      "Effect": "Allow",
      "Action": "kms:ListResourceTags",
      "Resource": "arn:aws:kms:*:111122223333:key/*"
    },
    {
      "Sid": "IAMPolicyManageTags",
      "Effect": "Allow",
      "Action": [
        "kms:TagResource",
        "kms:UntagResource"
      ],
      "Resource": "arn:aws:kms:*:111122223333:key/*",
      "Condition": {
          "ForAllValues:StringEquals": {"aws:TagKeys": "Project"}
      }
    }
  ]
}
```

Using tags to control access to KMS keys

You can control access to AWS KMS keys based on the tags on the KMS key. For example, you can write an IAM policy that allows principals to enable and disable only the KMS keys that have a particular tag. Or you can use an IAM policy to prevent principals from using KMS keys in cryptographic operations unless the KMS key has a particular tag.

This feature is part of AWS KMS support for attribute-based access control (ABAC). For information about using tags to control access to AWS resources, see What is ABAC for AWS? and Controlling Access to AWS Resources Using Resource Tags in the *IAM User Guide*. For help resolving access issues related to ABAC, see Troubleshooting ABAC for AWS KMS.

> **Note**
>
> It might take up to five minutes for tag and alias changes to affect KMS key authorization. Recent changes might be visible in API operations before they affect authorization.

AWS KMS supports the aws:ResourceTag/*tag-key* global condition context key, which lets you control access to KMS keys based on the tags on the KMS key. Because multiple KMS keys can have the same tag, this feature lets you apply the permission to a select set of KMS keys. You can also easily change the KMS keys in the set by changing their tags.

In AWS KMS, the `aws:ResourceTag/`*`tag-key`* condition key is supported only in IAM policies. It isn't supported in key policies, which apply only to one KMS key, or on operations that don't use a particular KMS key, such as the ListKeys or ListAliases operations.

Controlling access with tags provides a simple, scalable, and flexible way to manage permissions. However, if not properly designed and managed, it can allow or deny access to your KMS keys inadvertently. If you are using tags to control access, consider the following practices.

- Use tags to reinforce the best practice of least privileged access. Give IAM principals only the permissions they need on only the KMS keys they must use or manage. For example, use tags to label the KMS keys used for a project. Then give the project team permission to use only KMS keys with the project tag.
- Be cautious about giving principals the `kms:TagResource` and `kms:UntagResource` permissions that let them add, edit, and delete tags. When you use tags to control access to KMS keys, changing a tag can give principals permission to use KMS keys that they didn't otherwise have permission to use. It can also deny access to KMS keys that other principals require to do

their jobs. Key administrators who don't have permission to change key policies or create grants can control access to KMS keys if they have permission to manage tags.

Whenever possible, use a policy condition, such as `aws:RequestTag/tag-key` or `aws:TagKeys` to limit a principal's tagging permissions to particular tags or tag patterns on particular KMS keys.

- Review the principals in your AWS account that currently have tagging and untagging permissions and adjust them, if necessary. For example, the console default key policy for key administrators includes `kms:TagResource` and `kms:UntagResource` permission on that KMS key. IAM policies might allow tag and untag permissions on all KMS keys. For example, the AWSKeyManagementServicePowerUser managed policy allows principals to tag, untag, and list tags on all KMS keys.
- Before setting a policy that depends on a tag, review the tags on the KMS keys in your AWS account. Make sure that your policy applies only to the tags you intend to include. Use CloudTrail logs and CloudWatch alarms to alert you to tag changes that might affect access to your KMS keys.
- The tag-based policy conditions use pattern matching; they aren't tied to a particular instance of a tag. A policy that uses tag-based condition keys affects all new and existing tags that match the pattern. If you delete and recreate a tag that matches a policy condition, the condition applies to the new tag, just as it did to the old one.

For example, consider the following IAM policy. It allows the principals to call the GenerateDataKeyWithoutPlaintext and Decrypt operations only on KMS keys in your account that are the Asia Pacific (Singapore) Region and have a `"Project"="Alpha"` tag. You might attach this policy to roles in the example Alpha project.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "IAMPolicyWithResourceTag",
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKeyWithoutPlaintext",
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:ap-southeast-1:111122223333:key/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Project": "Alpha"
        }
      }
    }
  ]
}
```

The following example IAM policy allows the principals to use any KMS key in the account for certain cryptographic operations. But it prohibits the principals from using these cryptographic operations on KMS keys with a `"Type"="Reserved"` tag or no `"Type"` tag.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "IAMAllowCryptographicOperations",
      "Effect": "Allow",
      "Action": [
        "kms:Encrypt",
        "kms:GenerateDataKey*",
        "kms:Decrypt",
        "kms:ReEncrypt*"
      ],
      "Resource": "arn:aws:kms:*:111122223333:key/*"
    },
    {
      "Sid": "IAMDenyOnTag",
      "Effect": "Deny",
      "Action": [
        "kms:Encrypt",
        "kms:GenerateDataKey*",
        "kms:Decrypt",
        "kms:ReEncrypt*"
      ],
      "Resource": "arn:aws:kms:*:111122223333:key/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Type": "Reserved"
        }
      }
    },
    {
      "Sid": "IAMDenyNoTag",
      "Effect": "Deny",
      "Action": [
        "kms:Encrypt",
        "kms:GenerateDataKey*",
        "kms:Decrypt",
        "kms:ReEncrypt*"
      ],
      "Resource": "arn:aws:kms:*:111122223333:key/*",
      "Condition": {
        "Null": {
          "aws:ResourceTag/Type": "true"
        }
      }
    }
  ]
}
```

Enabling and disabling keys

You can disable and re-enable customer managed keys. When you create a KMS key, it is enabled by default. If you disable a KMS key, it cannot be used in any cryptographic operation until you re-enable it.

Because it's temporary and easily undone, disabling a KMS key is a safe alternative to deleting a KMS key, an action that is destructive and irreversible. If you are considering deleting a KMS key, disable it first and set a CloudWatch alarm or similar mechanism to be certain that you'll never need to use the key to decrypt encrypted data.

When you disable a KMS key, it becomes unusable right away (subject to eventual consistency). However, resources encrypted with data keys protected by the KMS key are not affected until the the KMS key is used again, such as to decrypt the data key. This issue affects AWS services, many of which use data keys to protect your resources. For details, see How unusable KMS keys affect data keys.

You cannot enable or disable AWS managed keys or AWS owned keys. AWS managed keys are permanently enabled for use by services that use AWS KMS. AWS owned keys are managed solely by the service that owns them.

Note

AWS KMS does not rotate the key material of customer managed keys while they are disabled. For more information, see How key rotation works.

Topics

Enabling and disabling KMS keys (console)

You can use the AWS KMS console to enable and disable customer managed keys.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Choose the check box for the KMS keys that you want to enable or disable.
5. To enable a KMS key, choose **Key actions**, **Enable**. To disable a KMS key, choose **Key actions**, **Disable**.

Enabling and disabling KMS keys (AWS KMS API)

The EnableKey operation enables a disabled AWS KMS key. These examples use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language. The `key-id` parameter is required.

This operation does not return any output. To see the key status, use the DescribeKey operation.

```
$ aws kms enable-key --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
```

The DisableKey operation disables an enabled KMS key. The `key-id` parameter is required.

```
$ aws kms disable-key --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
```

This operation does not return any output. To see the key status, use the DescribeKey operation, and see the `Enabled` field.

```
$ aws kms describe-key --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
{
    "KeyMetadata": {
        "Origin": "AWS_KMS",
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "Description": "",
        "KeyManager": "CUSTOMER",
        "MultiRegion": false,
        "Enabled": false,
        "KeyState": "Disabled",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "CreationDate": 1502910355.475,
        "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "AWSAccountId": "111122223333"
        "KeySpec": "SYMMETRIC_DEFAULT",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ]
    }
}
```

Rotating AWS KMS keys

To create new cryptographic material for your customer managed keys, you can create new KMS keys, and then change your applications or aliases to use the new KMS keys. Or, you can rotate the key material associated with an existing KMS key by enabling automatic key rotation or performing on-demand rotation.

By default, when you enable *automatic key rotation* for a KMS key, AWS KMS generates new cryptographic material for the KMS key every year. You can also specify a custom rotation-period to define the number of days after you enable automatic key rotation that AWS KMS will rotate your key material, and the number of days between each automatic rotation thereafter. If you need to immediately initiate key material rotation, you can perform *on-demand rotation*, regardless of whether or not automatic key rotation is enabled. On-demand rotations do not change existing automatic rotation schedules.

AWS KMS saves all previous versions of the cryptographic material in perpetuity so you can decrypt any data encrypted with that KMS key. AWS KMS does not delete any rotated key material until you delete the KMS key. You can track the rotation of key material for your KMS keys in Amazon CloudWatch, AWS CloudTrail, and the AWS Key Management Service console. You can also use GetKeyRotationStatus operation to verify whether automatic rotation is enabled for a KMS key and identify any in progress on-demand rotations. You can use ListKeyRotations operation to view the details of completed rotations.

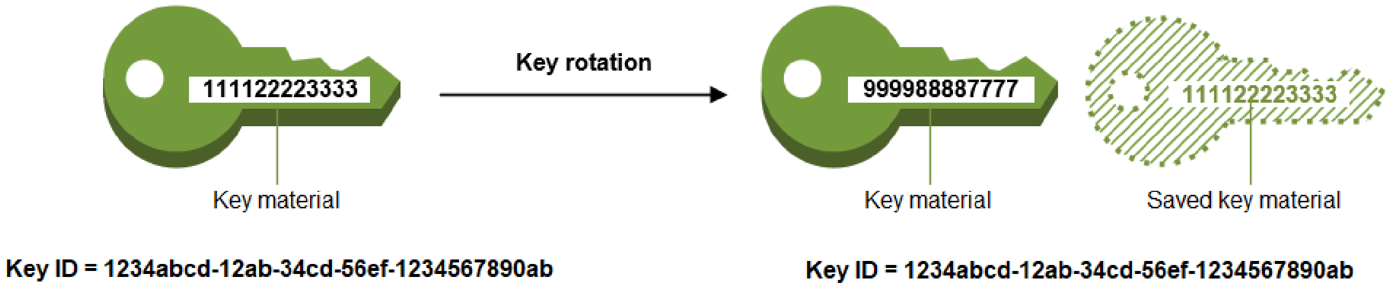
When you use a rotated KMS key to encrypt data, AWS KMS uses the current key material. When you use the rotated KMS key to decrypt ciphertext, AWS KMS uses the version of the key material that was used to encrypt it. You cannot select a particular version of the key material for decrypt operations, AWS KMS automatically chooses the correct version. Because AWS KMS transparently decrypts with the appropriate key material, you can safely use a rotated KMS key in applications and AWS services without code changes.

However, automatic key rotation has no effect on the data that the KMS key protects. It does not rotate the data keys that the KMS key generated or re-encrypt any data protected by the KMS key, and it will not mitigate the effect of a compromised data key.

AWS KMS supports automatic and on-demand key rotation only for symmetric encryption KMS keys with key material that AWS KMS creates. Automatic rotation is optional for customer managed KMS keys. AWS KMS always rotates the key material for AWS managed KMS keys every year. Rotation of AWS owned KMS keys is managed by the AWS service that owns the key.

> **Note**
>
> The rotation period for AWS managed keys changed in May 2022. For details, see AWS managed keys.

Key rotation changes only the *key material*, which is the cryptographic secret that is used in encryption operations. The KMS key is the same logical resource, regardless of whether or how many times its key material changes. The properties of the KMS key do not change, as shown in the following image.


111122223333
Key rotation
999988887777
111122223333
Key material
Key material
Saved key material
Key ID = 1234abcd-12ab-34cd-56ef-1234567890ab
Key ID = 1234abcd-12ab-34cd-56ef-1234567890ab

You might decide to create a new KMS key and use it in place of the original KMS key. This has the same effect as rotating the key material in an existing KMS key, so it's often thought of as manually rotating the key. Manual rotation is a good choice when you want to rotate KMS keys that are not eligible for automatic key rotation, including asymmetric KMS keys, HMAC KMS keys, KMS keys in custom key stores, and KMS keys with imported key material.

Key rotation and pricing

AWS KMS charges a monthly fee for first and second rotation of key material maintained for your KMS key. This price increase is capped at the second rotation, and any subsequent rotations will not be billed. For details, see AWS Key Management Service Pricing.

> **Note**
>
> You can use the AWS Cost Explorer Service to view a breakdown of your key storage charges. For example, you can filter your view to see the total charges for keys billed as current and rotated KMS keys by specifying `$REGION-KMS-Keys` for the **Usage Type** and grouping the data by **API Operation**.
> You might still see instances of the legacy `Unknown` API operation for historical dates.

Key rotation and quotas

Each KMS key counts as one key when calculating key resource quotas, regardless of the number of rotated key material versions.

For detailed information about key material and rotation, see AWS Key Management Service Cryptographic Details.

Topics

- How key rotation works
- How to enable and disable automatic key rotation
- How to perform on-demand key rotation
- Rotating keys manually

Why rotate KMS keys?

Cryptographic best practices discourage extensive reuse of keys that encrypt data directly, such as the data keys that AWS KMS generates. When 256-bit data keys encrypt millions of messages they can become exhausted and begin to produce ciphertext with subtle patterns that clever actors can exploit to discover the bits in the key. To avoid this key exhaustion, it's best to use data keys once, or just a few times, which effectively rotates the key material.

However, KMS keys are most often used as *wrapping keys*, also known as *key-encryption keys*. Instead of encrypting data, wrapping keys encrypt the data keys that encrypt your data. As such, they are used far less often than data keys, and are almost never reused enough to risk key exhaustion.

Despite this very low exhaustion risk, you might be required to rotate your KMS keys due to business or contract rules or government regulations. When you are compelled to rotate KMS keys, we recommend that you use automatic key rotation where it is supported, and manual key rotation when automatic key rotation is not supported.

You might consider performing on-demand rotations to demonstrate key material rotation capabilities or to validate automation scripts. We recommend using on-demand rotations for unplanned rotations, and using automatic key rotation with with a custom rotation period whenever possible.

How key rotation works

Key rotation in AWS KMS is a designed to be transparent and easy to use. AWS KMS supports optional automatic and on-demand key rotation only for customer managed keys.

Automatic key rotation

> AWS KMS rotates the KMS key automatically on the next rotation date defined by your rotation period. You don't need to remember or schedule the update.

On-demand rotation

Immediately initiate rotation of the key material associated with your KMS key, regardless of whether or not automatic key rotation is enabled.

Managing key material

AWS KMS retains all key material for a KMS key, even if key rotation is disabled. AWS KMS deletes key material only when you delete the KMS key.

Using key material

When you use a rotated KMS key to encrypt data, AWS KMS uses the current key material. When you use the rotated KMS key to decrypt ciphertext, AWS KMS uses the same version of the key material that was used to encrypt it. You cannot select a particular version of the key material for decrypt operations, AWS KMS automatically chooses the correct version.

Rotation period

Rotation period defines the number of days after you enable automatic key rotation that AWS KMS will rotate your key material, and the number of days between each automatic key rotation thereafter. If you do not specify a value for `RotationPeriodInDays` when you enable automatic key rotation, the default value is 365 days.

You can use the kms:RotationPeriodInDays condition key to further constrain the values that principals can specify in the `RotationPeriodInDays` parameter.

Rotation date

AWS KMS automatically rotates the KMS key on the rotation date defined by your rotation period. The default rotation period is 365 days.

Customer managed keys

Because automatic key rotation is optional on customer managed keys and can be enabled and disabled at any time, the rotation date depends on the date that rotation was most recently enabled. The date can change if you modify the rotation period for a key that you previously enabled automatic key rotation on. The rotation date can change many times over the life of the key.

For example, if you create a customer managed key on January 1, 2022, and enable automatic key rotation with the default rotation period of 365 days on March 15, 2022, AWS KMS rotates the key material on March 15, 2023, March 15, 2024, and every 365 days thereafter.

The following examples assume that automatic key rotation was enabled with the default rotation period of 365 days. These examples demonstrate special cases that might impact a key's rotation period.

- Disable key rotation — If you disable automatic key rotation at any point, the KMS key continues to use the version of the key material it was using when rotation was disabled. If you enable automatic key rotation again, AWS KMS rotates the key material based on the new rotation-enable date.
- Disabled KMS keys — While a KMS key is disabled, AWS KMS does not rotate it. However, the key rotation status does not change, and you cannot change it while the KMS key is disabled. When the KMS key is re-enabled, if the key material is past its last scheduled rotation date , AWS KMS rotates it immediately. If the key material has not missed its last scheduled rotation date, AWS KMS resumes the original key rotation schedule.
- KMS keys pending deletion — While a KMS key is pending deletion, AWS KMS does not rotate it. The key rotation status is set to `false` and you cannot change it while deletion is pending. If deletion is canceled, the previous key rotation status is restored. If the key material is past its last scheduled rotation date, AWS KMS rotates it immediately. If the key material has not missed its last scheduled rotation date, AWS KMS resumes the original key rotation schedule.

AWS managed keys

AWS KMS automatically rotates AWS managed keys every year (approximately 365 days). You cannot enable or disable key rotation for AWS managed keys.

The key material for an AWS managed key is first rotated one year after its creation date, and every year (approximately 365 days from the last rotation) thereafter.

> **Note**
>
> In May 2022, AWS KMS changed the rotation schedule for AWS managed keys from every three years (approximately 1,095 days) to every year (approximately 365 days).
> New AWS managed keys are automatically rotated one year after they are created, and approximately every year thereafter.
> Existing AWS managed keys are automatically rotated one year after their most recent rotation, and every year thereafter.

AWS owned keys

You cannot enable or disable key rotation for AWS owned keys. The key rotation strategy for an AWS owned key is determined by the AWS service that creates and manages the key. For details, see the *Encryption at Rest* topic in the user guide or developer guide for the service.

Supported KMS key types

Automatic key rotation is supported only on symmetric encryption KMS keys with key material that AWS KMS generates (Origin = AWS_KMS).

Automatic key rotation is *not* supported on the following types of KMS keys, but you can rotate these KMS keys manually.

- Asymmetric KMS keys
- HMAC KMS keys
- KMS keys in custom key stores
- KMS keys with imported key material

Multi-Region keys

You can enable and disable automatic key rotation for multi-Region keys. You set the property only on the primary key. When AWS KMS synchronizes the keys, it copies the property setting from the primary key to its replica keys. When the key material of the primary key is rotated, AWS KMS automatically copies that key material to all of its replica keys. For details, see Rotating multi-Region keys.

AWS services

You can enable automatic key rotation on the customer managed keys that you use for server-side encryption in AWS services. The annual rotation is transparent and compatible with AWS services.

Monitoring key rotation

When AWS KMS rotates the key material for an AWS managed key or customer managed key, it writes a `KMS CMK Rotation` event to Amazon EventBridge and a RotateKey event to your AWS CloudTrail log. You can use these records to verify that the KMS key was rotated.

You can use the AWS Key Management Service console to view the number of remaining on-demand rotations and a list of all completed key material rotations for a KMS key.

You can use ListKeyRotations operation to view the details of completed rotations.

Eventual consistency

> Key rotation is subject to the same eventual consistency effects as other AWS KMS management operations. There might be a slight delay before the new key material is available throughout AWS KMS. However, rotating key material does not cause any interruption or delay in cryptographic operations. The current key material is used in cryptographic operations until the new key material is available throughout AWS KMS. When key material for a multi-Region key is automatically rotated, AWS KMS uses the current key material until the new key material is available in all Regions with a related multi-Region key.

How to enable and disable automatic key rotation

By default, when you enable *automatic key rotation* for a KMS key, AWS KMS generates new cryptographic material for the KMS key every year. You can also specify a custom rotation-period to define the number of days after you enable automatic key rotation that AWS KMS will rotate your key material, and the number of days between each automatic rotation thereafter.

Automatic key rotation has the following benefits:

- The properties of the KMS key, including its key ID, key ARN, region, policies, and permissions, do not change when the key is rotated.
- You do not need to change applications or aliases that refer to the key ID or key ARN of the KMS key.
- Rotating key material does not affect the use of the KMS key in any AWS service.
- After you enable key rotation, AWS KMS rotates the KMS key automatically on the next rotation date defined by your rotation period. You don't need to remember or schedule the update.

Authorized users can use the AWS KMS console and the AWS KMS API to enable and disable automatic key rotation and view the key rotation status.

Topics

Enabling and disabling automatic key rotation (console)

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**. (You cannot enable or disable rotation of AWS managed keys. They are automatically rotated every year.)
4. Choose the alias or key ID of a KMS key.
5. Choose the **Key rotation** tab.

   The **Key rotation** tab appears only on the detail page of symmetric encryption KMS keys with key material that AWS KMS generated (the **Origin** is **AWS_KMS**), including multi-Region symmetric encryption KMS keys.

   You cannot automatically rotate asymmetric KMS keys, HMAC KMS keys, KMS keys with imported key material, or KMS keys in custom key stores. However, you can rotate them manually.
6. In the **Automatic key rotation** section, choose **Edit**.
7. For **Key rotation**, select **Enable**.

   > **Note**
   >
   > If a KMS key is disabled or pending deletion, AWS KMS does not rotate the key material and you cannot update the automatic key rotation status or rotation period. Enable the KMS key or cancel deletion to update the automatic key rotation configuration. For details, see How key rotation works and Key states of AWS KMS keys.

8. (Optional) Type a rotation period between 90 and 2560 days. The default value is 365 days. If you do not specify a custom rotation period, AWS KMS will rotate the key material every year.

   You can use the kms:RotationPeriodInDays condition key to limit the values that principals can specify for the rotation period.
9. Choose **Save**.

Enabling and disabling automatic key rotation (AWS KMS API)

You can use the AWS Key Management Service (AWS KMS) API to enable and disable automatic key rotation, and view the current rotation status of any customer managed key. These examples use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

The EnableKeyRotation operation enables automatic key rotation for the specified KMS key. The DisableKeyRotation operation disables it. To identify the KMS key in these operations, use its key ID or key ARN. By default, key rotation is disabled for customer managed keys.

You can use the kms:RotationPeriodInDays condition key to limit the values that principals can specify for the `RotationPeriodInDays` parameter of an `EnableKeyRotation` request.

The following example enables key rotation with a rotation period of 180 days on the specified symmetric encryption KMS key and uses the GetKeyRotationStatus operation to see the result. Then, it disables key rotation and, again, uses **GetKeyRotationStatus** to see the change.

```
$ aws kms enable-key-rotation \
    --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
    --rotation-period-in-days 180

$ aws kms get-key-rotation-status --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
{
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "KeyRotationEnabled": true,
    "RotationPeriodInDays": 180,
    "NextRotationDate": "2024-02-14T18:14:33.587000+00:00"
}

$ aws kms disable-key-rotation --key-id 1234abcd-12ab-34cd-56ef-1234567890ab

$ aws kms get-key-rotation-status --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
{
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "KeyRotationEnabled": false
}
```

How to perform on-demand key rotation

You can perform on-demand rotation of the key material in customer managed KMS keys, regardless of whether or not automatic key rotation is enabled. Disabling automatic rotation (DisableKeyRotation) does not impact your ability to perform on-demand rotations, nor does it cancel any in progress on-demand rotations. On-demand rotations do not change existing automatic rotation schedules. For example, consider a KMS key that has automatic key rotation enabled with a rotation period of 730 days. If the key is scheduled to automatically rotate on April 14, 2024, and you perform an on-demand rotation on April 10, 2024, the key will automatically rotate, as scheduled, on April 14, 2024 and every 730 days thereafter.

You can perform on-demand key rotation a maximum of 10 times per KMS key. You can use the AWS KMS console to view the number of remaining on-demand rotations available for a KMS key.

On-demand key rotation is supported only on symmetric encryption KMS keys. You cannot perform on-demand rotation of asymmetric KMS keys, HMAC KMS keys, KMS keys with imported key material, or KMS keys in a custom key store. To perform on-demand rotation of a set of related multi-Region keys, invoke the on-demand rotation on the primary key.

Authorized users can use the AWS KMS console and the AWS KMS API to initiate on-demand key rotation and view the key rotation status.

Topics

Initiating on-demand key rotation (console)

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**. (You cannot perform on-demand rotation of AWS managed keys. They are automatically rotated every year.)
4. Choose the alias or key ID of a KMS key.
5. Choose the **Key rotation** tab.

The **Key rotation** tab appears only on the detail page of symmetric encryption KMS keys with key material that AWS KMS generated (the **Origin** is **AWS_KMS**), including multi-Region symmetric encryption KMS keys.

You cannot perform on-demand rotation of asymmetric KMS keys, HMAC KMS keys, KMS keys with imported key material, or KMS keys in custom key stores. However, you can rotate them manually.

6. In the **On-demand key rotation** section, choose **Rotate key**.
7. Read and consider the warning and the information about the number of remaining on-demand rotations for the key. If you decide that you do not want to proceed with the on-demand rotation, choose **Cancel**.
8. Choose **Rotate key** to confirm on-demand rotation.

> **Note**
>
> On-demand rotation is subject to the same eventual consistency effects as other AWS KMS management operations. There might be a slight delay before the new key material is available throughout AWS KMS. The banner at the top of the console notifies you when the on-demand rotation is complete.

Initiating on-demand key rotation (AWS KMS API)

You can use the AWS Key Management Service (AWS KMS) API to initiate on-demand key rotation, and view the current rotation status of any customer managed key. These examples use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

The RotateKeyOnDemand operation immediately initiates on-demand key rotation for the specified KMS key. To identify the KMS key in these operations, use its key ID or key ARN.

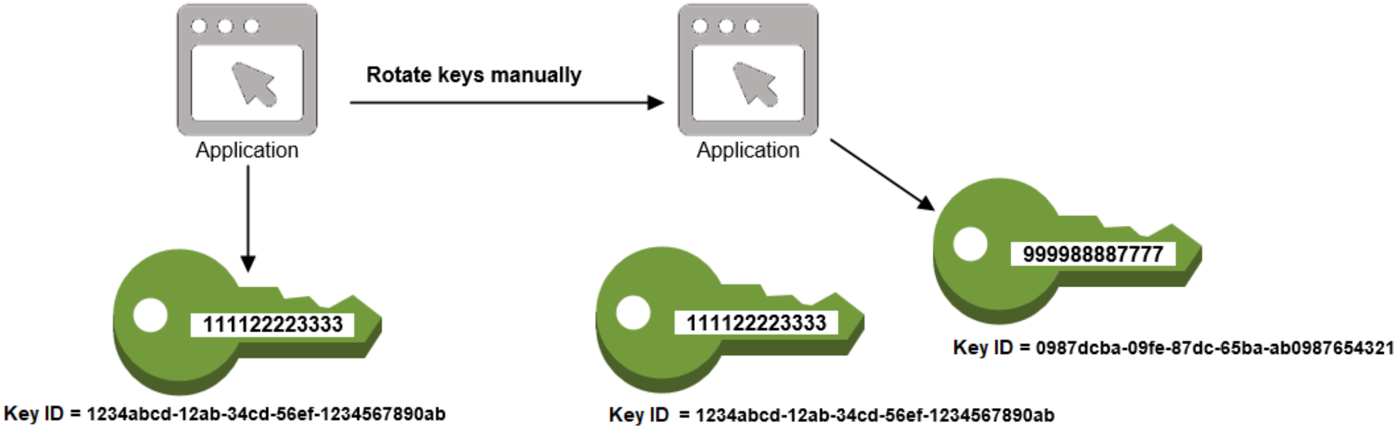
The following example initiates on-demand key rotation on the specified symmetric encryption KMS key and uses the GetKeyRotationStatus operation to verify that the on-demand rotation is in progress. The `OnDemandRotationStartDate` in the `kms:GetKeyRotationStatus` response identifies the date and time that an in progress on-demand rotation was initiated.

```
$ aws kms rotate-key-on-demand --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
{
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
}

$ aws kms get-key-rotation-status --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
{
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "KeyRotationEnabled": true,
    "NextRotationDate": "2024-03-14T18:14:33.587000+00:00",
    "OnDemandRotationStartDate": "2024-02-24T18:44:48.587000+00:00"
    "RotationPeriodInDays": 365
}
```

Rotating keys manually

You might want to create a new KMS key and use it in place of a current KMS key instead of enabling automatic key rotation. When the new KMS key has different cryptographic material than the current KMS key, using the new KMS key has the same effect as changing the key material in an existing KMS key. The process of replacing one KMS key with another is known as *manual key rotation*.


Rotate keys manually
Application
Application
111122223333
Key ID = 1234abcd-12ab-34cd-56ef-1234567890ab
111122223333
Key ID = 1234abcd-12ab-34cd-56ef-1234567890ab
999988887777
Key ID = 0987dcba-09fe-87dc-65ba-ab0987654321

Manual rotation is a good choice when you want to rotate KMS keys that are not eligible for automatic key rotation, such as asymmetric KMS keys, HMAC KMS keys, KMS keys in custom key stores, and KMS keys with imported key material.

Note

When you begin using the new KMS key, be sure to keep the original KMS key enabled so that AWS KMS can decrypt data that the original KMS key encrypted.

When you rotate KMS keys manually, you also need to update references to the KMS key ID or key ARN in your applications. Aliases, which associate a friendly name with a KMS key, can make this process easier. Use an alias to refer to a KMS key in your applications. Then, when you want to change the KMS key that the application uses, instead of editing your application code, change the target KMS key of the alias. For details, see Using aliases in your applications.

> **Note**
>
> Aliases that point to the latest version of a manually rotated KMS key are a good solution for the DescribeKey, Encrypt, GenerateDataKey, GenerateDataKeyPair, GenerateMac, and Sign operations. Aliases are not permitted in operations that manage KMS keys, such as DisableKey or ScheduleKeyDeletion.
> When calling the Decrypt operation on manually rotated symmetric encryption KMS keys, omit the `KeyId` parameter from the command. AWS KMS automatically uses the KMS key that encrypted the ciphertext.
> The `KeyId` parameter is required when calling `Decrypt` or Verify with an asymmetric KMS key, or calling VerifyMac with an HMAC KMS key. These requests fail when the value of the `KeyId` parameter is an alias that no longer points to the KMS key that performed the cryptographic operation, such as when a key is manually rotated. To avoid this error, you must track and specify the correct KMS key for each operation.

To change the target KMS key of an alias, use UpdateAlias operation in the AWS KMS API. For example, this command updates the `alias/TestKey` alias to point to a new KMS key. Because the operation does not return any output, the example uses the ListAliases operation to show that the alias is now associated with a different KMS key and the `LastUpdatedDate` field is updated. The ListAliases commands use the `query` parameter in the AWS CLI to get only the `alias/TestKey` alias.

```
$ aws kms list-aliases --query 'Aliases[?AliasName==`alias/TestKey`]'
{
    "Aliases": [
        {
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/TestKey",
            "AliasName": "alias/TestKey",
            "TargetKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
            "CreationDate": 1521097200.123,
            "LastUpdatedDate": 1521097200.123
        },
    ]
}


$ aws kms update-alias --alias-name alias/TestKey --target-key-id
 0987dcba-09fe-87dc-65ba-ab0987654321

$ aws kms list-aliases --query 'Aliases[?AliasName==`alias/TestKey`]'
{
    "Aliases": [
        {
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/TestKey",
            "AliasName": "alias/TestKey",
            "TargetKeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
            "CreationDate": 1521097200.123,
            "LastUpdatedDate": 1604958290.722
        },
    ]
}
```

Monitoring AWS KMS keys

Monitoring is an important part of understanding the availability, state, and usage of your AWS KMS keys in AWS KMS and maintaining the reliability, availability, and performance of your AWS solutions. Collecting monitoring data from all the parts of your AWS solution will help you debug a multipoint failure if one occurs. Before you start monitoring your KMS keys, however, create a monitoring plan that includes answers to the following questions:

- What are your monitoring goals?
- What resources will you monitor?
- How often will you monitor these resources?
- What monitoring tools will you use?
- Who will perform the monitoring tasks?
- Who should be notified when something happens?

The next step is to monitor your KMS keys over time to establish a baseline for normal AWS KMS usage and expectations in your environment. As you monitor your KMS keys, store historical monitoring data so that you can compare it with current data, identify normal patterns and anomalies, and devise methods to address issues.

For example, you can monitor AWS KMS API activity and events that affect your KMS keys. When data falls above or below your established norms, you might need to investigate or take corrective action.

To establish a baseline for normal patterns, monitor the following items:

- AWS KMS API activity for *data plane* operations. These are cryptographic operations that use a KMS key, such as Decrypt, Encrypt, ReEncrypt, and GenerateDataKey.
- AWS KMS API activity for *control plane* operations that are important to you. These operations manage a KMS key, and you might want to monitor those that change a KMS key's availability (such as ScheduleKeyDeletion, CancelKeyDeletion, DisableKey, EnableKey, ImportKeyMaterial, and DeleteImportedKeyMaterial) or change a KMS key's access control (such as PutKeyPolicy and RevokeGrant).
- Other AWS KMS metrics (such as the amount of time remaining until your imported key material expires) and events (such as the expiration of imported key material or the deletion or key rotation of a KMS key).

Monitoring tools

AWS provides various tools that you can use to monitor your KMS keys. You can configure some of these tools to do the monitoring for you, while some of the tools require manual intervention. We recommend that you automate monitoring tasks as much as possible.

Automated monitoring tools

You can use the following automated monitoring tools to watch your KMS keys and report when something has changed.

- **AWS CloudTrail Log Monitoring** – Share log files between accounts, monitor CloudTrail log files in real time by sending them to CloudWatch Logs, write log processing applications with the CloudTrail Processing Library, and validate that your log files have not changed after delivery by CloudTrail. For more information, see Working with CloudTrail Log Files in the *AWS CloudTrail User Guide*.
- **Amazon CloudWatch Alarms** – Watch a single metric over a time period that you specify, and perform one or more actions based on the value of the metric relative to a given threshold over

a number of time periods. The action is a notification sent to an Amazon Simple Notification Service (Amazon SNS) topic or Amazon EC2 Auto Scaling policy. CloudWatch alarms do not invoke actions simply because they are in a particular state; the state must have changed and been maintained for a specified number of periods. For more information, see Monitoring with Amazon CloudWatch.

- **Amazon EventBridge** – Match events and route them to one or more target functions or streams to capture state information and, if necessary, make changes or take corrective action. For more information, see Monitoring with Amazon EventBridge and the Amazon EventBridge User Guide.
- **Amazon CloudWatch Logs** – Monitor, store, and access your log files from AWS CloudTrail or other sources. For more information, see the Amazon CloudWatch Logs User Guide.

Manual monitoring tools

Another important part of monitoring KMS keys involves manually monitoring those items that the CloudWatch alarms and events don't cover. The AWS KMS, CloudWatch, AWS Trusted Advisor, and other AWS dashboards provide an at-a-glance view of the state of your AWS environment.

You can customize the **AWS managed keys** and **Customer managed keys** pages of the AWS KMS console to display the following information about each KMS key:

- Key ID
- Status
- Creation date
- Expiration date (for KMS keys with imported key material)
- Origin
- Custom key store ID (for KMS keys in custom key stores)

The CloudWatch console dashboard shows the following:

- Current alarms and status
- Graphs of alarms and resources
- Service health status

In addition, you can use CloudWatch to do the following:

- Create customized dashboards to monitor the services you care about
- Graph metric data to troubleshoot issues and discover trends
- Search and browse all your AWS resource metrics
- Create and edit alarms to be notified of problems

AWS Trusted Advisor can help you monitor your AWS resources to improve performance, reliability, security, and cost effectiveness. Four Trusted Advisor checks are available to all users; more than 50 checks are available to users with a Business or Enterprise support plan. For more information, see AWS Trusted Advisor.

Logging AWS KMS API calls with AWS CloudTrail

AWS KMS is integrated with AWS CloudTrail, a service that records all calls to AWS KMS by users, roles, and other AWS services. CloudTrail captures all API calls to AWS KMS as events, including calls from the AWS KMS console, AWS KMS APIs, AWS CloudFormation templates, the AWS Command Line Interface (AWS CLI), and AWS Tools for PowerShell.

CloudTrail logs all AWS KMS operations, including read-only operations, such as ListAliases and GetKeyRotationStatus, operations that manage KMS keys, such as CreateKey and PutKeyPolicy, and cryptographic operations, such as GenerateDataKey and Decrypt. It also logs internal operations that AWS KMS calls for you, such as DeleteExpiredKeyMaterial, DeleteKey, SynchronizeMultiRegionKey, and RotateKey.

CloudTrail logs successful operations and attempted calls that failed, such as when the caller is denied access to a resource. Cross-account operations on KMS keys are logged in both the caller account and the KMS key owner account. However, cross-account AWS KMS requests that are rejected because access is denied are logged only in the caller's account.

For security reasons, some fields are omitted from AWS KMS log entries, such as the `Plaintext` parameter of an Encrypt request, and the response to GetKeyPolicy or any cryptographic operation. To make it easier to search for CloudTrail log entries for particular KMS keys, AWS KMS adds the key ARN of the affected KMS key to the `responseElements` field in the log entries for some AWS KMS key management operations, even when the API operation doesn't return the key ARN.

Although by default, all AWS KMS actions are logged as CloudTrail events, you can exclude AWS KMS actions from a CloudTrail trail. For details, see Excluding AWS KMS events from a trail.

Learn more:

- For CloudTrail log examples of AWS KMS operations for an AWS Nitro enclave, see Monitoring requests for Nitro enclaves.

Topics

Logging events in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS KMS, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for AWS KMS, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple Regions and Receiving CloudTrail log files from multiple accounts

To learn more about CloudTrail, see the AWS CloudTrail User Guide. To learn about other ways to monitor the use of your KMS keys, see Monitoring AWS KMS keys.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- If the request was made with root credentials or the credentials of an IAM user.
- If the request was made with temporary security credentials for a role or federated user.
- If the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Searching events in CloudTrail

To search CloudTrail log entries, use the CloudTrail console or the CloudTrail LookupEvents operation. CloudTrail supports numerous attribute values for filtering your search, including event name, user name, and event source.

To help you search for AWS KMS log entries in CloudTrail, AWS KMS populates the following CloudTrail log entry fields.

> **Note**
>
> Beginning in December 2022, AWS KMS populates the **Resource type** and **Resource name** attributes in all management operations that change a particular KMS key. These attribute values might be null in older CloudTrail entries for the following operations: CreateAlias, CreateGrant, DeleteAlias, DeleteImportedKeyMaterial, ImportKeyMaterial, ReplicateKey, RetireGrant, RevokeGrant, UpdateAlias, and UpdatePrimaryRegion.

Attribute	Value	Log entries
Event source (`EventSource`)	`kms.amazonaws.com`	All operations.
Resource type (`ResourceType`)	`AWS::KMS::Key`	Management operations that change a particular KMS key, such as `CreateKey` and `EnableKey`, but not `ListKeys`.
Resource name (`ResourceName`)	Key ARN (or key ID and key ARN)	Management operations that change a particular KMS

Attribute	Value	Log entries
		key, such as `CreateKey` and `EnableKey` , but not `ListKeys`.

To help you find log entries for management operations on particular KMS keys, AWS KMS records the key ARN of the affected KMS key in the `responseElements.keyId` element of the log entry, even when the AWS KMS API operation doesn't return the key ARN.

For example, a successful call to the DisableKey operation doesn't return any values in the response, but instead of a null value, the `responseElements.keyId` value in the DisableKey log entry includes the key ARN of the disabled KMS key.

This feature was added in December 2022 and affects the following CloudTrail log entries: CreateAlias, CreateGrant, DeleteAlias, DeleteKey, DisableKey, EnableKey, EnableKeyRotation, ImportKeyMaterial, RotateKey, SynchronizeMultiRegionKey, TagResource, UntagResource, UpdateAlias, and UpdatePrimaryRegion.

Excluding AWS KMS events from a trail

To provide a record of the use and management of their AWS KMS resources, most AWS KMS users rely on the events in a CloudTrail trail. The trail can be an valuable source of data for auditing critical events, such as creating, disabling, and deleting AWS KMS keys, changing key policy, and the use of your KMS keys by AWS services on your behalf. In some cases, the metadata in a CloudTrail log entry, such as the encryption context in an encryption operation, can help you to avoid or resolve errors.

However, because AWS KMS can generate a large number of events, AWS CloudTrail lets you exclude AWS KMS events from a trail. This per-trail setting excludes all AWS KMS events; you cannot exclude particular AWS KMS events.

⚠ **Warning**

Excluding AWS KMS events from a CloudTrail Log can obscure actions that use your KMS keys. Be cautious when giving principals the `cloudtrail:PutEventSelectors` permission that is required to perform this operation.

To exclude AWS KMS events from a trail:

- In the CloudTrail console, use the **Log Key Management Service events** setting when you create a trail or update a trail. For instructions, see Logging Management Events with the AWS Management Console in the AWS CloudTrail User Guide.
- In the CloudTrail API, use the PutEventSelectors operation. Add the `ExcludeManagementEventSources` attribute to your event selectors with a value of `kms.amazonaws.com`. For an example, see Example: A trail that does not log AWS Key Management Service events in the AWS CloudTrail User Guide.

You can disable this exclusion at any time by changing the console setting or the event selectors for a trail. The trail will then start recording AWS KMS events. However, it cannot recover AWS KMS events that occurred while the exclusion was effective.

When you exclude AWS KMS events by using the console or API, the resulting CloudTrail `PutEventSelectors` API operation is also logged in your CloudTrail Logs. If AWS KMS events don't appear in your CloudTrail Logs, look for a `PutEventSelectors` event with the `ExcludeManagementEventSources` attribute set to `kms.amazonaws.com`.

Examples of AWS KMS log entries

AWS KMS writes entries to your CloudTrail log when you call an AWS KMS operation and when an AWS service calls an operation on your behalf. AWS KMS also writes an entry when it calls an operation for you. For example, it writes an entry when it deletes a KMS key that you scheduled for deletion.

The following topics display examples of CloudTrail log entries for AWS KMS operations.

For examples of CloudTrail log entries of requests to AWS KMS from AWS Nitro Enclaves, see Monitoring requests for Nitro enclaves.

Topics

- CreateKey
- Decrypt
- DeleteAlias
- DeleteCustomKeyStore
- DeleteExpiredKeyMaterial
- DeleteImportedKeyMaterial
- DeleteKey
- DescribeCustomKeyStores
- DescribeKey
- DisableKey
- DisableKeyRotation
- DisconnectCustomKeyStore
- EnableKey
- EnableKeyRotation
- Encrypt
- GenerateDataKey
- GenerateDataKeyPair
- GenerateDataKeyPairWithoutPlaintext
- GenerateDataKeyWithoutPlaintext
- GenerateMac
- GenerateRandom
- GetKeyPolicy
- GetKeyRotationStatus
- GetParametersForImport
- ImportKeyMaterial
- ListAliases
- ListGrants
- ListKeyRotations
- PutKeyPolicy
- ReEncrypt

- ReplicateKey
- RetireGrant
- RevokeGrant
- RotateKey
- RotateKeyOnDemand
- ScheduleKeyDeletion
- Sign
- SynchronizeMultiRegionKey
- TagResource
- UntagResource
- UpdateAlias
- UpdateCustomKeyStore
- UpdateKeyDescription
- UpdatePrimaryRegion
- VerifyMac
- Verify
- Amazon EC2 example one
- Amazon EC2 example two

CancelKeyDeletion

The following example shows an AWS CloudTrail log entry generated by calling the CancelKeyDeletion operation. For information about deleting AWS KMS keys, see Deleting AWS KMS keys.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2020-07-27T21:53:17Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CancelKeyDeletion",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "requestID": "e3452e68-d4b0-4ec7-a768-7ae96c23764f",
    "eventID": "d818bf03-6655-48e9-8b26-f279a07075fd",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

ConnectCustomKeyStore

The following example shows an AWS CloudTrail log entry generated by calling the `ConnectCustomKeyStore` operation. For information about connecting a custom key store, see Connecting and disconnecting an AWS CloudHSM key store.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2021-10-21T20:17:32Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "ConnectCustomKeyStore",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "customKeyStoreId": "cks-1234567890abcdef0"
    },
    "responseElements": null,
    "additionalEventData": {
        "customKeyStoreName": "ExampleKeyStore",
        "clusterId": "cluster-1a23b4cdefg"
    },
    "requestID": "abcde9e1-f1a3-4460-a423-577fb6e695c9",
    "eventID": "114b61b9-0ea6-47f5-a9d2-4f2bdd0017d5",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333"
}
```

CreateAlias

The following example shows an AWS CloudTrail log entry for the CreateAlias operation. The `resources` element includes fields for the alias and KMS key resources. For information about creating aliases in AWS KMS, see Creating an alias.

CloudTrail log entries for this operation recorded on or after December 2022 include the key ARN of the affected KMS key in the `responseElements.keyId` value, even though this operation does not return the key ARN.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2022-08-14T23:08:31Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateAlias",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "aliasName": "alias/ExampleAlias",
        "targetKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": {
        "keyId":"arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "requestID": "caec1e0c-ce03-419e-bdab-6ab1f7c57c01",
    "eventID": "2dd6e784-8286-46a6-befd-d64e5a02fb28",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        },
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:111122223333:alias/ExampleAlias"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

CreateCustomKeyStore

The following example shows an AWS CloudTrail log entry generated by calling the `CreateCustomKeyStore` operation on an AWS CloudHSM key store. For information about creating custom key stores, see Creating an AWS CloudHSM key store.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2021-10-21T20:17:32Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateCustomKeyStore",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "customKeyStoreName": "ExampleKeyStore",
        "clusterId": "cluster-1a23b4cdefg"
    },
    "responseElements": {
        "customKeyStoreId": "cks-1234567890abcdef0"
    },
    "requestID": "abcde9e1-f1a3-4460-a423-577fb6e695c9",
    "eventID": "114b61b9-0ea6-47f5-a9d2-4f2bdd0017d5",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333"
}
```

CreateGrant

The following example shows an AWS CloudTrail log entry for the CreateGrant operation. For information about creating grants in AWS KMS, see Grants in AWS KMS.

CloudTrail log entries for this operation recorded on or after December 2022 include the key ARN of the affected KMS key in the `responseElements.keyId` value, even though this operation does not return the key ARN.

```
{
  "eventVersion": "1.02",
  "userIdentity": {
      "type": "IAMUser",
      "principalId": "EX_PRINCIPAL_ID",
      "arn": "arn:aws:iam::111122223333:user/Alice",
      "accountId": "111122223333",
      "accessKeyId": "EXAMPLE_KEY_ID",
      "userName": "Alice"
  },
  "eventTime": "2014-11-04T00:53:12Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "CreateGrant",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "AWS Internal",
  "requestParameters": {
      "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "constraints": {
          "encryptionContextSubset": {
              "ContextKey1": "Value1"
          }
      },
      "operations": ["Encrypt",
      "RetireGrant"],
      "granteePrincipal": "EX_PRINCIPAL_ID"
  },
  "responseElements": {
      "grantId": "f020fe75197b93991dc8491d6f19dd3cebb24ee62277a05914386724f3d48758",
      "keyId":"arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  },
  "requestID": "f3c08808-63bc-11e4-bc2b-4198b6150d5c",
  "eventID": "5d529779-2d27-42b5-92da-91aaea1fc4b5",
  "readOnly": false,
  "resources": [{
      "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "accountId": "111122223333"
  }],
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

CreateKey

These examples show AWS CloudTrail log entries for the CreateKey operation.

A `CreateKey` log entry can result from a `CreateKey` request or the `CreateKey` operation for a ReplicateKey request.

The following example shows an CloudTrail log entry for a CreateKey operation that creates a symmetric encryption KMS key. For information about creating KMS keys, see Creating keys.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2022-08-10T22:38:27Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "description": "",
        "origin": "EXTERNAL",
        "bypassPolicyLockoutSafetyCheck": false,
        "customerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "keySpec": "SYMMETRIC_DEFAULT",
        "keyUsage": "ENCRYPT_DECRYPT"
    },
    "responseElements": {
        "keyMetadata": {
            "AWSAccountId": "111122223333",
            "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
            "arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "creationDate": "Aug 10, 2022, 10:38:27 PM",
            "enabled": false,
            "description": "",
            "keyUsage": "ENCRYPT_DECRYPT",
            "keyState": "PendingImport",
            "origin": "EXTERNAL",
            "keyManager": "CUSTOMER",
            "customerMasterKeySpec": "SYMMETRIC_DEFAULT",
            "keySpec": "SYMMETRIC_DEFAULT",
            "encryptionAlgorithms": [
                "SYMMETRIC_DEFAULT"
            ],
            "multiRegion": false
        }
    },
    "requestID": "1aef6713-0223-4ff7-9a6d-781360521930",
    "eventID": "36327b37-f4f6-40a9-92ab-48064ec905a2",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

The following example shows the CloudTrail log of a `CreateKey` operation that creates a symmetric encryption KMS key in an AWS CloudHSM key store.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::111122223333:user/Alice",
            "accountId": "111122223333",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
    },
    "eventTime": "2021-10-14T17:39:50Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyUsage": "ENCRYPT_DECRYPT",
        "bypassPolicyLockoutSafetyCheck": false,
        "origin": "AWS_CLOUDHSM",
        "keySpec": "SYMMETRIC_DEFAULT",
        "customerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "customKeyStoreId": "cks-1234567890abcdef0",
        "description": ""
    },
    "responseElements": {
        "keyMetadata": {
            "aWSAccountId": "111122223333",
            "keyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
            "arn": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321",
            "creationDate": "Oct 14, 2021, 5:39:50 PM",
            "enabled": true,
            "description": "",
            "keyUsage": "ENCRYPT_DECRYPT",
            "keyState": "Enabled",
            "origin": "AWS_CLOUDHSM",
            "customKeyStoreId": "cks-1234567890abcdef0",
            "cloudHsmClusterId": "cluster-1a23b4cdefg",
            "keyManager": "CUSTOMER",
            "customerMasterKeySpec": "SYMMETRIC_DEFAULT",
            "keySpec": "SYMMETRIC_DEFAULT",
            "encryptionAlgorithms": [
                "SYMMETRIC_DEFAULT"
            ],
            "multiRegion": false
        }
    },
    "additionalEventData": {
        "backingKey": "{\"keyHandle\":\"19\",\"backingKeyId\":\"backing-key-id\"}"
    },
    "requestID": "4f0b185c-588c-4767-9e90-c618f7e13cad",
    "eventID": "c73964b8-703d-49e4-bd9e-f773d0ee1e65",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

The following example shows the CloudTrail log of a `CreateKey` operation that creates a symmetric encryption KMS key in an external key store.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::111122223333:user/Alice",
            "accountId": "111122223333",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
    },
    "eventTime": "2022-09-07T22:37:45Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "tags": [],
        "keyUsage": "ENCRYPT_DECRYPT",
        "description": "",
        "origin": "EXTERNAL_KEY_STORE",
        "multiRegion": false,
        "keySpec": "SYMMETRIC_DEFAULT",
        "customerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "bypassPolicyLockoutSafetyCheck": false,
        "customKeyStoreId": "cks-1234567890abcdef0",
        "xksKeyId": "bb8562717f809024"
    },
    "responseElements": {
        "keyMetadata": {
            "aWSAccountId": "111122223333",
            "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
            "arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "creationDate": "Dec 7, 2022, 10:37:45 PM",
            "enabled": true,
            "description": "",
            "keyUsage": "ENCRYPT_DECRYPT",
            "keyState": "Enabled",
            "origin": "EXTERNAL_KEY_STORE",
            "customKeyStoreId": "cks-1234567890abcdef0",
            "keyManager": "CUSTOMER",
            "customerMasterKeySpec": "SYMMETRIC_DEFAULT",
            "keySpec": "SYMMETRIC_DEFAULT",
            "encryptionAlgorithms": [
                "SYMMETRIC_DEFAULT"
            ],
            "multiRegion": false,
            "xksKeyConfiguration": {
                "id": "bb8562717f809024"
            }
        }
    },
    "requestID": "ba197c82-3ac7-487a-8ff4-7736bbeb1316",
    "eventID": "838ad5f4-5fdd-4044-afd7-4dbd88c6af56",
    "readOnly": false,
    "resources": [
        {
            "accountId": "227179770375",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-east-1:227179770375:key/39c5eb22-
f37c-4956-92ca-89e8f8b57ab2"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

Decrypt

These examples show AWS CloudTrail log entries for the Decrypt operation.

The CloudTrail log entry for a `Decrypt` operation always includes the `encryptionAlgorithm` in the `requestParameters` even if the encryption algorithm wasn't specified in the request. The ciphertext in the request and the plaintext in the response are omitted.

Topics

- Decrypt with a standard symmetric encryption key
- Decrypt failure with a standard symmetric encryption key
- Decrypt with a KMS key in an AWS CloudHSM key store
- Decrypt with a KMS key in an external key store
- Decrypt failure with a KMS key in an external key store

Decrypt with a standard symmetric encryption key

The following is an example CloudTrail log entry for a `Decrypt` operation with a standard symmetric encryption key.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2020-07-27T22:58:24Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "encryptionContext": {
            "Department": "Engineering",
            "Project": "Alpha"
        }
    },
    "responseElements": null,
    "requestID": "12345126-30d5-4b28-98b9-9153da559963",
    "eventID": "abcde202-ba1a-467c-b4ba-f729d45ae521",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

Decrypt failure with a standard symmetric encryption key

The following example CloudTrail log entry records a failed `Decrypt` operation with a standard symmetric encryption KMS key. The exception (`errorCode`) and error message (`errorMessage`) are included help you to resolve the error.

In this case, the symmetric encryption KMS key specified in the `Decrypt` request was not the symmetric encryption KMS key that was used to encrypt the data.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2022-11-24T18:57:43Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "errorCode": "IncorrectKeyException"
    "errorMessage": "The key ID in the request does not identify a CMK that can perform
 this operation.",
    "requestParameters": {
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "encryptionContext": {
            "Department": "Engineering",
            "Project": "Alpha"
        }
    },
    "responseElements": null,
    "requestID": "22345126-30d5-4b28-98b9-9153da559963",
    "eventID": "abcde202-ba1a-467c-b4ba-f729d45ae521",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

Decrypt with a KMS key in an AWS CloudHSM key store

The following example CloudTrail log entry records a `Decrypt` operation with a KMS key in an AWS CloudHSM key store. All log entries for cryptographic operations with a KMS key in a custom key store include an `additionalEventData` field with the `customKeyStoreId`. The `additionalEventData` isn't specified in the request.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2021-10-26T23:41:27Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "requestParameters": {
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "encryptionContext": {
            "Department": "Development",
            "Purpose": "Test"
        }
    },
    "responseElements": null,
    "additionalEventData": {
        "customKeyStoreId": "cks-1234567890abcdef0"
    },
    "requestID": "e1b881f8-2048-41f8-b6cc-382b7857ec61",
    "eventID": "a79603d5-4cde-46fc-819c-a7cf547b9df4",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

Decrypt with a KMS key in an external key store

The following example CloudTrail log entry records a `Decrypt` operation with a KMS key in an external key store. In addition to the `customKeyStoreId`, the `additionalEventData` field includes the external key ID (`XksKeyId`). The `additionalEventData` isn't specified in the request.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2022-11-24T00:26:58Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "AWS Internal",
    "requestParameters": {
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "keyId": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321",
        "encryptionContext": {
            "Department": "Engineering",
            "Purpose": "Test"
        }
    },
    "responseElements": null,
    "additionalEventData": {
        "customKeyStoreId": "cks-9876543210fedcba9",
        "xksKeyId": "abc01234567890fe"
    },
    "requestID": "f1b881f8-2048-41f8-b6cc-382b7857ec61",
    "eventID": "b79603d5-4cde-46fc-819c-a7cf547b9df4",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

Decrypt failure with a KMS key in an external key store

The following example CloudTrail log entry records a failed request for a `Decrypt` operation with a KMS key in an external key store. CloudWatch logs requests that fail, in addition to successful requests. When recording a failure, the CloudTrail log entry includes the exception (errorCode) and the accompanying error message (errorMessage).

If the failed request reached your external key store proxy, as in this example, you can use the `requestId` value to associate the failed request with a corresponding request your external key store proxy logs, if your proxy provides them.

For help with `Decrypt` requests in external key stores, see Decryption errors.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2022-11-24T00:26:58Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "errorCode": "KMSInvalidStateException",
    "errorMessage": "The external key store proxy rejected the request because the
 specified ciphertext or additional authenticated data is corrupted, missing, or
 otherwise invalid.",
    "requestParameters": {
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "keyId": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321",
        "encryptionContext": {
            "Department": "Engineering",
            "Purpose": "Test"
        }
    },
    "responseElements": null,
    "additionalEventData": {
        "customKeyStoreId": "cks-9876543210fedcba9",
        "xksKeyId": "abc01234567890fe"
    },
    "requestID": "f1b881f8-2048-41f8-b6cc-382b7857ec61",
    "eventID": "b79603d5-4cde-46fc-819c-a7cf547b9df4",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

DeleteAlias

The following example shows an AWS CloudTrail log entry for the DeleteAlias operation. For information about deleting aliases, see Deleting an alias.

CloudTrail log entries for this operation recorded on or after December 2022 include the key ARN of the affected KMS key in the `responseElements.keyId` value, even though this operation does not return the key ARN.

```
{
    "eventVersion": "1.02",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2014-11-04T00:52:27Z"
            }
        }
    },
    "eventTime": "2014-11-04T00:52:27Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DeleteAlias",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "aliasName": "alias/my_alias"
    },
    "responseElements": {
        "keyId":"arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "requestID": "d9542792-63bc-11e4-bc2b-4198b6150d5c",
    "eventID": "12f48554-bb04-4991-9cfc-e7e85f68eda0",
    "readOnly": false,
    "resources": [{
        "ARN": "arn:aws:kms:us-east-1:111122223333:alias/my_alias",
        "accountId": "111122223333"
    },
    {
        "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "accountId": "111122223333"
    }],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

DeleteCustomKeyStore

The following example shows an AWS CloudTrail log entry generated by calling the `DeleteCustomKeyStore` operation. For information about creating custom key stores, see Deleting an AWS CloudHSM key store.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2021-10-21T20:17:32Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DeleteCustomKeyStore",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "customKeyStoreId": "cks-1234567890abcdef0"
    },
    "responseElements": null,
    "additionalEventData": {
        "customKeyStoreName": "ExampleKeyStore",
        "clusterId": "cluster-1a23b4cdefg"
    },
    "requestID": "abcde9e1-f1a3-4460-a423-577fb6e695c9",
    "eventID": "114b61b9-0ea6-47f5-a9d2-4f2bdd0017d5",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333"
}
```

DeleteExpiredKeyMaterial

When you import key material into an AWS KMS key (KMS key), you can set an expiration date and time for that key material. AWS KMS records an entry in your CloudTrail log when you import the key material (with the expiration settings) and when AWS KMS deletes the expired key material. For information about creating KMS key with imported key material, see Importing key material for AWS KMS keys.

The following example shows an AWS CloudTrail log entry generated when AWS KMS deletes the expired key material.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "accountId": "111122223333",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2021-01-01T16:00:00Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DeleteExpiredKeyMaterial",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": null,
    "responseElements": null,
    "eventID": "cfa932fd-0d3a-4a76-a8b8-616863a2b547",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsServiceEvent",
    "recipientAccountId": "111122223333",
    "serviceEventDetails": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    }
}
```

DeleteImportedKeyMaterial

If you import key material into a KMS key, you can delete the imported key material at any time by using the DeleteImportedKeyMaterial operation. When you delete imported key material from a KMS key, the key state of the KMS key changes to `PendingImport` and the KMS key cannot be used in any cryptographic operations. For details, see Deleting imported key material.

The following example shows an AWS CloudTrail log entry generated for the `DeleteImportedKeyMaterial` operation.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2022-10-04T21:43:33Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DeleteImportedKeyMaterial",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": {
        "keyId":"&example-key-arn-1;"
    },
    "requestID": "dcf0e82f-dad0-4622-a378-a5b964ad42c1",
    "eventID": "2afbb991-c668-4641-8a00-67d62e1fecbd",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

DeleteKey

These examples show the AWS CloudTrail log entry that is generated when a KMS key is deleted. To delete a KMS key, you use the ScheduleKeyDeletion operation. After the specified waiting period expires, AWS KMS deletes the KMS key and records an entry like the following one in your CloudTrail log to record that event.

CloudTrail log entries for this operation recorded on or after December 2022 include the key ARN of the affected KMS key in the `responseElements.keyId` value, even though this operation does not return the key ARN.

For an example of the CloudTrail log entry for the `ScheduleKeyDeletion` operation, see ScheduleKeyDeletion. For information about deleting KMS keys, see Deleting AWS KMS keys.

The following example CloudTrail log entry records a `DeleteKey` operation of a KMS key with key material in AWS KMS.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "accountId": "111122223333",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2020-07-31T00:07:00Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DeleteKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": null,
    "responseElements": null,
    "eventID": "b25f9cda-74e1-4458-847b-4972a0bf9668",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsServiceEvent",
    "recipientAccountId": "111122223333",
     "managementEvent": true,
    "eventCategory": "Management"
}
```

The following CloudTrail log entry records a `DeleteKey` operation of a KMS key in an AWS CloudHSM custom key store.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "accountId": "111122223333",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2021-10-26T23:41:27Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DeleteKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": null,
    "responseElements": {
        "keyId":"arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "additionalEventData": {
        "customKeyStoreId": "cks-1234567890abcdef0",
        "clusterId": "cluster-1a23b4cdefg",
        "backingKeys": "[{\"keyHandle\":\"01\",\"backingKeyId\":\"backing-key-id\"}]",
        "backingKeysDeletionStatus": "[{\"keyHandle\":\"01\",\"backingKeyId\":
\"backing-key-id\",\"deletionStatus\":\"SUCCESS\"}]"
    },
    "eventID": "1234585c-4b0c-4340-ab11-662414b79239",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsServiceEvent",
    "recipientAccountId": "111122223333",
    "managementEvent": true,
    "eventCategory": "Management"
}
```

DescribeCustomKeyStores

The following example shows an AWS CloudTrail log entry generated by calling the `DescribeCustomKeyStores` operation. For information about viewing custom key stores, see Viewing an AWS CloudHSM key store.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2021-10-21T20:17:32Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeCustomKeyStores",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "customKeyStoreId": "cks-1234567890abcdef0"
    },
    "responseElements": null,
    "requestID": "abcde9e1-f1a3-4460-a423-577fb6e695c9",
    "eventID": "2ea1735f-628d-43e3-b2ee-486d02913a78",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333"
}
```

DescribeKey

The following example shows an AWS CloudTrail log entry for the DescribeKey operation. AWS KMS records an entry like the following one when you call the `DescribeKey` operation or view KMS keys in the AWS KMS console. This call is the result of viewing a key in the AWS KMS management console.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2022-09-26T18:01:36Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "requestID": "12345126-30d5-4b28-98b9-9153da559963",
    "eventID": "abcde202-ba1a-467c-b4ba-f729d45ae521",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

DisableKey

The following example shows an AWS CloudTrail log entry for the DisableKey operation. For information about enabling and disabling AWS KMS keys in AWS KMS, see Enabling and disabling keys.

CloudTrail log entries for this operation recorded on or after December 2022 include the key ARN of the affected KMS key in the `responseElements.keyId` value, even though this operation does not return the key ARN.

```
{
    "eventVersion": "1.02",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-11-04T00:52:43Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DisableKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": {
        "keyId":"arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "requestID": "12345126-30d5-4b28-98b9-9153da559963",
    "eventID": "abcde202-ba1a-467c-b4ba-f729d45ae521",
    "readOnly": false,
    "resources": [{
        "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "accountId": "111122223333"
    }],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

DisableKeyRotation

The following example shows an AWS CloudTrail log entry generated by calling the DisableKeyRotation operation. For information about automatic key rotation, see Rotating AWS KMS keys.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2022-09-01T19:31:39Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DisableKeyRotation",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "requestID": "d6a9351a-ed6e-4581-88d1-2a9a8a538497",
    "eventID": "6313164c-83aa-4cc3-9e1a-b7c426f7a5b1",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

DisconnectCustomKeyStore

The following example shows an AWS CloudTrail log entry generated by calling the `DisconnectCustomKeyStore` operation. For information about disconnecting a custom key store, see Connecting and disconnecting an AWS CloudHSM key store.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2021-10-21T20:17:32Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DisconnectCustomKeyStore",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "customKeyStoreId": "cks-1234567890abcdef0"
    },
    "responseElements": null,
    "additionalEventData": {
        "customKeyStoreName": "ExampleKeyStore",
        "clusterId": "cluster-1a23b4cdefg"
    },
    "requestID": "abcde9e1-f1a3-4460-a423-577fb6e695c9",
    "eventID": "114b61b9-0ea6-47f5-a9d2-4f2bdd0017d5",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333"
}
```

EnableKey

The following example shows an AWS CloudTrail log entry for the EnableKey operation. For information about enabling and disabling AWS KMS keys in AWS KMS, see Enabling and disabling keys..

CloudTrail log entries for this operation recorded on or after December 2022 include the key ARN of the affected KMS key in the `responseElements.keyId` value, even though this operation does not return the key ARN.

```
{
    "eventVersion": "1.02",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-11-04T00:52:20Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "EnableKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": {
        "keyId":"arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "requestID": "d528a6fb-63bc-11e4-bc2b-4198b6150d5c",
    "eventID": "be393928-3629-4370-9634-567f9274d52e",
    "readOnly": false,
    "resources": [{
        "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "accountId": "111122223333"
    }],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

EnableKeyRotation

The following example shows an AWS CloudTrail log entry of a call to the EnableKeyRotation operation. For an example of the CloudTrail log entry that is written when the key is rotated, see RotateKey. For information about rotating AWS KMS keys, see Rotating AWS KMS keys.

Note

The rotation-period is an optional request parameter. If you do not specify a rotation period when you enable automatic key rotation, the default value is 365 days.

CloudTrail log entries for this operation recorded on or after December 2022 include the key ARN of the affected KMS key in the `responseElements.keyId` value, even though this operation does not return the key ARN.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2020-07-25T23:41:56Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "EnableKeyRotation",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "rotationPeriodInDays": 180
    },
    "responseElements": {
        "keyId":"arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "requestID": "81f5b794-452b-4d6a-932b-68c188165273",
    "eventID": "fefc43a7-8e06-419f-bcab-b3bf18d6a401",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

Encrypt

The following example shows an AWS CloudTrail log entry for the Encrypt operation.

```
{
    "eventVersion": "1.02",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2022-07-14T20:17:42Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Encrypt",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "encryptionContext": {
            "Department": "Engineering"
        },
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
    },
    "responseElements": null,
    "requestID": "f3423043-63bc-11e4-bc2b-4198b6150d5c",
    "eventID": "91235988-eb87-476a-ac2c-0cdc244e6dca",
    "readOnly": true,
    "resources": [{
        "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "accountId": "111122223333"
    }],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

GenerateDataKey

The following example shows an AWS CloudTrail log entry for the GenerateDataKey operation.

```
{
    "eventVersion": "1.02",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-11-04T00:52:40Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "keySpec": "AES_256",
        "encryptionContext": {
            "Department": "Engineering",
            "Project": "Alpha"
        }
    },
    "responseElements": null,
    "requestID": "e0eb83e3-63bc-11e4-bc2b-4198b6150d5c",
    "eventID": "a9dea4f9-8395-46c0-942c-f509c02c2b71",
    "readOnly": true,
    "resources": [{
        "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "accountId": "111122223333"
    }],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

GenerateDataKeyPair

The following example shows an AWS CloudTrail log entry for the GenerateDataKeyPair operation. This example records an operation that generates an RSA key pair encrypted under a symmetric encryption AWS KMS key.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::111122223333:user/Alice",
            "accountId": "111122223333",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
    },
    "eventTime": "2020-07-27T18:57:57Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKeyPair",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyPairSpec": "RSA_3072",
        "encryptionContext": {
            "Project": "Alpha"
        },
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "requestID": "52fb127b-0fe5-42bb-8e5e-f560febde6b0",
    "eventID": "9b6bd6d2-529d-4890-a949-593b13800ad7",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

GenerateDataKeyPairWithoutPlaintext

The following example shows an AWS CloudTrail log entry for the GenerateDataKeyPairWithoutPlaintext operation. This example records an operation that generates an RSA key pair that is encrypted under a symmetric encryption AWS KMS key.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::111122223333:user/Alice",
            "accountId": "111122223333",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
    },
    "eventTime": "2020-07-27T18:57:57Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKeyPairWithoutPlaintext",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyPairSpec": "RSA_4096",
        "encryptionContext": {
            "Index": "5"
        },
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "requestID": "52fb127b-0fe5-42bb-8e5e-f560febde6b0",
    "eventID": "9b6bd6d2-529d-4890-a949-593b13800ad7",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
```

```
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

GenerateDataKeyWithoutPlaintext

The following example shows an AWS CloudTrail log entry for the GenerateDataKeyWithoutPlaintext operation.

```
{
    "eventVersion": "1.02",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-11-04T00:52:23Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKeyWithoutPlaintext",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "errorCode": "InvalidKeyUsageException",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "keySpec": "AES_256",
        "encryptionContext": {
            "Project": "Alpha"
        }
    },
    "responseElements": null,
    "requestID": "d6b8e411-63bc-11e4-bc2b-4198b6150d5c",
    "eventID": "f7734272-9ec5-4c80-9f36-528ebbe35e4a",
    "readOnly": true,
    "resources": [{
        "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "accountId": "111122223333"
    }],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

GenerateMac

The following example shows an AWS CloudTrail log entry for the GenerateMac operation.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
     },
    "eventTime": "2022-12-23T19:26:54Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateMac",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "macAlgorithm": "HMAC_SHA_512",
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "requestID": "e0eb83e3-63bc-11e4-bc2b-4198b6150d5c",
    "eventID": "a9dea4f9-8395-46c0-942c-f509c02c2b71",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

GenerateRandom

The following example shows an AWS CloudTrail log entry for the GenerateRandom operation. Because this operation doesn't use an AWS KMS key, the `resources` field is empty.

```
{
    "eventVersion": "1.02",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-11-04T00:52:37Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateRandom",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": null,
    "responseElements": null,
    "requestID": "df1e3de6-63bc-11e4-bc2b-4198b6150d5c",
    "eventID": "239cb9f7-ae05-4c94-9221-6ea30eef0442",
    "readOnly": true,
    "resources": [],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

GetKeyPolicy

The following example shows an AWS CloudTrail log entry for the GetKeyPolicy operation. For information about viewing the key policy for a KMS key, see Viewing a key policy.

```
{
    "eventVersion": "1.02",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-11-04T00:50:30Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GetKeyPolicy",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "policyName": "default"
    },
    "responseElements": null,
    "requestID": "93746dd6-63bc-11e4-bc2b-4198b6150d5c",
    "eventID": "4aa7e4d5-d047-452a-a5a6-2cce282a7e82",
    "readOnly": true,
    "resources": [{
        "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "accountId": "111122223333"
    }],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

GetKeyRotationStatus

The following example shows an AWS CloudTrail log entry for the GetKeyRotationStatus operation. For information about automatic and on-demand rotation of key material for a KMS key, see Rotating AWS KMS keys.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2024-02-20T19:16:45Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GetKeyRotationStatus",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "requestID": "12f9b7e8-49b9-4c1c-a7e3-34ac0cdf0467",
    "eventID": "3d082126-9e7d-4167-8372-a6cfcbed4be6",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES256-GCM-SHA384",
        "clientProvidedHostHeader": "kms.us-east-1.amazonaws.com"
    }
}
```

GetParametersForImport

The following example shows an AWS CloudTrail log entry generated when you use the GetParametersForImport operation. This operation returns the public key and import token that you use when importing key material into a KMS key. The same CloudTrail entry is recorded when you use the `GetParametersForImport` operation or use the AWS KMS console to download the public key and import token.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::111122223333:user/Alice",
            "accountId": "111122223333",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
    },
    "eventTime": "2020-07-25T23:58:23Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GetParametersForImport",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "wrappingAlgorithm": "RSAES_OAEP_SHA_256",
        "wrappingKeySpec": "RSA_2048"
    },
    "responseElements": null,
    "requestID": "b5786406-e3c7-43d6-8d3c-6d5ef96e2278",
    "eventID": "4023e622-0c3e-4324-bdef-7f58193bba87",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

ImportKeyMaterial

The following example shows an AWS CloudTrail log entry generated when you use the ImportKeyMaterial operation. The same CloudTrail entry is recorded when you use the `ImportKeyMaterial` operation or use the AWS KMS console to import key material into an AWS KMS key.

CloudTrail log entries for this operation recorded on or after December 2022 include the key ARN of the affected KMS key in the `responseElements.keyId` value, even though this operation does not return the key ARN.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::111122223333:user/Alice",
            "accountId": "111122223333",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
    },
    "eventTime": "2020-07-26T00:08:00Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "ImportKeyMaterial",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "validTo": "Jan 1, 2021 8:00:00 PM",
        "expirationModel": "KEY_MATERIAL_EXPIRES"
    },
    "responseElements": {
        "keyId":"arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "requestID": "89e10ee7-a612-414d-95a2-a128346969fd",
    "eventID": "c7abd205-a5a2-4430-bbfa-fc10f3e2d79f",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

ListAliases

The following example shows an AWS CloudTrail log entry for the ListAliases operation. Because this operation doesn't use any particular alias or AWS KMS key, the `resources` field is empty. For information about viewing aliases in AWS KMS, see Viewing aliases.

```
{
    "eventVersion": "1.02",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-11-04T00:51:45Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "ListAliases",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "limit": 5,
        "marker":
  "eyJiIjoiYWxpYXMvZTU0Y2MxOTMtYTMwNC00YzEwLTliZWItYTJjZjA3NjA2OTJhIiwiYSI6ImFsaWFzL2U1NGNjMTkzL
    },
    "responseElements": null,
    "requestID": "bfe6c190-63bc-11e4-bc2b-4198b6150d5c",
    "eventID": "a27dda7b-76f1-4ac3-8b40-42dfba77bcd6",
    "readOnly": true,
    "resources": [],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

ListGrants

The following example shows an AWS CloudTrail log entry for the ListGrant operation. For information about grants in AWS KMS, see Grants in AWS KMS.

```
{
    "eventVersion": "1.02",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-11-04T00:52:49Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "ListGrants",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "marker":
 "eyJncmFudElkIjoiMWY4M2U2ZmM0YTY2NDgxYjQ2Yzc4MTdhM2Y4YmQwMDFkZDNiYmQ1MGVlYTMyY2RmOWFiNWY1Nzc1N
\u003d\u003d",
        "limit": 10
    },
    "responseElements": null,
    "requestID": "e5c23960-63bc-11e4-bc2b-4198b6150d5c",
    "eventID": "d24380f5-1b20-4253-8e92-dd0492b3bd3d",
    "readOnly": true,
    "resources": [{
        "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "accountId": "111122223333"
    }],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

ListKeyRotations

The following example shows an AWS CloudTrail log entry for the ListKeyRotations operation. For information about automatic and on-demand rotation of key material for a KMS key, see Rotating AWS KMS keys.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2024-02-20T19:16:45Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "ListKeyRotations",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "requestID": "99c88d32-f2db-455e-8a9a-23855258a452",
    "eventID": "8ce0e74b-b9c7-45a2-96ef-83136d38068e",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES256-GCM-SHA384",
        "clientProvidedHostHeader": "kms.us-east-1.amazonaws.com"
    }
}
```

PutKeyPolicy

The following example shows an AWS CloudTrail log entry generated by calling the PutKeyPolicy operation. For information about updating a key policy, see Changing a key policy.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2022-09-01T20:06:16Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "PutKeyPolicy",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "policyName": "default",
        "policy": "{\n  \"Version\" : \"2012-10-17\",\n  \"Id\" : \"key-default-1\",\n
 \"Statement\" : [ {\n    \"Sid\" : \"Enable IAM User Permissions\",\n    \"Effect\" :
 \"Allow\",\n    \"Principal\" : {\n      \"AWS\" : \"arn:aws:iam::111122223333:root
\"\n    },\n    \"Action\" : \"kms:*\",\n    \"Resource\" : \"*\"\n  } ]\n}",
        "bypassPolicyLockoutSafetyCheck": false
    },
    "responseElements": null,
    "requestID": "7bb906fa-dc21-4350-b65c-808ff0f72f55",
    "eventID": "c217db1f-903f-4a2f-8f88-9580182d6313",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

ReEncrypt

The following example shows an AWS CloudTrail log entry for the ReEncrypt operation. The `resources` field in this log entry specifies two AWS KMS keys, the source KMS key and the destination KMS key, in that order.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2020-07-27T23:09:13Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "ReEncrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "sourceEncryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "sourceEncryptionContext": {
            "Project": "Alpha",
            "Department": "Engineering"
        },
        "destinationKeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
        "destinationEncryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "destinationEncryptionContext": {
            "Level": "3A"
        }
    },
    "responseElements": null,
    "requestID": "03769fd4-acf9-4b33-adf3-2ab8ca73aadf",
    "eventID": "542d9e04-0e8d-4e05-bf4b-4bdeb032e6ec",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        },
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

ReplicateKey

The following example shows an AWS CloudTrail log entry generated by calling the ReplicateKey operation. A `ReplicateKey` request results in a `ReplicateKey` operation and a CreateKey operation.

For information about replicating multi-Region keys, see Creating multi-Region replica keys.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2020-11-18T01:29:18Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "ReplicateKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "replicaRegion": "us-west-2",
        "bypassPolicyLockoutSafetyCheck": false,
        "description": ""
    },
    "responseElements": {
        "replicaKeyMetadata": {
            "aWSAccountId": "111122223333",
            "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
            "arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "creationDate": "Nov 18, 2020, 1:29:18 AM",
            "enabled": false,
            "description": "",
            "keyUsage": "ENCRYPT_DECRYPT",
            "keyState": "Creating",
            "origin": "AWS_KMS",
            "keyManager": "CUSTOMER",
            "keySpec": "SYMMETRIC_DEFAULT",
            "customerMasterKeySpec": "SYMMETRIC_DEFAULT",
            "encryptionAlgorithms": [
                "SYMMETRIC_DEFAULT"
            ],
            "multiRegion": true,
            "multiRegionConfiguration": {
                "multiRegionKeyType": "REPLICA",
                "primaryKey": {
                    "arn": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
                    "region": "us-east-1"
                },
                "replicaKeys": [
                    {
                        "arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
                        "region": "us-west-2"
                    }
                ]
            }
        },
        "replicaPolicy": "{\n  \"Version\":\"2012-10-17\",\n  \"Statement\":[{\n
 \"Effect\":\"Allow\",\n    \"Principal\":{\"AWS\":\"arn:aws:iam::123456789012:user/
Alice\"},\n    \"Action\":\"kms:*\",\n    \"Resource\":\"*\"\n  }, {\n    \"Effect
\":\"Allow\",\n    \"Principal\":{\"AWS\":\"arn:aws:iam::012345678901:user/Bob\"},\n
    \"Action\":\"kms:CreateGrant\",\n    \"Resource\":\"*\"\n  }, {\n    \"Effect\":
\"Allow\",\n    \"Principal\":{\"AWS\":\"arn:aws:iam::012345678901:user/Charlie\"},\n
  \"Action\":\"kms:Encrypt\",\n    \"Resource\":\"*\"\n}]\n}",
    },
    "requestID": "abcdef68-63bc-11e4-bc2b-4198b6150d5c",
    "eventID": "fedcba44-6773-4f96-8763-1993aec9ae6a",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

RetireGrant

The following example shows an AWS CloudTrail log entry generated by calling the RetireGrant operation. For information about retiring grants, see Retiring and revoking grants.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2022-09-01T19:39:33Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "RetireGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": null,
    "responseElements": null,
    "additionalEventData": {
        "grantId": "abcde1237f76e4ba7987489ac329fbfba6ad343d6f7075dbd1ef191f0120514a"
    },
    "requestID": "1d274d57-5697-462c-a004-f25fcc29fa26",
    "eventID": "0771bcfb-3e24-4332-9ac8-e1c06563eecf",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

RevokeGrant

The following example shows an AWS CloudTrail log entry generated by calling the RevokeGrant operation. For information about revoking grants, see Retiring and revoking grants.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2022-09-01T19:35:17Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "RevokeGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "grantId": "abcde1237f76e4ba7987489ac329fbfba6ad343d6f7075dbd1ef191f0120514a"
    },
    "responseElements": null,
    "requestID": "59d94c03-c5b7-428d-ae6e-f2c4b47d2917",
    "eventID": "07a23a39-6526-4ae2-b31e-d35fbe9e24ee",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

RotateKey

These examples show the AWS CloudTrail log entries for the operations that rotate AWS KMS keys. For information about rotating KMS keys, see Rotating AWS KMS keys.

The following example shows a CloudTrail log entry for the operation that rotates a symmetric encryption KMS key on which automatic key rotation is enabled. For information about enabling automatic rotation, see How to enable and disable automatic key rotation.

For an example of the CloudTrail log entry that records the `EnableKeyRotation` operation, see EnableKeyRotation.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "accountId": "111122223333",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2021-01-14T01:41:59Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "RotateKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": null,
    "responseElements": null,
    "eventID": "a24b3967-ddad-417f-9b22-2332b918db06",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsServiceEvent",
    "recipientAccountId": "111122223333",
    "serviceEventDetails": {
        "rotationType": "AUTOMATIC",
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "eventCategory": "Management"
}
```

The following example shows a CloudTrail log entry for a RotateKeyOnDemand operation. For information about rotating symmetric encryption KMS keys on-demand, see How to perform on-demand key rotation.

For an example of the CloudTrail log entry that records the `RotateKeyOnDemand` operation, see RotateKeyOnDemand.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "accountId": "111122223333",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2021-01-14T01:41:59Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "RotateKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": null,
    "responseElements": null,
    "eventID": "a24b3967-ddad-417f-9b22-2332b918db06",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsServiceEvent",
    "recipientAccountId": "111122223333",
    "serviceEventDetails": {
        "rotationType": "ON_DEMAND",
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "eventCategory": "Management"
}
```

RotateKeyOnDemand

The following example shows an AWS CloudTrail log entry for the RotateKeyOnDemand operation. For an example of the CloudTrail log entry that is written when the key is rotated, see RotateKey. For more information about on-demand rotation of key material for a KMS key, see How to perform on-demand key rotation.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2024-02-20T17:41:57Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "RotateKeyOnDemand",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "requestID": "9e1dee86-eb84-42fd-8f25-e3fc7dbb32c8",
    "eventID": "00a09fbc-20d6-4a58-9b92-7da85984ab77",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES256-GCM-SHA384",
        "clientProvidedHostHeader": "kms.us-east-1.amazonaws.com"
    }
}
```

ScheduleKeyDeletion

These examples show AWS CloudTrail log entries for the ScheduleKeyDeletion operation.

For an example of the CloudTrail log entry that is written when the key is deleted, see DeleteKey. For information about deleting AWS KMS keys, see Deleting AWS KMS keys.

The following example records a `ScheduleKeyDeletion` request for a single-Region KMS key.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::111122223333:user/Alice",
            "accountId": "111122223333",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
    },
    "eventTime": "2021-03-23T18:58:30Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "ScheduleKeyDeletion",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "pendingWindowInDays": 20,
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "keyState": "PendingDeletion",
        "deletionDate": "Apr 12, 2021 18:58:30 PM"
    },
    "requestID": "ee408f36-ea01-422b-ac14-b0f147c68334",
    "eventID": "3c4226b0-1e81-48a8-a333-7fa5f3cbd118",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

The following example records a `ScheduleKeyDeletion` request for a multi-Region KMS key with replica keys.

Because AWS KMS won't delete a multi-Region key until all of its replica keys are deleted, in the `responseElements` field, the `keyState` is `PendingReplicaDeletion` and the `deletionDate` field is omitted.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::111122223333:user/Alice",
            "accountId": "111122223333",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
    },
    "eventTime": "2021-10-28T17:59:05Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "ScheduleKeyDeletion",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "pendingWindowInDays": 30,
        "keyId": "mrk-1234abcd12ab34cd56ef1234567890ab"
    },
    "responseElements": {
        "keyId": "arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
        "keyState": "PendingReplicaDeletion",
        "pendingWindowInDays": 30
    },
    "requestID": "12341411-d846-42a6-a476-b1cbe3011f89",
    "eventID": "abcda5f-396d-494c-9380-0c47860df5f1",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

The following example records a `ScheduleKeyDeletion` request for a KMS key in an AWS CloudHSM custom key store.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::111122223333:user/Alice",
            "accountId": "111122223333",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
    },
    "eventTime": "2021-10-26T23:25:25Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "ScheduleKeyDeletion",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321",
        "pendingWindowInDays": 30
    },
    "responseElements": {
        "keyId": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321",
        "deletionDate": "Nov 2, 2021, 11:25:25 PM",
        "keyState": "PendingDeletion",
        "pendingWindowInDays": 30
    },
    "additionalEventData": {
        "customKeyStoreId": "cks-1234567890abcdef0",
        "clusterId": "cluster-1a23b4cdefg",
        "backingKeys": "[{\"keyHandle\":\"01\",\"backingKeyId\":\"backing-key-id\"}]"
    },
    "requestID": "abcd9f60-2c9c-4a0b-a456-d5d998f7f321",
    "eventID": "ca01996a-01b0-4edd-bbbb-25d7b6d1a6fa",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

Sign

These examples show AWS CloudTrail log entries for the Sign operation.

The following example shows an CloudTrail log entry for a Sign operation that uses an asymmetric RSA KMS key to generate a digital signature for a file.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2022-03-07T22:36:44Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Sign",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "messageType": "RAW",
        "keyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
        "signingAlgorithm": "RSASSA_PKCS1_V1_5_SHA_256"
    },
    "responseElements": null,
    "requestID": "8d0b35e0-46cf-48b9-be99-bf2ebc9ab9fb",
    "eventID": "107b3cac-b125-4556-9702-12a2b9afcff7",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

SynchronizeMultiRegionKey

The following example shows an AWS CloudTrail log entry generated when AWS KMS synchronizes a multi-Region key. Synchronizing involves cross-Region calls to copy the shared properties of a multi-Region primary key to its replica keys. AWS KMS synchronizes multi-Region keys periodically to assure that all related multi-Region keys have the same key material.

The `resources` element of the CloudTrail log entry includes the key ARN of the multi-Region primary key, including its AWS Region. The related multi-Region replica keys and their Regions are not listed in this log entry.

CloudTrail log entries for this operation recorded on or after December 2022 include the key ARN of the affected KMS key in the `responseElements.keyId` value, even though this operation does not return the key ARN.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "accountId": "111122223333",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2020-11-18T02:04:37Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "SynchronizeMultiRegionKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": null,
    "responseElements": {
        "keyId":"arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "requestID": "12345681-de97-42e9-bed0-b02ae1abd8dc",
    "eventID": "abcdec99-2b5c-4670-9521-ddb8f031e146",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

TagResource

The following example shows an AWS CloudTrail log entry of a call to the TagResource operation to add a tag with a tag key of `Department` and a tag value of `IT`.

For an example of an `UntagResource` CloudTrail log entry that is written when the key is rotated, see UntagResource. For information about tagging AWS KMS keys, see Tagging keys.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2020-07-01T21:19:25Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "TagResource",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "tags": [
            {
                "tagKey": "Department",
                "tagValue": "IT"
            }
        ]
    },
    "responseElements": null,
    "requestID": "b942584a-f77d-4787-9feb-b9c5be6e746d",
    "eventID": "0a091b9b-0df5-4cf9-b667-6f2879532b8f",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

UntagResource

The following example shows an AWS CloudTrail log entry of a call to the UntagResource operation to delete a tag with a tag key of `Dept`.

CloudTrail log entries for this operation recorded on or after December 2022 include the key ARN of the affected KMS key in the `responseElements.keyId` value, even though this operation does not return the key ARN.

For an example of an `TagResource` CloudTrail log entry, see TagResource. For information about tagging AWS KMS keys, see Tagging keys.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2020-07-01T21:19:19Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "UntagResource",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "tagKeys": [
            "Dept"
        ]
    },
    "responseElements": {
        "keyId":"arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "requestID": "cb1d507b-6015-47f4-812b-179713af8068",
    "eventID": "0b00f4b0-036e-411d-aa75-87eb4a35a4b3",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

UpdateAlias

The following example shows an AWS CloudTrail log entry for the UpdateAlias operation. The `resources` element includes fields for the alias and KMS key resources. For information about creating aliases in AWS KMS, see Creating an alias.

CloudTrail log entries for this operation recorded on or after December 2022 include the key ARN of the affected KMS key in the `responseElements.keyId` value, even though this operation does not return the key ARN.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
         "principalId": "EX_PRINCIPAL_ID",
         "arn": "arn:aws:iam::111122223333:user/Alice",
         "accountId": "111122223333",
         "accessKeyId": "EXAMPLE_KEY_ID",
         "userName": "Alice"
    },
    "eventTime": "2020-11-13T23:18:15Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "UpdateAlias",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "aliasName": "alias/my_alias",
        "targetKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": {
        "keyId":"arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "requestID": "d9472f40-63bc-11e4-bc2b-4198b6150d5c",
    "eventID": "f72d3993-864f-48d6-8f16-e26e1ae8dff0",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:111122223333:alias/my_alias"
        },
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

UpdateCustomKeyStore

The following example shows an AWS CloudTrail log entry generated by calling the `UpdateCustomKeyStore` operation to update the cluster ID for a custom key store. For information about editing custom key stores, see Editing AWS CloudHSM key store settings.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2021-10-21T20:17:32Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "UpdateCustomKeyStore",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "customKeyStoreId": "cks-1234567890abcdef0",
        "clusterId": "cluster-1a23b4cdefg"
    },
    "responseElements": null,
    "additionalEventData": {
        "customKeyStoreName": "ExampleKeyStore",
        "clusterId": "cluster-1a23b4cdefg"
    },
    "requestID": "abcde9e1-f1a3-4460-a423-577fb6e695c9",
    "eventID": "114b61b9-0ea6-47f5-a9d2-4f2bdd0017d5",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333"
}
```

UpdateKeyDescription

The following example shows an AWS CloudTrail log entry generated by calling the UpdateKeyDescription operation.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2022-09-01T19:22:40Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "UpdateKeyDescription",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "description": "New key description"
    },
    "responseElements": null,
    "requestID": "8c3c1f8b-336d-4896-b034-4eb9916bc9b3",
    "eventID": "f5f3d548-2e9e-4658-8427-9dcb5b1ea791",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

UpdatePrimaryRegion

The following example shows the AWS CloudTrail log entries that are generated by calling the UpdatePrimaryRegion operation on a multi-Region key.

The `UpdatePrimaryRegion` operation writes two CloudTrail log entries: one in the Region with the multi-Region primary key that is converted to a replica key, and one in the Region with a multi-Region replica key that is converted to a primary key.

CloudTrail log entries for this operation recorded on or after December 2022 include the key ARN of the affected KMS key in the `responseElements.keyId` value, even though this operation does not return the key ARN.

The following example shows a CloudTrail log entry for `UpdatePrimaryRegion` in the Region where the multi-Region key changed from a primary key to a replica key (us-west-2). The `primaryRegion` field shows the Region that now hosts the primary key (ap-northeast-1).

```
{
    "eventVersion": "1.08",
    "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::111122223333:user/Alice",
            "accountId": "111122223333",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
    },
    "eventTime": "2021-03-10T20:23:37Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "UpdatePrimaryRegion",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "mrk-1234abcd12ab34cd56ef1234567890ab",
        "primaryRegion": "ap-northeast-1"
    },
    "responseElements": {
        "keyId":"arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "requestID": "ee408f36-ea01-422b-ac14-b0f147c68334",
    "eventID": "3c4226b0-1e81-48a8-a333-7fa5f3cbd118",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333"
}
```

The following example represents the CloudTrail log entry for `UpdatePrimaryRegion` in the Region where the multi-Region key changed from a replica key to a primary key (ap-northeast-1). This log entry doesn't identify the previous primary Region.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::111122223333:user/Alice",
            "accountId": "111122223333",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice",
            "invokedBy": "kms.amazonaws.com"
    },
    "eventTime": "2021-03-10T20:23:37Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "UpdatePrimaryRegion",
    "awsRegion": "ap-northeast-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "arn:aws:kms:ap-northeast-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
        "primaryRegion": "ap-northeast-1"
    },
    "responseElements": {
        "keyId":"arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "requestID": "ee408f36-ea01-422b-ac14-b0f147c68334",
    "eventID": "091e6be5-737f-43c6-8431-e3679d6d0619",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333"
}
```

VerifyMac

The following example shows an AWS CloudTrail log entry for the VerifyMac operation.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
     },
    "eventTime": "2022-03-31T19:25:54Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "VerifyMac",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "macAlgorithm": "HMAC_SHA_384",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "requestID": "f35da560-edff-4d6e-9b40-fb306fa9ef1e",
    "eventID": "6b464487-6dea-44cd-84ad-225d7450c975",
    "readOnly": true,
    "resources": [
        {
           "accountId": "111122223333",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

Verify

These examples show AWS CloudTrail log entries for the Verify operation.

The following example shows an CloudTrail log entry for a Verify operation that uses an asymmetric RSA KMS key to verify a digital signature.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2022-03-07T22:50:41Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Verify",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "signingAlgorithm": "RSASSA_PKCS1_V1_5_SHA_256",
        "keyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
        "messageType": "RAW"
    },
    "responseElements": null,
    "requestID": "c73ab82a-af82-4750-ae2c-b6bb790e9c28",
    "eventID": "3b4331cd-5b7b-4de5-bf5f-82ec22f0dac0",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

Amazon EC2 example one

The following example records an IAM principal creating an encrypted volume using the default volume key in the Amazon EC2 management console.

The following example shows a CloudTrail log entry in which user Alice creates an encrypted volume with a default volume key in the Amazon EC2 management console. The EC2 log file record includes a `volumeId` field with a value of `"vol-13439757"`. The AWS KMS record contains an `encryptionContext` field with a value of `"aws:ebs:id": "vol-13439757"`. Similarly, the `principalId` and `accountId` between the two records match. The records reflect the fact that creating an encrypted volume generates a data key that is used to encrypt the volume content.

```
{
  "Records": [
     {
      "eventVersion": "1.02",
      "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::111122223333:user/Alice",
            "accountId": "111122223333",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
      "eventTime": "2014-11-05T20:50:18Z",
      "eventSource": "ec2.amazonaws.com",
      "eventName": "CreateVolume",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "AWS Internal",
      "requestParameters": {
        "size": "10",
        "zone": "us-east-1a",
        "volumeType": "gp2",
        "encrypted": true
      },
      "responseElements": {
        "volumeId": "vol-13439757",
        "size": "10",
        "zone": "us-east-1a",
        "status": "creating",
        "createTime": 1415220618876,
        "volumeType": "gp2",
        "iops": 30,
        "encrypted": true
      },
      "requestID": "1565210e-73d0-4912-854c-b15ed349e526",
      "eventID": "a3447186-135f-4b00-8424-bc41f1a93b4f",
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    },
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
      },
      "eventTime": "2014-11-05T20:50:19Z",
      "eventSource": "kms.amazonaws.com",
      "eventName": "GenerateDataKeyWithoutPlaintext",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "&AWS; Internal",
      "requestParameters": {
        "encryptionContext": {
          "aws:ebs:id": "vol-13439757"
        },
        "numberOfBytes": 64,
        "keyId": "alias/aws/ebs"
      },
      "responseElements": null,
      "requestID": "create-123456789012-758241111-1415220618",
      "eventID": "4bd2a696-d833-48cc-b72c-05e61b608399",
      "readOnly": true,
      "resources": [
        {
          "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
          "accountId": "111122223333"
        }
      ],
      "eventType": "AwsApiCall",
      "recipientAccountId": "111122223333"
    }
  ]
}
```

Amazon EC2 example two

In the following example, an IAM principal running an Amazon EC2 instance creates and mounts a data volume that is encrypted under a KMS key. This action generates multiple CloudTrail log records.

When the volume is created, Amazon EC2, acting on behalf of the customer, gets an encrypted data key from AWS KMS (`GenerateDataKeyWithoutPlaintext`). Then it creates a grant (`CreateGrant`) that allows it to decrypt the data key. When the volume is mounted, Amazon EC2 calls AWS KMS to decrypt the data key (`Decrypt`).

The `instanceId` of the Amazon EC2 instance, `"i-81e2f56c"`, appears in the `RunInstances` event. The same instance ID qualifies the `granteePrincipal` of the grant that is created (`"111122223333:aws:ec2-infrastructure:i-81e2f56c"`) and the assumed role that is the principal in the `Decrypt` call (`"arn:aws:sts::111122223333:assumed-role/aws:ec2-infrastructure/i-81e2f56c"`).

The key ARN of the KMS key that protects the data volume, `arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab`, appears in all three AWS KMS calls (`CreateGrant`, `GenerateDataKeyWithoutPlaintext`, and `Decrypt`).

```
{
  "Records": [
    {
      "eventVersion": "1.02",
      "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::111122223333:user/Alice",
            "accountId": "111122223333",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
      },
      "eventTime": "2014-11-05T21:35:27Z",
      "eventSource": "ec2.amazonaws.com",
      "eventName": "RunInstances",
      "awsRegion": "us-west-2",
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "AWS Internal",
      "requestParameters": {
        "instancesSet": {
          "items": [
            {
              "imageId": "ami-b66ed3de",
              "minCount": 1,
              "maxCount": 1
            }
          ]
        },
        "groupSet": {
          "items": [
            {
              "groupId": "sg-98b6e0f2"
            }
          ]
        },
        "instanceType": "m3.medium",
        "blockDeviceMapping": {
          "items": [
            {
              "deviceName": "/dev/xvda",
              "ebs": {
                "volumeSize": 8,
                "deleteOnTermination": true,
                "volumeType": "gp2"
              }
            },
            {
              "deviceName": "/dev/sdb",
              "ebs": {
                "volumeSize": 8,
                "deleteOnTermination": false,
                "volumeType": "gp2",
                "encrypted": true
              }
            }
          ]
        },
        "monitoring": {
          "enabled": false
        },
        "disableApiTermination": false,
        "instanceInitiatedShutdownBehavior": "stop",
        "clientToken": "XdKUT141516171819",
        "ebsOptimized": false
      },
      "responseElements": {
        "reservationId": "r-5ebc9f74",
        "ownerId": "111122223333",
        "groupSet": {
          "items": [
            {
              "groupId": "sg-98b6e0f2",
              "groupName": "launch-wizard-2"
            }
          ]
        },
        "instancesSet": {
          "items": [
            {
              "instanceId": "i-81e2f56c",
              "imageId": "ami-b66ed3de",
              "instanceState": {
                "code": 0,
                "name": "pending"
              },
              "amiLaunchIndex": 0,
              "productCodes": {

              },
              "instanceType": "m3.medium",
              "launchTime": 1415223328000,
              "placement": {
                "availabilityZone": "us-east-1a",
                "tenancy": "default"
              },
              "monitoring": {
                "state": "disabled"
              },
              "stateReason": {
                "code": "pending",
                "message": "pending"
              },
              "architecture": "x86_64",
              "rootDeviceType": "ebs",
              "rootDeviceName": "/dev/xvda",
              "blockDeviceMapping": {

              },
              "virtualizationType": "hvm",
              "hypervisor": "xen",
              "clientToken": "XdKUT1415223327917",
              "groupSet": {
                "items": [
                  {
                    "groupId": "sg-98b6e0f2",
                    "groupName": "launch-wizard-2"
                  }
                ]
              },
              "networkInterfaceSet": {

              },
              "ebsOptimized": false
            }
          ]
        }
      },
      "requestID": "41c4b4f7-8bce-4773-bf0e-5ae3bb5cbce2",
      "eventID": "cd75a605-2fee-4fda-b847-9c3d330ebaae",
      "eventType": "AwsApiCall",
      "recipientAccountId": "111122223333"
    },
    {
      "eventVersion": "1.02",
      "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::111122223333:user/Alice",
            "accountId": "111122223333",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
      },
      "eventTime": "2014-11-05T21:35:35Z",
      "eventSource": "kms.amazonaws.com",
      "eventName": "CreateGrant",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "AWS Internal",
      "requestParameters": {
        "constraints": {
          "encryptionContextSubset": {
            "aws:ebs:id": "vol-f67bafb2"
          }
        },
        "granteePrincipal": "111122223333:aws:ec2-infrastructure:i-81e2f56c",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
      },
      "responseElements": {
        "grantId": "abcde1237f76e4ba7987489ac329fbfba6ad343d6f7075dbd1ef191f0120514a"
      },
      "requestID": "41c4b4f7-8bce-4773-bf0e-5ae3bb5cbce2",
      "eventID": "c1ad79e3-0d3f-402a-b119-d5c31d7c6a6c",
      "readOnly": false,
      "resources": [
        {
          "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
          "accountId": "111122223333"
        }
      ],
      "eventType": "AwsApiCall",
      "recipientAccountId": "111122223333"
    },
    {
      "eventVersion": "1.02",
      "userIdentity": {
          "type": "IAMUser",
          "principalId": "EX_PRINCIPAL_ID",
          "arn": "arn:aws:iam::111122223333:user/Alice",
          "accountId": "111122223333",
          "accessKeyId": "EXAMPLE_KEY_ID",
          "userName": "Alice"
      },
      "eventTime": "2014-11-05T21:35:32Z",
      "eventSource": "kms.amazonaws.com",
      "eventName": "GenerateDataKeyWithoutPlaintext",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "AWS Internal",
      "requestParameters": {
        "encryptionContext": {
          "aws:ebs:id": "vol-f67bafb2"
        },
        "numberOfBytes": 64,
        "keyId": "alias/aws/ebs"
      },
      "responseElements": null,
      "requestID": "create-111122223333-758247346-1415223332",
      "eventID": "ac3cab10-ce93-4953-9d62-0b6e5cba651d",
      "readOnly": true,
      "resources": [
        {
          "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
          "accountId": "111122223333"
        }
      ],
      "eventType": "AwsApiCall",
      "recipientAccountId": "111122223333"
    },
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "AssumedRole",
        "principalId": "111122223333:aws:ec2-infrastructure:i-81e2f56c",
        "arn": "arn:aws:sts::111122223333:assumed-role/aws:ec2-infrastructure/
i-81e2f56c",
        "accountId": "111122223333",
        "accessKeyId": "",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2014-11-05T21:35:38Z"
          },
          "sessionIssuer": {
            "type": "Role",
            "principalId": "111122223333:aws:ec2-infrastructure",
            "arn": "arn:aws:iam::111122223333:role/aws:ec2-infrastructure",
            "accountId": "111122223333",
            "userName": "aws:ec2-infrastructure"
          }
        }
      },
      "eventTime": "2014-11-05T21:35:47Z",
      "eventSource": "kms.amazonaws.com",
      "eventName": "Decrypt",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "192.0.2.0",
      "requestParameters": {
        "encryptionContext": {
          "aws:ebs:id": "vol-f67bafb2"
        }
      },
      "responseElements": null,
      "requestID": "b4b27883-6533-11e4-b4d9-751f1761e9e5",
      "eventID": "edb65380-0a3e-4123-bbc8-3d1b7cff49b0",
      "readOnly": true,
      "resources": [
        {
          "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
          "accountId": "111122223333"
        }
      ],
      "eventType": "AwsApiCall",
      "recipientAccountId": "111122223333"
    }
  ]
}
```

Monitoring with Amazon CloudWatch

You can monitor your AWS KMS keys using Amazon CloudWatch, an AWS service that collects and processes raw data from AWS KMS into readable, near real-time metrics. These data are recorded for a period of two weeks so that you can access historical information and gain a better understanding of the usage of your KMS keys and their changes over time.

You can use Amazon CloudWatch to alert you to important events, such as the following ones.

- The imported key material in a KMS key is nearing its expiration date.
- A KMS key that is pending deletion is still being used.

- The key material in a KMS key was automatically rotated.
- A KMS key was deleted.

You can also create an Amazon CloudWatch alarm that alerts you when your request rate reaches a certain percentage of a quota value. For details, see Manage your AWS KMS API request rates using Service Quotas and Amazon CloudWatch in the *AWS Security Blog*.

Topics

- AWS KMS metrics and dimensions
- Viewing AWS KMS metrics
- Creating CloudWatch alarms to monitor KMS keys

AWS KMS metrics and dimensions

AWS KMS predefines Amazon CloudWatch metrics to make it easier for you to monitor critical data and create alarms. You can view the AWS KMS metrics using the AWS Management Console and the Amazon CloudWatch API.

This section lists each AWS KMS metrics and the dimensions for each metric, and provides some basic guidance for creating CloudWatch alarms based on these metrics and dimensions.

> **Note**
>
> **Dimension group name**:
> To view a metric in the Amazon CloudWatch console, in the **Metrics** section, select the dimension group name. Then you can filter by the **Metric name**. This topic includes the metric name and dimension group name for each AWS KMS metric.

Topics

- SecondsUntilKeyMaterialExpiration
- ExternalKeyStoreThrottle
- XksProxyCertificateDaysToExpire
- XksProxyCredentialAge
- XksProxyErrors

- XksExternalKeyManagerStates
- XksProxyLatency

SecondsUntilKeyMaterialExpiration

The number of seconds remaining until the imported key material in a KMS key expires. This metric is valid only for KMS keys with imported key material (a key material origin of `EXTERNAL`) and an expiration date.

Use this metric to track the time that remains until your imported key material expires. When that time falls below a threshold that you define, you might want to reimport the key material with a new expiration date. The `SecondsUntilKeyMaterialExpiration` metric is specific to a KMS key. You cannot use this metric to monitor multiple KMS keys or KMS keys that you might create in the future. For help with creating a CloudWatch alarm to monitor this metric, see Creating a CloudWatch alarm for expiration of imported key material.

The most useful statistic for this metric is `Minimum`, which tells you the smallest amount of time remaining for all data points in the specified statistical period. The only valid unit for this metric is `Seconds`.

Dimension group name: **Per-Key Metrics**

Dimensions for `SecondsUntilKeyMaterialExpiration`

Dimension	Description; related to AWS
KeyId	Value for each KMS key.

ExternalKeyStoreThrottle

The number of requests for cryptographic operations on KMS keys in each external key store that AWS KMS throttles (responds with a `ThrottlingException`). This metric applies only to external key stores.

The `ExternalKeyStoreThrottle` metric applies only to KMS keys in an external key store and only to requests for cryptographic operations and the DescribeKey operation. AWS KMS throttles these requests when the request rate exceeds the custom key store request quota for your external key store. This metric does not include throttling by your external key store proxy or external key manager.

Use this metric to review and adjust the value of the your custom key store request quota. If this metric indicates that AWS KMS is frequently throttling your requests for these KMS keys, you might consider requesting an increase in your custom key store request quota value. For help, see Requesting a quota increase in the *Service Quotas User Guide*.

If you are getting very frequent `KMSInvalidStateException` errors with a message that explains that the request was rejected "due to a very high request rate" or the request was rejected "because the external key store proxy did not respond in time," it might indicate that your external key manager or external key store proxy cannot keep pace with the current request rate. If possible, lower your request rate. You might also consider requesting a decrease in your custom key store request quota value. Decreasing this quota value might increase throttling (and the `ExternalKeyStoreThrottle` metric value), but it indicates that AWS KMS is rejecting excess requests quickly before they are sent to your external key store proxy or external key manager. To request a quota decrease, please visit the AWS Support Center and create a case.

Dimension group name: **Keystore Throttle Metrics**

Dimension	Description
CustomKeyStoreId	Value for each external key store.
KmsOperation	Value for each AWS KMS API operation. This metric applies only to cryptographic operations and the `DescribeKey` operation on KMS keys in an external key store.
KeySpec	Value for each type of KMS key. The only supported key spec for KMS keys in an external key store is SYMMETRIC_DEFAULT.

XksProxyCertificateDaysToExpire

The number of days until the TLS certificate for your external key store proxy endpoint (`XksProxyUriEndpoint`) expires. This metric applies only to external key stores.

Use this metric to create a CloudWatch alarm that notifies you about the upcoming expiration of your TLS certificate. When the certificate expires, AWS KMS cannot communicate with the external key store proxy. All data protected by KMS keys in your external key store becomes inaccessible until you renew the certificate.

A certificate alarm prevents certificate expiration that might prevent you from accessing your encrypted resources. Set the alarm to give your organization time to renew the certificate before it expires.

Dimension group name: **XKS Proxy Certificate Metrics**

Dimension	Description
CustomKeyStoreId	Value for each external key store.
CertificateName	Subject name (CN) in the TLS certificate.

XksProxyCredentialAge

The number of days since the current external key store proxy authentication credential (`XksProxyAuthenticationCredential`) was associated with the external key store. This count begins when you enter the authentication credential as part of creating or updating your external key store. This metric applies only to external key stores.

This value is designed to remind you about the age of your authentication credential. However, because we begin the count when you associate the credential with your external key store, not when you create your authentication credential on your external key store proxy, this might not be an accurate indicator of the credential age on the proxy.

Use this metric to create a CloudWatch alarm that reminds you to rotate your external key store proxy authentication credential.

Dimension group name: **Per-Keystore Metrics**

Dimension	Description
CustomKeyStoreId	Value for each external key store.

XksProxyErrors

The number of exceptions related to AWS KMS requests to your external key store proxy. This count includes exceptions that the external key store proxy returns to AWS KMS and timeout errors that occur when the external key store proxy does not respond to AWS KMS within the 250 millisecond timeout interval. This metric applies only to external key stores.

Use this metric to track the error rate of KMS keys in your external key store. It reveals the most frequent errors, so you can prioritize your engineering effort. For example, KMS keys that are generating high rates of non-retryable errors might indicate a problem with the configuration of your external key store. To view your external key store configuration, see Viewing an external key store. To edit your external key store settings, see Editing external key store properties.

Dimension group name: **XKS Proxy Error Metrics**

Dimension	Description
CustomKeyStoreId	Value for each external key store.
KmsOperation	Value for each AWS KMS API operation that generated a request to the XKS proxy.
XksOperation	Value for each external key store proxy API operation.
KeySpec	Value for each type of KMS key. The only supported key spec for KMS keys in an external key store is SYMMETRIC_DEFAULT.
ErrorType	Values: • Retryable errors: Likely to be transient, such as networking errors. • Non-retryable errors: Likely to indicate a problem with the custom key store configuration or external components. • N/A: Successful request; no errors
ExceptionName	Values: • Name of the exception

Dimension	Description
	- None: Successful request; no errors

XksExternalKeyManagerStates

A count of the number of external key manager instances in each of the following health states: `Active`, `Degraded`, and `Unavailable`. The information for this metric comes from the external key store proxy associated with each external key store. This metric applies only to external key stores.

The following are the health states for the external key manager instances associated with an external key store. Each external key store proxy might use different indicators to measure the health states of your external key manager. For details, see the documentation for your external key store proxy.

- `Active`: The external key manager is healthy.
- `Degraded`: The external key manager is unhealthy, but can still serve traffic
- `Unavailable`: The external key manager cannot serve traffic.

Use this metric to create a CloudWatch alarm that alerts you to degraded and unavailable external key manager instances. To determine which external key manager instances are in each state, consult your external key store proxy logs.

Dimension group name: **XKS External Key Manager Metrics**

Dimension	Description
CustomKeyStoreId	Value for each external key store.
XksExternalKeyManagerState	Value for each health state.

XksProxyLatency

The number of milliseconds it takes for an external key store proxy to respond to an AWS KMS request. If the request timed out, the recorded value is the 250 millisecond timeout limit. This metric applies only to external key stores.

Use this metric to evaluate the performance of your external key store proxy and external key manager. For example, if the proxy is frequently timing out on encryption and decryption operations, consult your external proxy administrator.

Slow responses might also indicate that your external key manager cannot handle the current request traffic. AWS KMS recommends that your external key manager be able to handle up to 1800 requests for cryptographic operations per second. If your external key manager cannot handle the 1800 requests per second rate, consider requesting a decrease in your request quota for KMS keys in a custom key store. Requests for cryptographic operations using the KMS keys in your external key store will fail fast with a throttling exception, rather than being processed and later rejected by your external key store proxy or external key manager.

Dimension group name: **XKS Proxy Latency Metrics**

Dimension	Description
CustomKeyStoreId	Value for each external key store.
KmsOperation	Value for each AWS KMS API operation that generated a request to the XKS proxy.
XksOperation	Value for each external key store proxy API operation.
KeySpec	Value for each type of KMS key. The only supported key spec for KMS keys in an external key store is SYMMETRIC_DEFAULT.

Viewing AWS KMS metrics

You can view the AWS KMS metrics using the AWS Management Console and the Amazon CloudWatch API.

To view metrics using the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. If necessary, change the region. From the navigation bar, select the region where your AWS resources reside.
3. In the navigation pane, choose **Metrics**, **All metrics**.
4. On the **Browse** tab, search for `KMS`, and them choose **KMS**.
5. Choose the dimension group name of the metric you want to view.

   For example, for the `SecondsUntilKeyMaterialExpiration` metric, choose **Per-Key Metrics**.
6. For a graph of the metric value, choose the metric name, then choose `Add to graph`. To convert the line graph to a value, choose **Line**, then choose **Number**.

To view metrics using the Amazon CloudWatch API

To view AWS KMS metrics using the CloudWatch API, send a ListMetrics request with `Namespace` set to `AWS/KMS`. The following example shows how to do this with the AWS Command Line Interface (AWS CLI).

```
$  aws cloudwatch list-metrics --namespace AWS/KMS

{
    "Metrics": [
        {
            "Namespace": "AWS/KMS",
            "MetricName": "SecondsUntilKeyMaterialExpiration",
            "Dimensions": [
                {
                    "Name": "KeyId",
                    "Value": "1234abcd-12ab-34cd-56ef-1234567890ab"
                }
            ]
        },
        {
            "Namespace": "AWS/KMS",
            "MetricName": "ExternalKeyStoreThrottle",
            "Dimensions": [
                {
                    "Name": "CustomKeyStoreId",
                    "Value": "cks-1234567890abcdef0"
                },
                {
                    "Name": "KmsOperation",
                    "Value": "Encrypt"
                },
                {
                    "Name": "KeySpec",
                    "Value": "SYMMETRIC_DEFAULT"
                }
            ]
        },
        {
            "Namespace": "AWS/KMS",
            "MetricName": "XksProxyCertificateDaysToExpire",
            "Dimensions": [
                {
                    "Name": "CustomKeyStoreId",
                    "Value": "cks-1234567890abcdef0"
                },
                {
                    "Name": "CertificateName",
                    "Value": "myproxy.xks.example.com"
                }
            ]
        },
        {
            "Namespace": "AWS/KMS",
            "MetricName": "XksProxyCredentialAge",
            "Dimensions": [
                {
                    "Name": "CustomKeyStoreId",
                    "Value": "cks-1234567890abcdef0"
                }
            ]
        },
        {
            "Namespace": "AWS/KMS",
            "MetricName": "XksProxyErrors",
            "Dimensions": [
                {
                    "Name": "CustomKeyStoreId",
                    "Value": "cks-1234567890abcdef0"
                },
                {
                    "Name": "KmsOperation",
                    "Value": "Decrypt"
                },
                {
                    "Name": "XksOperation",
                    "Value": "Decrypt"
                },
                {
                     "Name": "KeySpec",
                     "Value": "SYMMETRIC_DEFAULT"
                },
                {
                    "Name": "ErrorType",
                    "Value": "Retryable errors"
                },
                {
                     "Name": "ExceptionName",
                     "Value": "KMSInvalidStateException"
                }
            ]
        },
        {
            "Namespace": "AWS/KMS",
            "MetricName": "XksProxyHsmStates",
            "Dimensions": [
                {
                    "Name": "CustomKeyStoreId",
                    "Value": "cks-1234567890abcdef0"
                },
                {
                    "Name": "XksProxyHsmState",
                    "Value": "Active"
                }
            ]
        },
        {
            "Namespace": "AWS/KMS",
            "MetricName": "XksProxyLatency",
            "Dimensions": [
                {
                    "Name": "CustomKeyStoreId",
                    "Value": "cks-1234567890abcdef0"
                },
                {
                    "Name": "KmsOperation",
                    "Value": "Decrypt"
                },
                {
                    "Name": "XksOperation",
                    "Value": "Decrypt"
                },
                {
                     "Name": "KeySpec",
                     "Value": "SYMMETRIC_DEFAULT"
                }
            ]
        }
    ]
}
```

Creating CloudWatch alarms to monitor KMS keys

You can create an Amazon CloudWatch alarm based on an AWS KMS metric. The alarm sends an email message when a metric value exceeds a threshold specified in the alarm configuration. The alarm can send the email message to an Amazon Simple Notification Service (Amazon SNS) topic or an Amazon EC2 Auto Scaling policy. For detailed information about CloudWatch alarms, see Using Amazon CloudWatch alarms in the Amazon CloudWatch User Guide

Create an alarm for expiring imported key material

You can use the `SecondsUntilKeyMaterialExpiration` metric to create a CloudWatch alarm that notifies you when the imported key material in a KMS key is about to expire.

When you import key material into a KMS key, you can optionally specify a date and time when the key material expires. When the key material expires, AWS KMS deletes the key material and the KMS key becomes unusable. To use the KMS key again, you must reimport the key material.

For instructions, see Creating a CloudWatch alarm for expiration of imported key material.

Create an alarm for use of KMS keys that are pending deletion

When you schedule deletion of a KMS key, AWS KMS enforces a waiting period before deleting the KMS key. You can use the waiting period to ensure that you don't need the KMS key now or in the future. You can also configure a CloudWatch alarm to warn you if a person or application attempts to use the KMS key in a cryptographic operation during the waiting period. If you receive a notification from such an alarm, you might want to cancel deletion of the KMS key.

For instructions, see Creating an alarm that detects use of a KMS key pending deletion.

Create an alarm to monitor an external key store

You can create CloudWatch alarms based on the metrics for external key stores and KMS keys in external key stores.

For example, we recommend that you set a CloudWatch alarm to notify you when the TLS certificate for your external key store is about to expire (XksProxyCertificateDaysToExpire), when your and when your external key store proxy reports that your external key manager instances are in a degraded or unavailable state (XksProxyHsmStates).

For instructions, see Monitoring an external key store.

Monitoring with Amazon EventBridge

You can use Amazon EventBridge (formerly Amazon CloudWatch Events) to alert you to the following important events in the lifecycle of your KMS keys.

- The key material in a KMS key was automatically rotated.
- The imported key material in a KMS key expired.
- A KMS key that had been scheduled for deletion was deleted.

AWS KMS integrates with Amazon EventBridge to notify you of important events that affect your KMS keys. Each event is represented in JSON (JavaScript Object Notation) and includes the event name, the date and time when the event occurred, and the affected. You can collect these events and establish rules that route them to one or more *targets* such as AWS Lambda functions, Amazon SNS topics, Amazon SQS queues, streams in Amazon Kinesis Data Streams, or built-in targets.

For more information about using EventBridge with other kinds of events, including those emitted by AWS CloudTrail when it records a read/write API request, see the Amazon EventBridge User Guide.

The following topics describe the EventBridge events that AWS KMS generates.

KMS CMK Rotation

AWS KMS supports automatic rotation of the key material in symmetric encryption KMS keys. Annual key material rotation is optional for customer managed keys. The key material for AWS managed keys is automatically rotated every year.

Whenever AWS KMS rotates key material, it sends a `KMS CMK Rotation` event to EventBridge. AWS KMS generates this event on a best-effort basis.

The following is an example of this event.

```
{
  "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "KMS CMK Rotation",
  "source": "aws.kms",
  "account": "111122223333",
  "time": "2022-08-10T16:37:50Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  ],
  "detail": {
    "key-id": "1234abcd-12ab-34cd-56ef-1234567890ab"
  }
}
```

KMS Imported Key Material Expiration

When you import key material into a KMS key, you can optionally specify a time at which the key material expires. When the key material expires, AWS KMS deletes the key material and sends a corresponding `KMS Imported Key Material Expiration` event to EventBridge. AWS KMS generates this event on a best-effort basis.

The following is an example of this event.

```
{
  "version": "0",
  "id": "9da9af57-9253-4406-87cb-7cc400e43465",
  "detail-type": "KMS Imported Key Material Expiration",
  "source": "aws.kms",
  "account": "111122223333",
  "time": "2022-08-10T16:37:50Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  ],
  "detail": {
    "key-id": "1234abcd-12ab-34cd-56ef-1234567890ab"
  }
}
```

KMS CMK Deletion

When you schedule deletion of a KMS key, AWS KMS enforces a waiting period before deleting the KMS key. After the waiting period ends, AWS KMS deletes the KMS key and sends a `KMS CMK Deletion` event to EventBridge. AWS KMS guarantees this EventBridge event. Due to retries, it might generate multiple events within a few seconds that delete the same KMS key.

The following is an example of this event.

```
{
  "version": "0",
  "id": "e9ce3425-7d22-412a-a699-e7a5fc3fbc9a",
  "detail-type": "KMS CMK Deletion",
  "source": "aws.kms",
  "account": "111122223333",
  "time": "2022-08-10T16:37:50Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  ],
  "detail": {
    "key-id": "1234abcd-12ab-34cd-56ef-1234567890ab"
  }
}
```

Creating AWS KMS resources with AWS CloudFormation

AWS Key Management Service is integrated with AWS CloudFormation, a service that helps you to model and set up your AWS resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes KMS keys and aliases, and AWS

CloudFormation provisions and configures those resources for you. For information about AWS KMS support for CloudFormation, see the KMS resource type reference in the *AWS CloudFormation User Guide*.

When you use AWS CloudFormation, you can reuse your template to set up your AWS KMS resources consistently and repeatedly. Describe your resources once, and then provision the same resources over and over in multiple AWS accounts and Regions.

To provision and configure resources for AWS KMS and other AWS services, you must understand AWS CloudFormation templates. Templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your AWS CloudFormation stacks. If you're unfamiliar with JSON or YAML, you can use AWS CloudFormation Designer to help you get started with AWS CloudFormation templates. For more information, see What is AWS CloudFormation Designer? in the *AWS CloudFormation User Guide*.

Regions

AWS KMS CloudFormation resources are supported in all Regions in which AWS CloudFormation is supported.

AWS KMS resources in AWS CloudFormation templates

AWS KMS supports the following AWS CloudFormation resources.

- The `AWS::KMS::Key` resource specifies a KMS key in AWS Key Management Service. You can use this resource to create symmetric encryption KMS keys, asymmetric KMS keys for encryption or signing, and symmetric HMAC KMS keys. You can use `AWS::KMS::Key` to create multi-Region primary keys of all supported types. To replicate a multi-Region key, use the `AWS::KMS::ReplicaKey` resource.
- `AWS::KMS::Alias` creates an alias and associates it with a KMS key. The KMS key can be defined in the template, or created by another mechanism.
- `AWS::KMS::ReplicaKey` creates a multi-Region replica key. To create a multi-Region primary key, use the `AWS::KMS::Key` resource. You cannot use this resource to replicate multi-Region keys with imported key material. For details about multi-Region keys, see Multi-Region keys in AWS KMS.

⚠ **Important**

If you change the value of the `KeyUsage`, `KeySpec`, or `MultiRegion` property of an existing KMS key, the existing KMS key is scheduled for deletion and a new KMS key is created with the specified value.

While scheduled for deletion, the existing KMS key becomes unusable. If you don't cancel the scheduled deletion of the existing KMS key outside of AWS CloudFormation, all data encrypted under the existing KMS key becomes unrecoverable when the KMS key is deleted.

The KMS keys that the template creates are actual resources in your AWS account. Authorized principals can use and manage the KMS keys that the template creates, either by using the template, the AWS KMS console, or the AWS KMS APIs. When you delete a KMS key from your template, the KMS key is scheduled for deletion using a waiting period that you specify in advance.

For example, you can use an AWS CloudFormation template to create a test KMS key with a key policy, key spec, key usage, aliases, and tags you prefer. You can run it through your test suite, review your results, and then use the template to schedule the test key for deletion. Later, you can run the template again to create a test key with the same properties.

Or you can use an AWS CloudFormation template to define a particular KMS key configuration that satisfies your business rules and security standards. Then you can use that template any time you need to create a KMS key. You don't have to worry about misconfigured keys. If your preferred configuration changes, you can use your template to update your KMS keys. For example, the template makes it easy to programmatically enable automatic key rotation on all KMS keys that the template defines.

For more information about AWS KMS resources, including examples, see the KMS resource type reference in the *AWS CloudFormation User Guide*.

Learn more about AWS CloudFormation

To learn more about AWS CloudFormation, see the following resources:

- AWS CloudFormation
- AWS CloudFormation User Guide
- AWS CloudFormation API Reference

- AWS CloudFormation Command Line Interface User Guide

Deleting AWS KMS keys

Deleting an AWS KMS key is destructive and potentially dangerous. It deletes the key material and all metadata associated with the KMS key and is irreversible. After a KMS key is deleted, you can no longer decrypt the data that was encrypted under that KMS key, which means that data becomes unrecoverable. (The only exceptions are multi-Region replica keys and asymmetric and HMAC KMS keys with imported key material.) This risk is significant for asymmetric KMS keys used for encryption where, without warning or error, users can continue to generate ciphertexts with the public key that cannot be decrypted after the private key is deleted from AWS KMS.

You should delete a KMS key only when you are sure that you don't need to use it anymore. If you are not sure, consider disabling the KMS key instead of deleting it. You can re-enable a disabled KMS key and cancel the scheduled deletion of a KMS key, but you cannot recover a deleted KMS key.

You can only schedule the deletion of a customer managed key. You cannot delete AWS managed keys or AWS owned keys.

Before deleting a KMS key, you might want to know how many ciphertexts were encrypted under that KMS key. AWS KMS does not store this information and does not store any of the ciphertexts. To get this information, you must determine past usage of a KMS key. For help, go to Determining past usage of a KMS key.

AWS KMS never deletes your KMS keys unless you explicitly schedule them for deletion and the mandatory waiting period expires.

However, you might choose to delete a KMS key for one or more of the following reasons:

- To complete the key lifecycle for KMS keys that you no longer need
- To avoid the management overhead and costs associated with maintaining unused KMS keys
- To reduce the number of KMS keys that count against your KMS key resource quota

Note

If you close your AWS account, your KMS keys become inaccessible and you are no longer billed for them.

AWS KMS records an entry in your AWS CloudTrail log when you schedule deletion of the KMS key and when the KMS key is actually deleted.

For information about deleting multi-Region primary and replica keys, see Deleting multi-Region keys.

Topics

About the waiting period

Because it is destructive and potentially dangerous to delete a KMS key, AWS KMS requires you to set a waiting period of 7 – 30 days. The default waiting period is 30 days.

However, the actual waiting period might be up to 24 hours longer than the one you scheduled. To get the actual date and time when the KMS key will be deleted, use the DescribeKey operation. Or in the AWS KMS console, on detail page for the KMS key, in the **General configuration** section, see the **Scheduled deletion date**. Be sure to note the time zone.

During the waiting period, the KMS key status and key state is **Pending deletion**.

- A KMS key pending deletion cannot be used in any cryptographic operations.
- AWS KMS does not rotate the key material of KMS keys that are pending deletion.

After the waiting period ends, AWS KMS deletes the KMS key, its aliases, and all related AWS KMS metadata.

Scheduling the deletion of a KMS key might not immediately affect data keys encrypted by the KMS key. For details, see How unusable KMS keys affect data keys.

Use the waiting period to ensure that you don't need the KMS key now or in the future. You can configure an Amazon CloudWatch alarm to warn you if a person or application attempts to use the KMS key during the waiting period. To recover the KMS key, you can cancel key deletion before the waiting period ends. After the waiting period ends you cannot cancel key deletion, and AWS KMS deletes the KMS key.

Deleting asymmetric KMS keys

Users who are authorized can delete symmetric or asymmetric KMS keys. The procedure to schedule the deletion of these KMS keys is the same for both types of keys. However, because the public key of an asymmetric KMS key can be downloaded and used outside of AWS KMS, the operation poses significant additional risks, especially for asymmetric KMS keys used for encryption (the key usage is ENCRYPT_DECRYPT).

- When you schedule the deletion of a KMS key, the key state of KMS key changes to **Pending deletion**, and the KMS key cannot be used in cryptographic operations. However, scheduling deletion has no effect on public keys outside of AWS KMS. Users who have the public key can continue to use them to encrypt messages. They do not receive any notification that the key state is changed. Unless the deletion is canceled, ciphertext created with the public key cannot be decrypted.
- Alarms, logs, and other strategies that detect attempted use of KMS key that is pending deletion cannot detect use of the public key outside of AWS KMS.
- When the KMS key is deleted, all AWS KMS actions involving that KMS key fail. However, users who have the public key can continue to use them to encrypt messages. These ciphertexts cannot be decrypted.

If you must delete an asymmetric KMS key with a key usage of ENCRYPT_DECRYPT, use your CloudTrail Log entries to determine whether the public key has been downloaded and shared. If it has, verify that the public key is not being used outside of AWS KMS. Then, consider disabling the KMS key instead of deleting it.

The risk posed by deleting an asymmetric KMS key is mitigated for asymmetric KMS keys with imported key material. For details, see Deleting a KMS key with imported key material.

Deleting multi-Region keys

Users who are authorized can schedule the deletion of multi-Region primary and replica keys. However, AWS KMS will not delete a multi-Region primary key that has replica keys. Also, as long as its primary key exists, you can recreate a deleted multi-Region replica key. For details, see Deleting multi-Region keys.

Deleting KMS keys with imported key material

Authorized users can schedule the deletion of KMS keys with imported key material. This action permanently deletes the KMS key, its key material, and all metadata associated with the KMS key.

You cannot create a new symmetric encryption KMS key that can decrypt the ciphertexts of a deleted symmetric encryption key with imported key material, even if you have a copy of its key material. However, if you have the key material, you can effectively recreate an asymmetric KMS key or HMAC KMS key with imported key material. For details, see Deleting a KMS key with imported key material.

Controlling access to key deletion

If you use IAM policies to allow AWS KMS permissions, IAM identities that have AWS administrator access (`"Action": "*"`) or AWS KMS full access (`"Action": "kms:*"`) are already allowed to schedule and cancel key the deletion of KMS keys. To allow key administrators to schedule and cancel key deletion in the key policy, use the AWS KMS console or the AWS KMS API.

Typically, only key administrators have permission to schedule or cancel key deletion. However, you can give these permissions to other IAM identities by adding the `kms:ScheduleKeyDeletion` and `kms:CancelKeyDeletion` permission to the key policy or an IAM policy. You can also use the `kms:ScheduleKeyDeletionPendingWindowInDays` condition key to further constrain the values that principals can specify in the `PendingWindowInDays` parameter of a ScheduleKeyDeletion request.

Allow key administrators to schedule and cancel key deletion (console)

To give key administrators permission to schedule and cancel key deletion.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Choose the alias or key ID of the KMS key whose permissions you want to change.
5. Choose the **key policy** tab.
6. The next step differs for the *default view* and *policy view* of your key policy. Default view is available only if you are using the default console key policy. Otherwise, only policy view is available.

   When default view is available, a **Switch to policy view** or **Switch to default view** button appears on the **Key policy** tab.

   - In default view:

     - Under **Key deletion**, choose **Allow key administrators to delete this key**.
   - In policy view:

     a. Choose **Edit**.

     b. In the policy statement for key administrators, add the `kms:ScheduleKeyDeletion` and `kms:CancelKeyDeletion` permissions to the `Action` element.

     ```
     {
       "Sid": "Allow access for Key Administrators",
       "Effect": "Allow",
       "Principal": {"AWS": "arn:aws:iam::111122223333:user/KMSKeyAdmin"},
       "Action": [
         "kms:Create*",
         "kms:Describe*",
         "kms:Enable*",
         "kms:List*",
         "kms:Put*",
         "kms:Update*",
         "kms:Revoke*",
         "kms:Disable*",
         "kms:Get*",
         "kms:Delete*",
         "kms:ScheduleKeyDeletion",
         "kms:CancelKeyDeletion"
       ],
     ```

```
    "Resource": "*"
  }
```

c. Choose **Save changes**.

Allow key administrators permission to schedule and cancel key deletion (AWS CLI)

You can use the AWS Command Line Interface to add permissions for scheduling and canceling key deletion.

To add permission to schedule and cancel key deletion

1. Use the `aws kms get-key-policy` command to retrieve the existing key policy, and then save the policy document to a file.
2. Open the policy document in your preferred text editor. In the policy statement for key administrators, add the `kms:ScheduleKeyDeletion` and `kms:CancelKeyDeletion` permissions. The following example shows a policy statement with these two permissions:

```
{
  "Sid": "Allow access for Key Administrators",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::111122223333:user/KMSKeyAdmin"},
  "Action": [
    "kms:Create*",
    "kms:Describe*",
    "kms:Enable*",
    "kms:List*",
    "kms:Put*",
    "kms:Update*",
    "kms:Revoke*",
    "kms:Disable*",
    "kms:Get*",
    "kms:Delete*",
    "kms:ScheduleKeyDeletion",
    "kms:CancelKeyDeletion"
  ],
  "Resource": "*"
}
```

3. Use the `aws kms put-key-policy` command to apply the key policy to the KMS key.

Scheduling and canceling key deletion

The following procedures describe how to schedule key deletion and cancel key deletion of single-Region AWS KMS keys (KMS keys) in AWS KMS using the AWS Management Console, the AWS CLI, and the AWS SDK for Java.

For information about scheduling the deletion of multi-Region keys, see Deleting multi-Region keys.

> ⚠ **Warning**
>
> Deleting a KMS key is destructive and potentially dangerous. You should proceed only when you are sure that you don't need to use the KMS key anymore and won't need to use it in the future. If you are not sure, you should disable the KMS key instead of deleting it.

Before you can delete a KMS key, you must have permission to do so. For information about giving these permissions to key administrators, see Controlling access to key deletion. You can also use the `kms:ScheduleKeyDeletionPendingWindowInDays` condition key to further constrain the waiting period, such as enforcing a minimum waiting period.

AWS KMS records an entry in your AWS CloudTrail log when you schedule deletion of the KMS key and when the KMS key is actually deleted.

Scheduling and canceling key deletion (console)

In the AWS Management Console, you can schedule and cancel the deletion of multiple KMS keys at one time.

To schedule key deletion

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.

   You cannot schedule the deletion of AWS managed keys or AWS owned keys.
4. Choose the check box next to the KMS key that you want to delete.

5. Choose **Key actions**, **Schedule key deletion**.
6. Read and consider the warning, and the information about canceling the deletion during the waiting period. If you decide to cancel the deletion, at the bottom of the page, choose **Cancel**.
7. For **Waiting period (in days)**, enter a number of days between 7 and 30.
8. Review the KMS keys that you are deleting.
9. Choose the check box next to **Confirm you want to schedule this key for deletion in *<number of days>* days.**.
10. Choose **Schedule deletion**.

The KMS key status changes to **Pending deletion**.

To cancel key deletion

1. Open the AWS KMS console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Choose the check box next to the KMS key that you want to recover.
5. Choose **Key actions**, **Cancel key deletion**.

The KMS key status changes from **Pending deletion** to **Disabled**. To use the KMS key, you must enable it.

Scheduling and canceling key deletion (AWS CLI)

Use the `aws kms schedule-key-deletion` command to schedule key deletion of a customer managed key, as shown in the following example.

You cannot schedule the deletion of an AWS managed key or AWS owned key.

```
$ aws kms schedule-key-deletion --key-id 1234abcd-12ab-34cd-56ef-1234567890ab --
pending-window-in-days 10
```

When used successfully, the AWS CLI returns output like the output shown in the following example:

```
{
    "KeyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "DeletionDate": 1598304792.0,
    "KeyState": "PendingDeletion",
    "PendingWindowInDays": 10
}
```

Use the aws kms cancel-key-deletion command to cancel key deletion from the AWS CLI as shown in the following example.

```
$ aws kms cancel-key-deletion --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
```

When used successfully, the AWS CLI returns output like the output shown in the following example:

```
{
    "KeyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
}
```

The status of the KMS key changes from **Pending Deletion** to **Disabled**. To use the KMS key, you must enable it.

Scheduling and canceling key deletion (AWS SDK for Java)

The following example demonstrates how to schedule the deletion of a customer managed key with the AWS SDK for Java. This example requires that you previously instantiated an AWSKMSClient as kms.

```
String KeyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

int PendingWindowInDays = 10;

ScheduleKeyDeletionRequest scheduleKeyDeletionRequest =
new
 ScheduleKeyDeletionRequest().withKeyId(KeyId).withPendingWindowInDays(PendingWindowInDays);
kms.scheduleKeyDeletion(scheduleKeyDeletionRequest);
```

The following example demonstrates how to cancel key deletion with the AWS SDK for Java. This example requires that you previously instantiated an AWSKMSClient as kms.

```
String KeyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

CancelKeyDeletionRequest cancelKeyDeletionRequest =
new CancelKeyDeletionRequest().withKeyId(KeyId);
kms.cancelKeyDeletion(cancelKeyDeletionRequest);
```

The status of the KMS key changes from **Pending Deletion** to **Disabled**. To use the KMS key, you must enable it.

Creating an alarm that detects use of a KMS key pending deletion

You can combine the features of AWS CloudTrail, Amazon CloudWatch Logs, and Amazon Simple Notification Service (Amazon SNS) to create an Amazon CloudWatch alarm that notifies you when someone in your account tries to use a KMS key that is pending deletion. If you receive this notification, you might want to cancel deletion of the KMS key and reconsider your decision to delete it.

The following procedures create an alarm that notifies you whenever the "*Key ARN* is pending deletion" error message is written to your CloudTrail log files. This error message indicates that a person or application tried to use the KMS key in a cryptographic operation. Because the notification is linked to the error message, it is not triggered when you use API operations that are permitted on KMS keys that are pending deletion, such as `ListKeys`, `CancelKeyDeletion`, and `PutKeyPolicy`. To see a list of the AWS KMS API operations that return this error message, see Key states of AWS KMS keys.

The notification email that you receive does not list the KMS key or the cryptographic operation. You can find that information in your CloudTrail log. Instead, the email reports that the alarm state changed from **OK** to **Alarm**. For more information about CloudWatch alarms and state changes, see Using Amazon CloudWatch alarms in the *Amazon CloudWatch User Guide*.

⚠ **Warning**

This Amazon CloudWatch alarm cannot detect use of the public key of an asymmetric KMS key outside of AWS KMS. For details about the special risks of deleting asymmetric KMS keys used for public key cryptography, including creating ciphertexts that cannot be decrypted, see Deleting asymmetric KMS keys.

Topics

Requirements for a CloudWatch alarm

Before you create a CloudWatch alarm, you must create an AWS CloudTrail trail and configure CloudTrail to deliver CloudTrail log files to Amazon CloudWatch Logs. You also need an Amazon SNS topic for the alarm notification.

- Create a CloudTrail trail.

  CloudTrail is automatically enabled on your AWS account when you create the account. However, for an ongoing record of events in your account, including events for AWS KMS, create a trail.

- Configure CloudTrail to deliver your log files CloudWatch Logs.

  Configure delivery of your CloudTrail log files to CloudWatch Logs. This allows CloudWatch Logs to monitor the logs for AWS KMS API requests that attempt to use a KMS key that is pending deletion.

- Create an Amazon SNS topic.

  When your alarm triggers, it notifies you by sending an email message to an email address in an Amazon Simple Notification Service (Amazon SNS) topic.

Creating the CloudWatch alarm

In this procedure, you create a CloudWatch log group metric filter that finds instances of the pending deletion exception. Then, you create a CloudWatch alarm based on the log group metric. For information about log group metric filters, see Creating metrics from log events using filters in the Amazon CloudWatch Logs User Guide.

1. Create a CloudWatch metric filter that parses CloudTrail logs.

   Follow the instructions in Create a metric filter for a log group using the following required values. For other fields, accept the default values and provide names as requested.

Field	Value
Filter pattern	`{ $.eventSource = kms* && $.errorMessage = "* is pending deletion."}`
Metric value	1

2. Create a CloudWatch alarm based on the metric filter that you created in Step 1.

   Follow the instructions in Creating a CloudWatch alarm based on a log group-metric filter using the following required values. For other fields, accept the default values and provide names as requested.

Field	Value
Metric filter	The name of the metric filter that you created in Step 1.
Threshold type	Static
Conditions	**Whenever** *`metric-name`* is **Greater** than 1
Datapoints to alarm	1 out of 1
Missing data treatment	**Treat missing data as good (not breaching threshold)**

After you complete this procedure, you will receive a notification each time your new CloudWatch alarm enters the `ALARM` state. If you receive a notification for this alarm, it might mean that a KMS key that is scheduled for deletion is still needed to encrypt or decrypt data. In that case, cancel deletion of the KMS key and reconsider your decision to delete it.

Determining past usage of a KMS key

Before deleting a KMS key, you might want to know how many ciphertexts were encrypted under that key. AWS KMS does not store this information, and does not store any of the ciphertexts. Knowing how a KMS key was used in the past might help you decide whether or not you will need it in the future. This topic suggest several strategies that can help you determine the past usage of a KMS key.

> ⚠ **Warning**
>
> These strategies for determining past and actual usage are effective only for AWS users and AWS KMS operations. They cannot detect use of the public key of an asymmetric KMS key outside of AWS KMS. For details about the special risks of deleting asymmetric KMS keys used for public key cryptography, including creating ciphertexts that cannot be decrypted, see Deleting asymmetric KMS keys.

Topics

Examining KMS key permissions to determine the scope of potential usage

Determining who or what currently has access to a KMS key might help you determine how widely the KMS key was used and whether it is still needed. To learn how to determine who or what currently has access to a KMS key, go to Determining access to AWS KMS keys.

Examining AWS CloudTrail logs to determine actual usage

You might be able to use a KMS key usage history to help you determine whether you have ciphertexts encrypted under a particular KMS key.

All AWS KMS API activity is recorded in AWS CloudTrail log files. If you have created a CloudTrail trail in the region where your KMS key is located, you can examine your CloudTrail log files to view a history of all AWS KMS API activity for a particular KMS key. If you don't have a trail, you can still view recent events in your CloudTrail event history. For details about how AWS KMS uses CloudTrail, see Logging AWS KMS API calls with AWS CloudTrail.

The following examples show CloudTrail log entries that are generated when a KMS key is used to protect an object stored in Amazon Simple Storage Service (Amazon S3). In this example, the object is uploaded to Amazon S3 using Protecting data using server-side encryption with KMS keys (SSE-KMS). When you upload an object to Amazon S3 with SSE-KMS, you specify the KMS key to use for protecting the object. Amazon S3 uses the AWS KMS GenerateDataKey operation to request a unique data key for the object, and this request event is logged in CloudTrail with an entry similar to the following:

```
{
  "eventVersion": "1.02",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROACKCEVSQ6C2EXAMPLE:example-user",
    "arn": "arn:aws:sts::111122223333:assumed-role/Admins/example-user",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2015-09-10T23:12:48Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::111122223333:role/Admins",
        "accountId": "111122223333",
        "userName": "Admins"
      }
    },
    "invokedBy": "internal.amazonaws.com"
  },
  "eventTime": "2015-09-10T23:58:18Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "GenerateDataKey",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "internal.amazonaws.com",
  "userAgent": "internal.amazonaws.com",
  "requestParameters": {
    "encryptionContext": {"aws:s3:arn": "arn:aws:s3:::example_bucket/example_object"},
    "keySpec": "AES_256",
    "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  },
  "responseElements": null,
  "requestID": "cea04450-5817-11e5-85aa-97ce46071236",
  "eventID": "80721262-21a5-49b9-8b63-28740e7ce9c9",
  "readOnly": true,
  "resources": [{
    "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "accountId": "111122223333"
  }],
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

When you later download this object from Amazon S3, Amazon S3 sends a `Decrypt` request to AWS KMS to decrypt the object's data key using the specified KMS key. When you do this, your CloudTrail log files include an entry similar to the following:

```
{
  "eventVersion": "1.02",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROACKCEVSQ6C2EXAMPLE:example-user",
    "arn": "arn:aws:sts::111122223333:assumed-role/Admins/example-user",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2015-09-10T23:12:48Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::111122223333:role/Admins",
        "accountId": "111122223333",
        "userName": "Admins"
      }
    },
    "invokedBy": "internal.amazonaws.com"
  },
  "eventTime": "2015-09-10T23:58:39Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "Decrypt",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "internal.amazonaws.com",
  "userAgent": "internal.amazonaws.com",
  "requestParameters": {
    "encryptionContext": {"aws:s3:arn": "arn:aws:s3:::example_bucket/example_object"}},
  "responseElements": null,
  "requestID": "db750745-5817-11e5-93a6-5b87e27d91a0",
  "eventID": "ae551b19-8a09-4cfc-a249-205ddba330e3",
  "readOnly": true,
  "resources": [{
    "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "accountId": "111122223333"
  }],
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

All AWS KMS API activity is logged by CloudTrail. By evaluating these log entries, you might be able to determine the past usage of a particular KMS key, and this might help you determine whether or not you want to delete it.

To see more examples of how AWS KMS API activity appears in your CloudTrail log files, go to Logging AWS KMS API calls with AWS CloudTrail. For more information about CloudTrail go to the AWS CloudTrail User Guide.

Key states of AWS KMS keys

An AWS KMS key always has a key state. Operations on the KMS key and its environment can change that key state, either transiently, or until another operation changes its key state.

The table in this section shows how key states affect calls to AWS KMS API operations. As a result of its key state, an operation on a KMS key is expected to succeed (#), fail (X), or succeed only under certain conditions (?). The result often differs for KMS keys with imported key material.

This table includes only the API operations that use an existing KMS key. Other operations, such as CreateKey and ListKeys, are omitted.

Topics

- Key states and KMS key types
- Key state table

Key states and KMS key types

The type of the KMS key determines the key states it can have.

- All KMS keys can be in the Enabled, Disabled, and PendingDeletion states.
- Most KMS keys are created in the Enabled state. Keys with imported key material are created in the PendingImport state.
- The PendingImport state applies only to KMS keys with imported key material.
- The Unavailable state applies only to a KMS key in a custom key store. A KMS key in an AWS CloudHSM key store is Unavailable when the custom key store is intentionally disconnected from its AWS CloudHSM cluster. A KMS key in an external key store is Unavailable when the custom key store is intentionally disconnected from its external key store proxy. You can view and manage unavailable KMS keys, but you cannot use them in cryptographic operations.

  The key state of a KMS key in a custom key store is not affected by changes to its backing key. A KMS key in a AWS CloudHSM key store is not affected by changes to its associated key material in the AWS CloudHSM cluster. A KMS key in an external key store is not affected by changes to its external key in an external key manager. If the backing key is disabled or deleted, the KMS key state doesn't change, but cryptographic operations using the KMS key fail.
- The Creating, Updating, and PendingReplicaDeletion key states apply only to multi-Region keys.
  - A multi-Region replica key is in the transient Creating key state while it is being created. This process might still be in progress when the ReplicateKey operation completes. When the replicate process completes, the replica key is in the Enabled or PendingImport state.
  - Multi-Region keys are in the transient Updating key state while the primary Region is being updated. This process might still be in progress when the UpdatePrimaryRegion operation completes. When the update process completes, the primary and replica keys resume the Enabled key state.
  - When you schedule deletion of a multi-Region primary key that has replica keys, the primary key is in the PendingReplicaDeletion state until all of its replica keys are deleted. Then its key state changes to PendingDeletion. For details, see Deleting multi-Region keys.

Key state table

The following table shows how the key state of a KMS key affects AWS KMS operations.

The descriptions of the numbered footnotes ([*n*]) are at the end of this topic.

Note

You might need to scroll horizontally or vertically to see all of the data in this table.

API	Enabled	Disabled	Pending deletion Pending replica deletion	Pending import	Unavailable	Creating	Updating
CancelKeyDeletion	🚫 [4]	🚫 [4]	✓	🚫 [4]	🚫 [4], [13]	🚫 [4]	🚫 [4]
CreateAlias	✓	✓	🚫 [3]	✓	✓	✓	✓
CreateGrant	✓	🚫 [1]	🚫 [2] or [3]	🚫 [5]	✓	🚫 [14]	✓
Decrypt	✓	🚫 [1]	🚫 [2] or [3]	🚫 [5]	🚫 [11]	🚫 [14]	✓
DeleteAlias	✓	✓	✓	✓	✓	✓	✓
DeleteImportedKeyMaterial	✓ [9]	✓ [9]	✓ [9]	✓	N/A	🚫 [14]	🚫 [15]

API	Enabled	Disabled	Pending deletion Pending replica deletion	Pending import	Unavailable	Creating	Updating
				(No effect)			
DescribeKey	✓	✓	✓	✓	✓	✓	✓
DisableKey	✓	✓	✗ [3]	✗ [5]	✓ [12]	✗ [14]	✗ [15]
DisableKeyRotation	? [7]	✗ [1] or [7]	✗ [3] or [7]	✗ [6]	✗ [7]	✗ [14]	? [7]
EnableKey	✓	✓	✗ [3]	✗ [5]	✓ [12]	✗ [14]	✗ [15]
EnableKeyRotation	? [7]	✗ [1] or [7]	✗ [3] or [7]	✗ [6]	✗ [7]	✗ [14]	? [7]
Encrypt	✓	✗ [1]	✗ [2] or [3]	✗ [5]	✗ [11]	✗ [14]	✓
GenerateDataKey	✓	✗ [1]	✗ [2] or [3]	✗ [5]	✗ [11]	✗ [14]	✓

API	Enabled	Disabled	Pending deletion Pending replica deletion	Pending import	Unavailable	Creating	Updating
GenerateDataKeyPair	✓	⊗ [1]	⊗ [2] or [3]	⊗ [5]	⊗ [11]	⊗ [14]	✓
GenerateDataKeyPairWithoutPlaintext	✓	⊗ [1]	⊗ [2] or [3]	⊗ [5]	⊗ [11]	⊗ [14]	✓
GenerateDataKeyWithoutPlaintext	✓	⊗ [1]	⊗ [2] or [3]	⊗ [5]	⊗ [11]	⊗ [14]	✓
GenerateMac	✓	⊗ [1]	⊗ [2] or [3]	N/A	N/A	⊗ [14]	✓
GetKeyPolicy	✓	✓	✓	✓	✓	✓	✓
GetKeyRotationStatus	? [7]	? [7]	? [7]	⊗ [6]	⊗ [7]	? [7]	? [7]
GetParametersForImport	? [9]	? [9]	⊗ [8] or [9]	✓	⊗ [9]	⊗ [14]	⊗ [15]

API	Enabled	Disabled	Pending deletion Pending replica deletion	Pending import	Unavailable	Creating	Updating
GetPublicKey	✓	✓	🚫 [2] or [3]	N/A	N/A	🚫 [14]	✓
ImportKeyMaterial	❓ [9]	❓ [9]	🚫 [8] or [9]	✓	🚫 [9]	🚫 [14]	✓
ListAliases	✓	✓	✓	✓	✓	✓	✓
ListGrants	✓	✓	✓	✓	✓	✓	✓
ListKeyPolicies	✓	✓	✓	✓	✓	✓	✓
ListKeyRotations	❓ [7]	❓ [7]	❓ [7]	🚫 [6]	🚫 [7]	❓ [7]	❓ [7]
ListResourceTags	✓	✓	✓	✓	✓	✓	✓
PutKeyPolicy	✓	✓	✓	✓	✓	✓	✓
ReEncrypt	✓	🚫 [1]	🚫 [2] or [3]	🚫 [5]	🚫 [11]	🚫 [14]	✓

API	Enabled	Disabled	Pending deletion Pending replica deletion	Pending import	Unavailable	Creating	Updating
ReplicateKey	✓	🚫 [1]	🚫 [2] or [3]	🚫 [5]	N/A	🚫 [14]	🚫 [15]
RetireGrant	✓	✓	✓	✓	✓	✓	✓
RevokeGrant	✓	✓	✓	✓	✓	✓	✓
RotateKeyOnDemanc	? [7]	🚫 [1] or [7]	🚫 [3] or [7]	🚫 [6]	🚫 [7]	🚫 [14]	? [7]
ScheduleKeyDeletion	✓	✓	🚫 [3]	✓	✓	✓	🚫 [15]
Sign	✓	🚫 [1]	🚫 [2] or [3]	N/A	N/A	🚫 [14]	✓
TagResource	✓	✓	🚫 [3]	✓	✓	✓	✓
UntagResource	✓	✓	🚫 [3]	✓	✓	✓	✓

API	Enabled	Disabled	Pending deletion Pending replica deletion	Pending import	Unavailable	Creating	Updating
UpdateAlias	✓	✓	? [10]	✓	✓	✓	✓
UpdateKeyDescription	✓	✓	🚫 [3]	✓	✓	✓	✓
UpdatePrimaryRegion	✓	🚫 [1]	🚫 [2] or [3]	🚫 [5]	N/A	🚫 [14]	✓
Verify	✓	🚫 [1]	🚫 [2] or [3]	N/A	N/A	🚫 [14]	✓
VerifyMac	✓	🚫 [1]	🚫 [2] or [3]	N/A	N/A	🚫 [14]	✓

Table Details

- [1] DisabledException: *<key ARN>* is disabled.
- [2] DisabledException: *<key ARN>* is pending deletion (or pending replica deletion).
- [3] KMSInvalidStateException: *<key ARN>* is pending deletion (or pending replica deletion).

- [4] `KMSInvalidStateException: <key ARN> is not pending deletion (or pending replica deletion).`
- [5] `KMSInvalidStateException: <key ARN> is pending import.`
- [6] `UnsupportedOperationException: <key ARN> origin is EXTERNAL which is not valid for this operation.`
- [7] If the KMS key has imported key material or is in a custom key store: `UnsupportedOperationException`.
- [8] If the KMS key has imported key material: `KMSInvalidStateException`
- [9] If the KMS key cannot or does not have imported key material: `UnsupportedOperationException`.
- [10] If the source KMS key is pending deletion, the command succeeds. If the destination KMS key is pending deletion, the command fails with error: `KMSInvalidStateException : <key ARN> is pending deletion.`
- [11] `KMSInvalidStateException: <key ARN> is unavailable.` You cannot perform this operation on an unavailable KMS key.
- [12] The operation succeeds, but the key state of the KMS key does not change until it becomes available.
- [13] While a KMS key in a custom key store is pending deletion, its key state remains `PendingDeletion` even if the KMS key becomes unavailable. This allows you to cancel deletion of the KMS key at any time during the waiting period.
- [14] `KMSInvalidStateException: <key ARN> is creating.` AWS KMS throws this exception while it is replicating a multi-Region key (`ReplicateKey`).
- [15] `KMSInvalidStateException: <key ARN> is updating.` AWS KMS throws this exception while it is updating the primary Region of a multi-Region key (`UpdatePrimaryRegion`).

Authentication and access control for AWS KMS

To use AWS KMS, you must have credentials that AWS can use to authenticate your requests. The credentials must include permissions to access AWS resources: AWS KMS keys and aliases. No AWS principal has any permissions to a KMS key unless that permission is provided explicitly and never denied. There are no implicit or automatic permission to use or manage a KMS key.

The primary way to manage access to your AWS KMS resources is with *policies*. Policies are documents that describe which principals can access which resources. Policies attached to an IAM identity are called *identity-based policies* (or *IAM policies*), and policies attached to other kinds of resources are called *resource policies*. AWS KMS resource policies for KMS keys are called *key policies*. All KMS keys have a key policy.

To control access to your AWS KMS aliases, use IAM policies. To allow principals to create aliases, you must provide the permission to the alias in an IAM policy and permission to the key in a key policy. For details, see Controlling access to aliases.

To control access to your KMS keys, you can use the following policy mechanisms.

- **Key policy** – Every KMS key has a key policy. It is the primary mechanism for controlling access to a KMS key. You can use the key policy alone to control access, which means the full scope of access to the KMS key is defined in a single document (the key policy). For more information about using key policies, see Key policies.
- **IAM policies** – You can use IAM policies in combination with the key policy and grants to control access to a KMS key. Controlling access this way enables you to manage all of the permissions for your IAM identities in IAM. To use an IAM policy to allow access to a KMS key, the key policy must explicitly allow it. For more information about using IAM policies, see IAM policies.
- **Grants** – You can use grants in combination with the key policy and IAM policies to allow access to a KMS key. Controlling access this way enables you to allow access to the KMS key in the key policy, and to allow identities to delegate their access to others. For more information about using grants, see Grants in AWS KMS.

KMS keys belong to the AWS account in which they were created. However, no identity or principal, including the AWS account root user, has permission to use or manage a KMS key unless that permission is explicitly provided in a key policy, IAM policy or grant. The IAM identity who creates a KMS key is not considered to be the key owner and they don't automatically have permission to use or manage the KMS key that they created. Like any other identity, the key creator needs to get permission through a key policy, IAM policy, or grant. However, identities who have the `kms:CreateKey` permission can set the initial key policy and give themselves permission to use or manage the key.

The following topics provide details about how you can use AWS Identity and Access Management (IAM) and AWS KMS permissions to help secure your resources by controlling who can access them.

Topics

Concepts in AWS KMS access control

Learn the concepts used in discussions of access control in AWS KMS.

Topics

- AWS KMS resources

Authentication

Authentication is the process of verifying your identity. To send a request to AWS KMS, you must sign into AWS using your AWS credentials.

Authorization

Authorization provides the permission to send requests to create, manage, or use AWS KMS resources. For example, you must be authorized to use a KMS key in a cryptographic operation.

To control access to your AWS KMS resources, use key policies, IAM policies, and grants. Every KMS key must have a key policy. If the key policy allows it, you can also use IAM policies and grants to give principals access to the KMS key. To refine your authorization, you can use condition keys that allow or deny access only when a request or resource meets the conditions you specify. You can also allow access to principals you trust in other AWS accounts.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signing AWS API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or

store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.

  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

An AWS KMS key policy is a resource-based policy that controls access to a KMS key. Every KMS key must have a key policy. You can use other authorization mechanism to allow access to the KMS key, but only if the key policy allows it. (You can use an IAM policy to *deny* access to a KMS key even if the key policy doesn't explicitly permit it.)

Resource-based policies are JSON policy documents that you attach to a resource, such as a KMS key, to control access to the specific resource. The resource-based policy defines the actions that a specified principal can perform on that resource and under what conditions. You don't specify the resource in a resource-based policy, but you must specify a principal, such as accounts, users, roles, federated users, or AWS services. Resource-based policies are inline policies that are located in that service that manages the resource. You can't use AWS managed policies from IAM, such as the `AWSKeyManagementServicePowerUser` managed policy, in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

AWS KMS does not support ACLs.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's

permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

AWS KMS resources

In AWS KMS, the primary resource is an AWS KMS key. AWS KMS also supports an alias, an independent resource that provides a friendly name for a KMS key. Some AWS KMS operations allow you to use an alias to identify a KMS key.

Each instance of a KMS key or alias has a unique Amazon Resource Name (ARN) with a standard format. In AWS KMS resources, the AWS service name is `kms`.

- **AWS KMS key**

  ARN format:

  ```
  arn:AWS partition name:AWS service name:AWS Region:AWS account ID:key/key ID
  ```

  Example ARN:

  ```
  arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
  ```
- **Alias**

  ARN format:

  ```
  arn:AWS partition name:AWS service name:AWS Region:AWS account ID:alias/alias name
  ```

  Example ARN:

  ```
  arn:aws:kms:us-west-2:111122223333:alias/example-alias
  ```

AWS KMS provides a set of API operations to work with your AWS KMS resources. For more information about identifying KMS keys in the AWS Management Console and AWS KMS API operations, see Key identifiers (KeyId). For a list of AWS KMS operations, see the AWS Key Management Service API Reference.

Key policies in AWS KMS

A key policy is a resource policy for an AWS KMS key. Key policies are the primary way to control access to KMS keys. Every KMS key must have exactly one key policy. The statements in the key policy determine who has permission to use the KMS key and how they can use it. You can also use IAM policies and grants to control access to the KMS key, but every KMS key must have a key policy.

No AWS principal, including the account root user or key creator, has any permissions to a KMS key unless they are explicitly allowed, and never denied, in a key policy, IAM policy, or grant.

Unless the key policy explicitly allows it, you cannot use IAM policies to *allow* access to a KMS key. Without permission from the key policy, IAM policies that allow permissions have no effect. (You can use an IAM policy to *deny* a permission to a KMS key without permission from a key policy.) The default key policy enables IAM policies. To enable IAM policies in your key policy, add the policy statement described in Allows access to the AWS account and enables IAM policies.

Unlike IAM policies, which are global, key policies are Regional. A key policy controls access only to a KMS key in the same Region. It has no effect on KMS keys in other Regions.

Topics

Creating a key policy

You can create and manage key policies in the AWS KMS console, by using AWS KMS API operations, such as CreateKey, ReplicateKey, and PutKeyPolicy, or by using an AWS CloudFormation template.

When you create a KMS key in the AWS KMS console, the console walks you through the steps of creating a key policy based on the default key policy for the console. When you use the `CreateKey` or `ReplicateKey` APIs, if you don't specify a key policy, these APIs apply the default key policy for keys created programmatically. When you use the `PutKeyPolicy` API, you are required to specify a key policy.

Each policy document can have one or more policy statements. The following example shows a valid key policy document with one policy statement.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Describe the policy statement",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/Alice"
      },
      "Action": "kms:DescribeKey",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:KeySpec": "SYMMETRIC_DEFAULT"
        }
      }
    }
  ]
}
```

Topics

- Key policy format
- Elements in a key policy
- Example key policy

Key policy format

A key policy document must conform to the following rules:

- Up to 32 kilobytes (32,768 bytes)

- The Sid element in a key policy statement can include spaces. (Spaces are prohibited in the Sid element of an IAM policy document.)

A key policy document can include only the following characters:

- Printable ASCII characters
- Printable characters in the Basic Latin and Latin-1 Supplement character set
- The tab (\u0009), line feed (\u000A), and carriage return (\u000D) special characters

Elements in a key policy

A key policy document must have the following elements:

Version

Specifies the key policy document version. Set the version to 2012-10-17 (the latest version).

Statement

Encloses the policy statements. A key policy document must have at least one statement.

Each key policy statement consists of up to six elements. The Effect, Principal, Action, and Resource elements are required.

Sid

(Optional) The statement identifier (Sid) an arbitrary string you can use to describe the statement. The Sid in a key policy can include spaces. (You can't include spaces in an IAM policy Sid element.)

Effect

(Required) Determines whether to allow or deny the permissions in the policy statement. Valid values are Allow or Deny. If you don't explicitly allow access to a KMS key, access is implicitly denied. You can also explicitly deny access to a KMS key. You might do this to make sure that a user cannot access it, even when a different policy allows access.

Principal

(Required) The principal is the identity that gets the permissions specified in the policy statement. You can specify AWS accounts, IAM users, IAM roles, and some AWS services as principals in a key policy. IAM user groups are not a valid principal in any policy type.

An asterisk value, such as `"AWS": "*"` represents all AWS identities in all accounts.

> ⚠ **Important**
>
> Do not set the Principal to an asterisk (*) in any key policy statement that allows permissions unless you use conditions to limit the key policy. An asterisk gives every identity in every AWS account permission to use the KMS key, unless another policy statement explicitly denies it. Users in other AWS accounts can use your KMS key whenever they have corresponding permissions in their own account.

> ⓘ **Note**
>
> IAM best practices discourage the use of IAM users with long-term credentials. Whenever possible, use IAM roles, which provide temporary credentials. For details, see Security best practices in IAM in the *IAM User Guide*.

When the principal in a key policy statement is an *AWS account principal* expressed as `arn:aws:iam::`*`111122223333`*`:root"`, the policy statement doesn't give permission to any IAM principal. Instead, it gives the AWS account permission to use IAM policies to delegate the permissions specified in the key policy. (A principal in `arn:aws:iam::`*`111122223333`*`:root"` format does *not* represent the AWS account root user, despite the use of "root" in the account identifier. However, the account principal represents the account and its administrators, including the account root user.)

When the principal is another AWS account or its principals, the permissions are effective only when the account is enabled in the Region with the KMS key and key policy. For information about Regions that are not enabled by default ("opt-in Regions"), see Managing AWS Regions in the *AWS General Reference*.

To allow a different AWS account or its principals to use a KMS key, you must provide permission in a key policy and in an IAM policy in the other account. For details, see Allowing users in other accounts to use a KMS key.

Action

(Required) Specify the API operations to allow or deny. For example, the `kms:Encrypt` action corresponds to the AWS KMS Encrypt operation. You can list more than one action in a policy statement. For more information, see Permissions reference.

Resource

(Required) In a key policy, the value of the Resource element is `"*"`, which means "this KMS key." The asterisk (`"*"`) identifies the KMS key to which the key policy is attached.

> **Note**
>
> If the required `Resource` element is missing from a key policy statement, the policy statement has no effect. A key policy statement without a `Resource` element doesn't apply to any KMS key.
> When a key policy statement is missing its `Resource` element, the AWS KMS console correctly reports an error, but the CreateKey and PutKeyPolicy APIs succeed, even though the policy statement is ineffective.

Condition

(Optional) Conditions specify requirements that must be met for a key policy to take effect. With conditions, AWS can evaluate the context of an API request to determine whether or not the policy statement applies.

To specify conditions, you use predefined *condition keys*. AWS KMS supports AWS global condition keys and AWS KMS condition keys. To support attribute-based access control (ABAC), AWS KMS provides condition keys that control access to a KMS key based on tags and aliases. For details, see ABAC for AWS KMS.

The format for a condition is:

```
"Condition": {"condition operator": {"condition key": "condition value"}}
```

such as:

```
"Condition": {"StringEquals": {"kms:CallerAccount": "111122223333"}}
```

For more information about AWS policy syntax, see AWS IAM Policy Reference in the *IAM User Guide*.

Example key policy

The following example shows a complete key policy for a symmetric encryption KMS key. You can use it for reference as you read about the key policy concepts in this chapter. This key policy combines the example policy statements from the preceding default key policy section into a single key policy that accomplishes the following:

- Allows the example AWS account, 111122223333, full access to the KMS key. It allows the account and its administrators, including the account root user (for emergencies), to use IAM policies in the account to allow access to the KMS key.
- Allows the `ExampleAdminRole` IAM role to administer the KMS key.
- Allows the `ExampleUserRole` IAM role to use the KMS key.

```
{
    "Id": "key-consolepolicy",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow access for Key Administrators",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/ExampleAdminRole"
            },
            "Action": [
                "kms:Create*",
                "kms:Describe*",
                "kms:Enable*",
                "kms:List*",
                "kms:Put*",
                "kms:Update*",
                "kms:Revoke*",
                "kms:Disable*",
                "kms:Get*",
                "kms:Delete*",
                "kms:TagResource",
                "kms:UntagResource",
                "kms:ScheduleKeyDeletion",
                "kms:CancelKeyDeletion",
                "kms:RotateKeyOnDemand"
            ],
            "Resource": "*"
        },
        {
            "Sid": "Allow use of the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/ExampleUserRole"
            },
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        },
        {
            "Sid": "Allow attachment of persistent resources",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/ExampleUserRole"
            },
            "Action": [
                "kms:CreateGrant",
                "kms:ListGrants",
                "kms:RevokeGrant"
            ],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "kms:GrantIsForAWSResource": "true"
                }
            }
        }
    ]
}
```

Default key policy

When you create a KMS key, you can specify the key policy for the new KMS key. If you don't provide one, AWS KMS creates one for you. The default key policy that AWS KMS uses differs depending on whether you create the key in the AWS KMS console or you use the AWS KMS API.

Default key policy when you create a KMS key programmatically

When you create a KMS key programmatically with the AWS KMS API (including by using the AWS SDKs, AWS Command Line Interface or AWS Tools for PowerShell), and you don't specify a key policy, AWS KMS applies a very simple default key policy. This default key policy has one policy statement that gives the AWS account that owns the KMS key permission to use IAM policies to allow access to all AWS KMS operations on the KMS key. For more information about this policy statement, see Allows access to the AWS account and enables IAM policies.

Default key policy when you create a KMS key with the AWS Management Console

When you create a KMS key with the AWS Management Console, the key policy begins with the policy statement that allows access to the AWS account and enables IAM policies. The console then adds a key administrators statement, a key users statement, and (for most key types) a statement that allows principals to use the KMS key with other AWS services. You can use the features of the AWS KMS console to specify the IAM users, IAMroles, and AWS accounts who are key administrators and those who are key users (or both).

Permissions

- Allows access to the AWS account and enables IAM policies
- Allows key administrators to administer the KMS key
- Allows key users to use the KMS key
  - Allows key users to use a KMS key for cryptographic operations
  - Allows key users to use the KMS key with AWS services

Allows access to the AWS account and enables IAM policies

The following default key policy statement is critical.

- It gives the AWS account that owns the KMS key full access to the KMS key.

  Unlike other AWS resource policies, an AWS KMS key policy does not automatically give permission to the account or any of its identities. To give permission to account administrators, the key policy must include an explicit statement that provides this permission, like this one.
- It allows the account to use IAM policies to allow access to the KMS key, in addition to the key policy.

  Without this permission, IAM policies that allow access to the key are ineffective, although IAM policies that deny access to the key are still effective.
- It reduces the risk of the key becoming unmanageable by giving access control permission to the account administrators, including the account root user, which cannot be deleted.

The following key policy statement is the entire default key policy for KMS keys created programmatically. It's the first policy statement in the default key policy for KMS keys created in the AWS KMS console.

```
{
  "Sid": "Enable IAM User Permissions",
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:root"
   },
  "Action": "kms:*",
  "Resource": "*"
}
```

Allows IAM policies to allow access to the KMS key.

The key policy statement shown above gives the AWS account that owns the key permission to use IAM policies, as well as key policies, to allow all actions (`kms:*`) on the KMS key.

The principal in this key policy statement is the account principal, which is represented by an ARN in this format: `arn:aws:iam::`*`account-id`*`:root`. The account principal represents the AWS account and its administrators.

When the principal in a key policy statement is the account principal, the policy statement doesn't give any IAM principal permission to use the KMS key. Instead, it allows the account to use IAM policies to *delegate* the permissions specified in the policy statement. This default key policy statement allows the account to use IAM policies to delegate permission for all actions (`kms:*`) on the KMS key.

Reduces the risk of the KMS key becoming unmanageable.

Unlike other AWS resource policies, an AWS KMS key policy does not automatically give permission to the account or any of its principals. To give permission to any principal, including the account principal, you must use a key policy statement that provides the permission explicitly. You are not required to give the account principal, or any principal, access to the KMS key. However, giving access to the account principal helps you prevent the key from becoming unmanageable.

For example, suppose you create a key policy that gives only one user access to the KMS key. If you then delete that user, the key becomes unmanageable and you must contact AWS Support to regain access to the KMS key.

The key policy statement shown above gives permission to control the key to the account principal, which represents the AWS account and its administrators, including the account root user. The account root user is the only principal that cannot be deleted unless you delete the AWS account. IAM best practices discourage acting on behalf of the account root user, except in an emergency. However, you might need to act as the account root user if you delete all other users and roles with access to the KMS key.

Allows key administrators to administer the KMS key

The default key policy created by the console allows you to choose IAM users and roles in the account and make them *key administrators*. This statement is called the *key administrators statement*. Key administrators have permissions to manage the KMS key, but do not have permissions to use the KMS key in cryptographic operations. You can add IAM users and roles to the list of key administrators when you create the KMS key in the default view or the policy view.

⚠ **Warning**

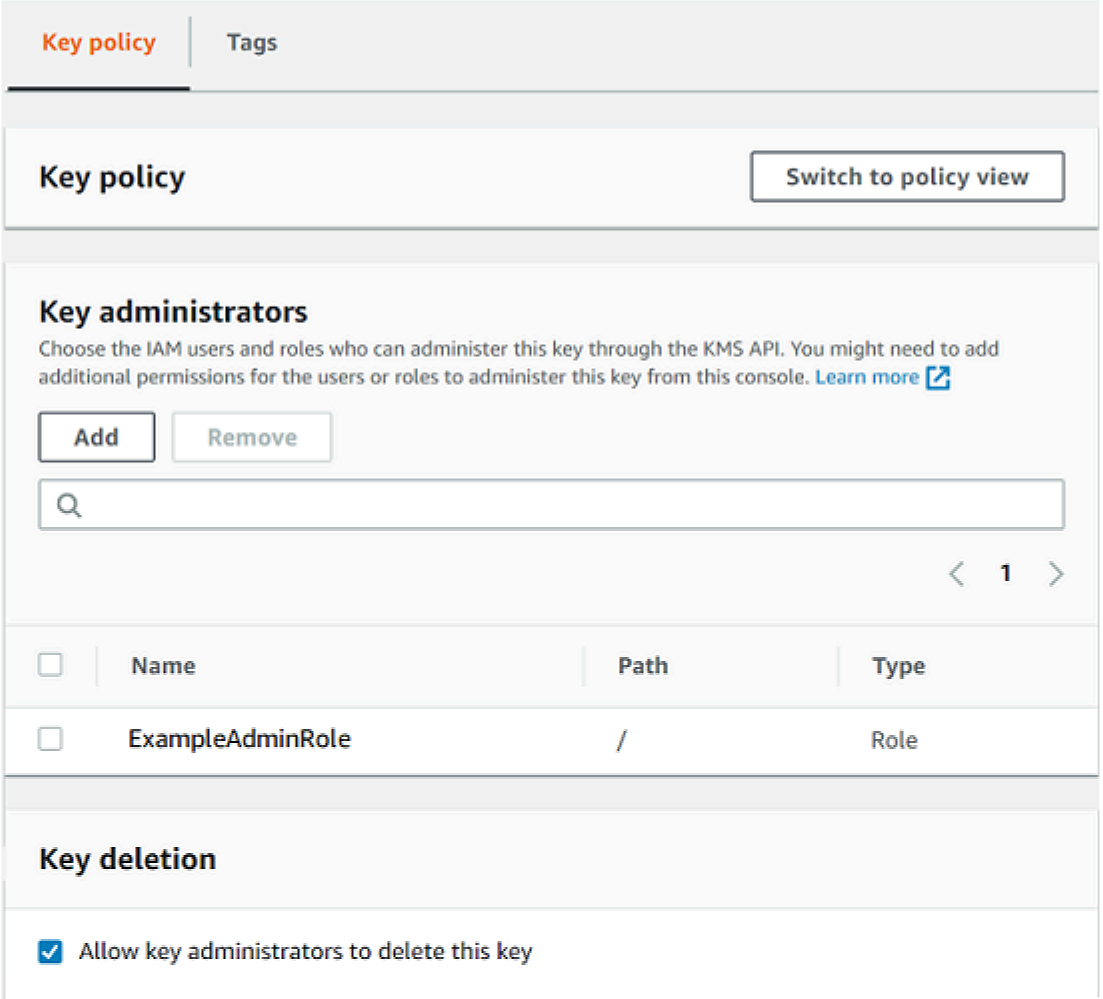
Because key administrators have permission to change the key policy and create grants, they can give themselves and others AWS KMS permissions not specified in this policy.

Principals who have permission to manage tags and aliases can also control access to a KMS key. For details, see ABAC for AWS KMS.

Note

IAM best practices discourage the use of IAM users with long-term credentials. Whenever possible, use IAM roles, which provide temporary credentials. For details, see Security best practices in IAM in the *IAM User Guide*.

The following example shows the key administrators statement in the default view of the AWS KMS console.



Key policy

Key administrators

Choose the IAM users and roles who can administer this key through the KMS API. You might need to add additional permissions for the users or roles to administer this key from this console. Learn more

□	Name	Path	Type
□	ExampleAdminRole	/	Role

Key deletion

☑ Allow key administrators to delete this key

The following is an example key administrators statement in the policy view of the AWS KMS console. This key administrators statement is for a single-region symmetric encryption KMS key.

```
{
  "Sid": "Allow access for Key Administrators",
  "Effect": "Allow",
  "Principal": {"AWS":"arn:aws:iam::111122223333:role/ExampleAdminRole"},
  "Action": [
    "kms:Create*",
    "kms:Describe*",
    "kms:Enable*",
    "kms:List*",
    "kms:Put*",
    "kms:Update*",
    "kms:Revoke*",
    "kms:Disable*",
    "kms:Get*",
    "kms:Delete*",
    "kms:TagResource",
    "kms:UntagResource",
    "kms:ScheduleKeyDeletion",
    "kms:CancelKeyDeletion",
    "kms:RotateKeyOnDemand"
  ],
  "Resource": "*"
}
```

The default key administrators statement for the most common KMS key, a single-Region symmetric encryption KMS key, allows the following permissions. For detailed information about each permission, see the AWS KMS permissions.

When you use the AWS KMS console to create a KMS key, the console adds the users and roles you specify to the `Principal` element in the key administrators statement.

Many of these permissions contain the wildcard character (*), which allows all permissions that begin with the specified verb. As a result, when AWS KMS adds new API operations, key administrators are automatically allowed to use them. You don't have to update your key policies to include the new operations. If you prefer to limit your key administrators to a fixed set of API operations, you can change your key policy.

`kms:Create*`

Allows `kms:CreateAlias` and `kms:CreateGrant`. (The `kms:CreateKey` permission is valid only in an IAM policy.)

kms:Describe*

Allows kms:DescribeKey. The kms:DescribeKey permission is required to view the key details page for a KMS key in the AWS Management Console.

kms:Enable*

Allows kms:EnableKey. For symmetric encryption KMS keys, it also allows kms:EnableKeyRotation.

kms:List*

Allows kms:ListGrants, kms:ListKeyPolicies, and kms:ListResourceTags. (The kms:ListAliases and kms:ListKeys permissions, which are required to view KMS keys in the AWS Management Console, are valid only in IAM policies.)

kms:Put*

Allows kms:PutKeyPolicy. This permission allows key administrators to change the key policy for this KMS key.

kms:Update*

Allows kms:UpdateAlias and kms:UpdateKeyDescription. For multi-Region keys, it allows kms:UpdatePrimaryRegion on this KMS key.

kms:Revoke*

Allows kms:RevokeGrant, which allows key administrators to delete a grant even if they are not a retiring principal in the grant.

kms:Disable*

Allows kms:DisableKey. For symmetric encryption KMS keys, it also allows kms:DisableKeyRotation.

kms:Get*

Allows kms:GetKeyPolicy and kms:GetKeyRotationStatus. For KMS keys with imported key material, it allows kms:GetParametersForImport. For asymmetric KMS keys, it allows kms:GetPublicKey. The kms:GetKeyPolicy permission is required to view the key policy of a KMS key in the AWS Management Console.

`kms:Delete*`

Allows `kms:DeleteAlias`. For keys with imported key material, it allows `kms:DeleteImportedKeyMaterial`. The `kms:Delete*` permission does not allow key administrators to delete the KMS key (`ScheduleKeyDeletion`).

`kms:TagResource`

Allows `kms:TagResource`, which allows key administrators to add tags to the KMS key. Because tags can also be used to control access to the KMS key, this permission can allow administrators to allow or deny access to the KMS key. For details, see ABAC for AWS KMS.

`kms:UntagResource`

Allows `kms:UntagResource`, which allows key administrators to delete tags from the KMS key. Because tags can be used to control access to the key, this permission can allow administrators to allow or deny access to the KMS key. For details, see ABAC for AWS KMS.

`kms:ScheduleKeyDeletion`

Allows `kms:ScheduleKeyDeletion`, which allows key administrators to delete this KMS key. To delete this permission, clear the **Allow key administrators to delete this key** option.

`kms:CancelKeyDeletion`

Allows `kms:CancelKeyDeletion`, which allows key administrators to cancel deletion of this KMS key. To delete this permission, clear the **Allow key administrators to delete this key** option.

`kms:RotateKeyOnDemand`

Allows `kms:RotateKeyOnDemand`, which allows key administrators to perform on-demand rotation of the key material in this KMS key.

AWS KMS adds the following permissions to the default key administrators statement when you create special-purpose keys.

`kms:ImportKeyMaterial`

The `kms:ImportKeyMaterial` permission allows key administrators to import key material into the KMS key. This permission is included in the key policy only when you create a KMS key with no key material.

`kms:ReplicateKey`

The `kms:ReplicateKey` permission allows key administrators to create a replica of a multi-Region primary key in a different AWS Region. This permission is included in the key policy only when you create a multi-Region primary or replica key.

`kms:UpdatePrimaryRegion`

The `kms:UpdatePrimaryRegion` permission allows key administrators to change a multi-Region replica key to a multi-Region primary key. This permission is included in the key policy only when you create a multi-Region primary or replica key.

Allows key users to use the KMS key

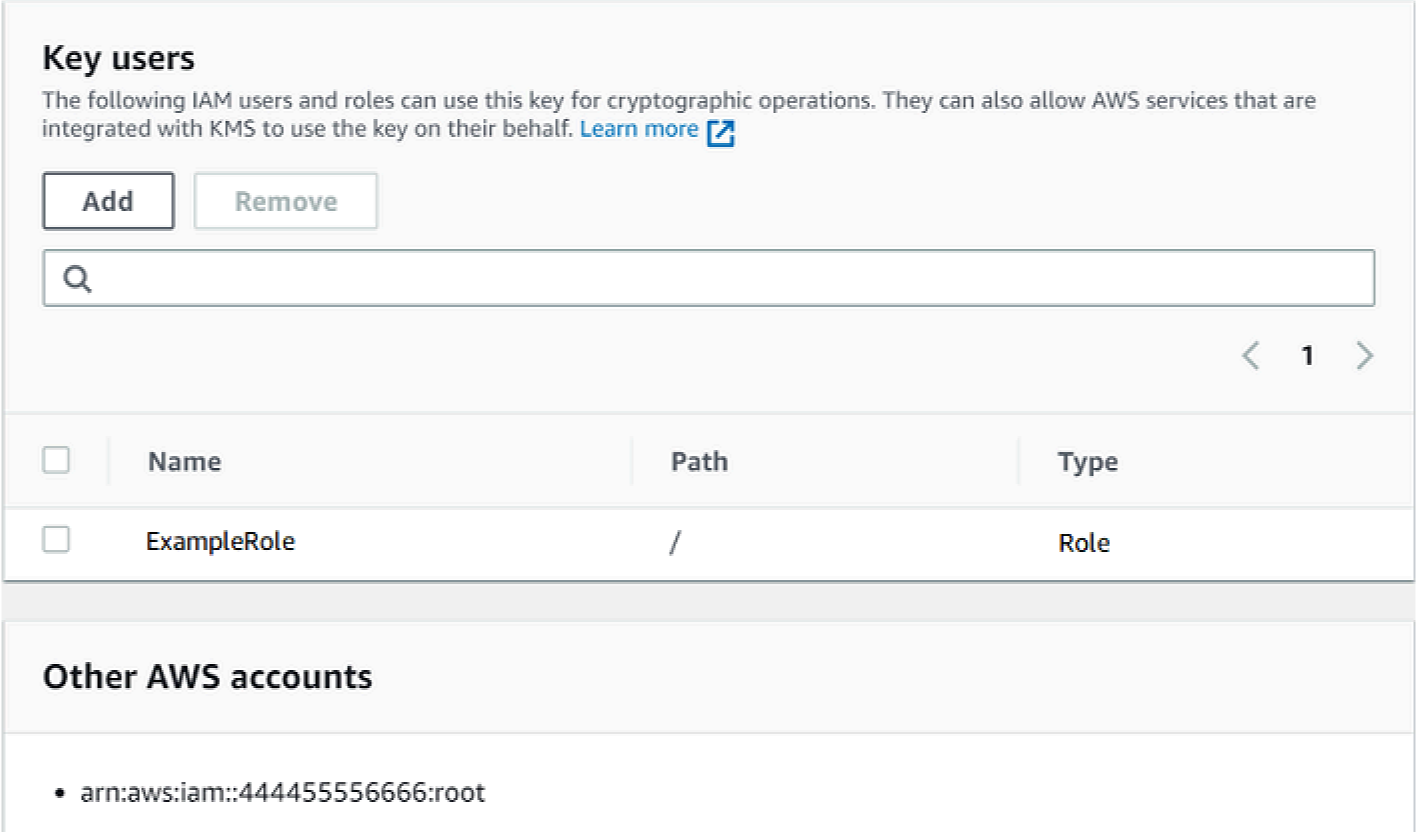
The default key policy that the console creates for KMS keys allows you to choose IAM users and IAM roles in the account, and external AWS accounts, and make them *key users*.

The console adds two policy statements to the key policy for key users.

- Use the KMS key directly — The first key policy statement gives key users permission to use the KMS key directly for all supported cryptographic operations for that type of KMS key.
- Use the KMS key with AWS services — The second policy statement gives key users permission to allow AWS services that are integrated with AWS KMS to use the KMS key on their behalf to protect resources, such as Amazon S3 buckets and Amazon DynamoDB tables.

You can add IAM users, IAM roles, and other AWS accounts to the list of key users when you create the KMS key. You can also edit the list with the console's default view for key policies, as shown in the following image. The default view for key policies is on the key details page. For more information about allowing users in other AWS accounts to use the KMS key, see Allowing users in other accounts to use a KMS key.

> **Note**
>
> IAM best practices discourage the use of IAM users with long-term credentials. Whenever possible, use IAM roles, which provide temporary credentials. For details, see Security best practices in IAM in the *IAM User Guide*.



Key users

The following IAM users and roles can use this key for cryptographic operations. They can also allow AWS services that are integrated with KMS to use the key on their behalf. Learn more

□	Name	Path	Type
□	ExampleRole	/	Role

Other AWS accounts

- arn:aws:iam::444455556666:root

The default *key users statements* for a single-Region symmetric allows the following permissions. For detailed information about each permission, see the AWS KMS permissions.

When you use the AWS KMS console to create a KMS key, the console adds the users and roles you specify to the `Principal` element in each key users statement.

```
{
  "Sid": "Allow use of the key",
  "Effect": "Allow",
  "Principal": {"AWS": [
    "arn:aws:iam::111122223333:role/ExampleRole",
    "arn:aws:iam::444455556666:root"
  ]},
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:DescribeKey"
  ],
  "Resource": "*"
},
{
  "Sid": "Allow attachment of persistent resources",
  "Effect": "Allow",
  "Principal": {"AWS": [
    "arn:aws:iam::111122223333:role/ExampleRole",
    "arn:aws:iam::444455556666:root"
  ]},
  "Action": [
    "kms:CreateGrant",
    "kms:ListGrants",
    "kms:RevokeGrant"
  ],
  "Resource": "*",
  "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
}
```

Allows key users to use a KMS key for cryptographic operations

Key users have permission to use the KMS key directly in all cryptographic operations supported on the KMS key. They can also use the DescribeKey operation to get detailed information about the KMS key in the AWS KMS console or by using the AWS KMS API operations.

By default, the AWS KMS console adds key users statements like those in the following examples to the default key policy. Because they support different API operations, the actions in the policy statements for symmetric encryption KMS keys, HMAC KMS keys, asymmetric KMS keys for public key encryption, and asymmetric KMS keys for signing and verification are slightly different.

Symmetric encryption KMS keys

The console adds the following statement to the key policy for symmetric encryption KMS keys.

```
{
  "Sid": "Allow use of the key",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::111122223333:role/ExampleKeyUserRole"},
  "Action": [
    "kms:Decrypt",
    "kms:DescribeKey",
    "kms:Encrypt",
    "kms:GenerateDataKey*",
    "kms:ReEncrypt*"
  ],
  "Resource": "*"
}
```

HMAC KMS keys

The console adds the following statement to the key policy for HMAC KMS keys.

```
{
  "Sid": "Allow use of the key",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::111122223333:role/ExampleKeyUserRole"},
  "Action": [
    "kms:DescribeKey",
    "kms:GenerateMac",
    "kms:VerifyMac"
  ],
  "Resource": "*"
}
```

Asymmetric KMS keys for public key encryption

The console adds the following statement to the key policy for asymmetric KMS keys with a key usage of **Encrypt and decrypt**.

```
{
  "Sid": "Allow use of the key",
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/ExampleKeyUserRole"
  },
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:DescribeKey",
    "kms:GetPublicKey"
  ],
  "Resource": "*"
}
```

Asymmetric KMS keys for signing and verification

The console adds the following statement to the key policy for asymmetric KMS keys with a key usage of **Sign and verify**.

```
{
  "Sid": "Allow use of the key",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::111122223333:role/ExampleKeyUserRole"},
  "Action": [
    "kms:DescribeKey",
    "kms:GetPublicKey",
    "kms:Sign",
    "kms:Verify"
  ],
  "Resource": "*"
}
```

Asymmetric KMS keys for deriving shared secrets

The console adds the following statement to the key policy for asymmetric KMS keys with a key usage of **Key agreement**.

```
{
  "Sid": "Allow use of the key",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::111122223333:role/ExampleKeyUserRole"},
  "Action": [
    "kms:DescribeKey",
    "kms:GetPublicKey",
    "kms:DeriveSharedSecret"
  ],
  "Resource": "*"
}
```

The actions in these statements give the key users the following permissions.

`kms:Encrypt`

Allows key users to encrypt data with this KMS key.

`kms:Decrypt`

Allows key users to decrypt data with this KMS key.

`kms:DeriveSharedSecret`

Allows key users to derive shared secrets with this KMS key.

`kms:DescribeKey`

Allows key users to get detailed information about this KMS key including its identifiers, creation date, and key state. It also allows the key users to display details about the KMS key in the AWS KMS console.

`kms:GenerateDataKey*`

Allows key users to request a symmetric data key or an asymmetric data key pair for client-side cryptographic operations. The console uses the * wildcard character to represent permission for the following API operations: GenerateDataKey, GenerateDataKeyWithoutPlaintext, GenerateDataKeyPair, and GenerateDataKeyPairWithoutPlaintext. These permissions are valid only on the symmetric KMS keys that encrypt the data keys.

kms:GenerateMac

Allows key users to use an HMAC KMS key to generate an HMAC tag.

kms:GetPublicKey

Allows key users to download the public key of the asymmetric KMS key. Parties with whom you share this public key can encrypt data outside of AWS KMS. However, those ciphertexts can be decrypted only by calling the Decrypt operation in AWS KMS.

kms:ReEncrypt*

Allows key users to re-encrypt data that was originally encrypted with this KMS key, or to use this KMS key to re-encrypt previously encrypted data. The ReEncrypt operation requires access to both source and destination KMS keys. To accomplish this, you can allow the `kms:ReEncryptFrom` permission on the source KMS key and `kms:ReEncryptTo` permission on the destination KMS key. However, for simplicity, the console allows `kms:ReEncrypt*` (with the `*` wildcard character) on both KMS keys.

kms:Sign

Allows key users to sign messages with this KMS key.

kms:Verify

Allows key users to verify signatures with this KMS key.

kms:VerifyMac

Allows key users to use an HMAC KMS key to verify an HMAC tag.

Allows key users to use the KMS key with AWS services

The default key policy in the console also gives key users the grant permissions they need to protect their data in AWS services that use grants. AWS services often use grants to get specific and limited permission to use a KMS key.

This key policy statement allows the key user to create, view, and revoke grants on the KMS key, but only when the grant operation request comes from an AWS service integrated with AWS KMS. The kms:GrantIsForAWSResource policy condition doesn't allow the user to call these grant operations directly. When the key user allows it, an AWS service can create a grant on the user's behalf that allows the service to use the KMS key to protect the user's data.

Key users require these grant permissions to use their KMS key with integrated services, but these permissions are not sufficient. Key users also need permission to use the integrated services. For details about giving users access to an AWS service that integrates with AWS KMS, consult the documentation for the integrated service.

```
{
  "Sid": "Allow attachment of persistent resources",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::111122223333:role/ExampleKeyUserRole"},
  "Action": [
    "kms:CreateGrant",
    "kms:ListGrants",
    "kms:RevokeGrant"
  ],
  "Resource": "*",
  "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
}
```

For example, key users can use these permissions on the KMS key in the following ways.

- Use this KMS key with Amazon Elastic Block Store (Amazon EBS) and Amazon Elastic Compute Cloud (Amazon EC2) to attach an encrypted EBS volume to an EC2 instance. The key user implicitly gives Amazon EC2 permission to use the KMS key to attach the encrypted volume to the instance. For more information, see How Amazon Elastic Block Store (Amazon EBS) uses AWS KMS.

- Use this KMS key with Amazon Redshift to launch an encrypted cluster. The key user implicitly gives Amazon Redshift permission to use the KMS key to launch the encrypted cluster and create encrypted snapshots. For more information, see How Amazon Redshift uses AWS KMS.
- Use this KMS key with other AWS services integrated with AWS KMS that use grants to create, manage, or use encrypted resources with those services.

The default key policy allows key users to delegate their grant permission to *all* integrated services that use grants. However, you can create a custom key policy that restricts the permission to specified AWS services. For more information, see the kms:ViaService condition key.

Viewing a key policy

You can view the key policy for an AWS KMS customer managed key or an AWS managed key in your account by using the AWS Management Console or the GetKeyPolicy operation in the AWS KMS API. You cannot use these techniques to view the key policy of a KMS key in a different AWS account.

To learn more about AWS KMS key policies, see Key policies in AWS KMS. To learn how to determine which users and roles have access to a KMS key, see the section called “Determining access”.

Topics

- Viewing a key policy (console)
- Viewing a key policy (AWS KMS API)

Viewing a key policy (console)

Authorized users can view the key policy for an AWS managed key or a customer managed key on the **Key policy** tab of the AWS Management Console.

To view the key policy for a KMS key in the AWS Management Console, you must have kms:ListAliases, kms:DescribeKey, and kms:GetKeyPolicy permissions.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.

2. To change the AWS Region, use the Region selector in the upper-right corner of the page.

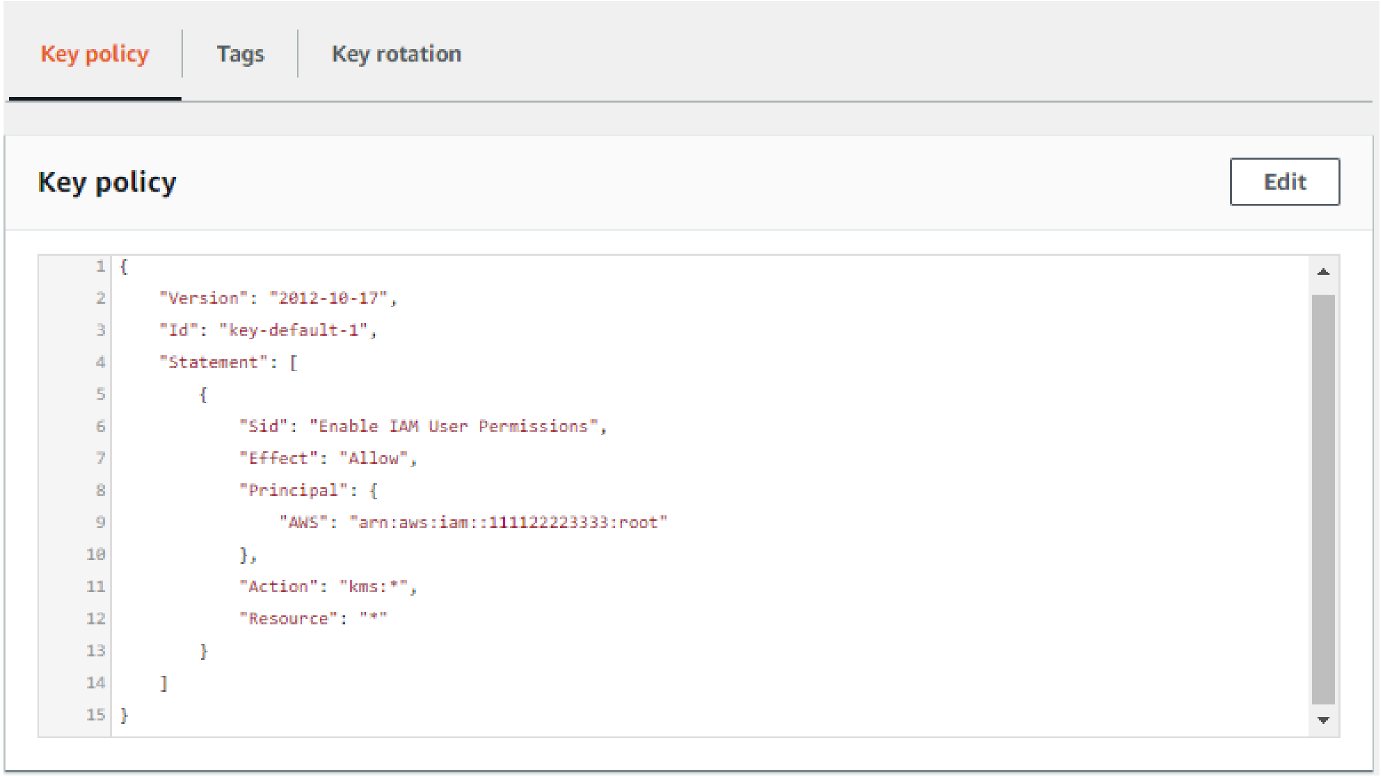
3. To view the keys in your account that AWS creates and manages for you, in the navigation pane, choose **AWS managed keys**. To view the keys in your account that you create and manage, in the navigation pane choose **Customer managed keys**.

4. In the list of KMS keys, choose the alias or key ID of the KMS key that you want to examine.

5. Choose the **Key policy** tab.

   On the **Key policy** tab, you might see the key policy document. This is *policy view*. In the key policy statements, you can see the principals who have been given access to the KMS key by the key policy, and you can see the actions they can perform.

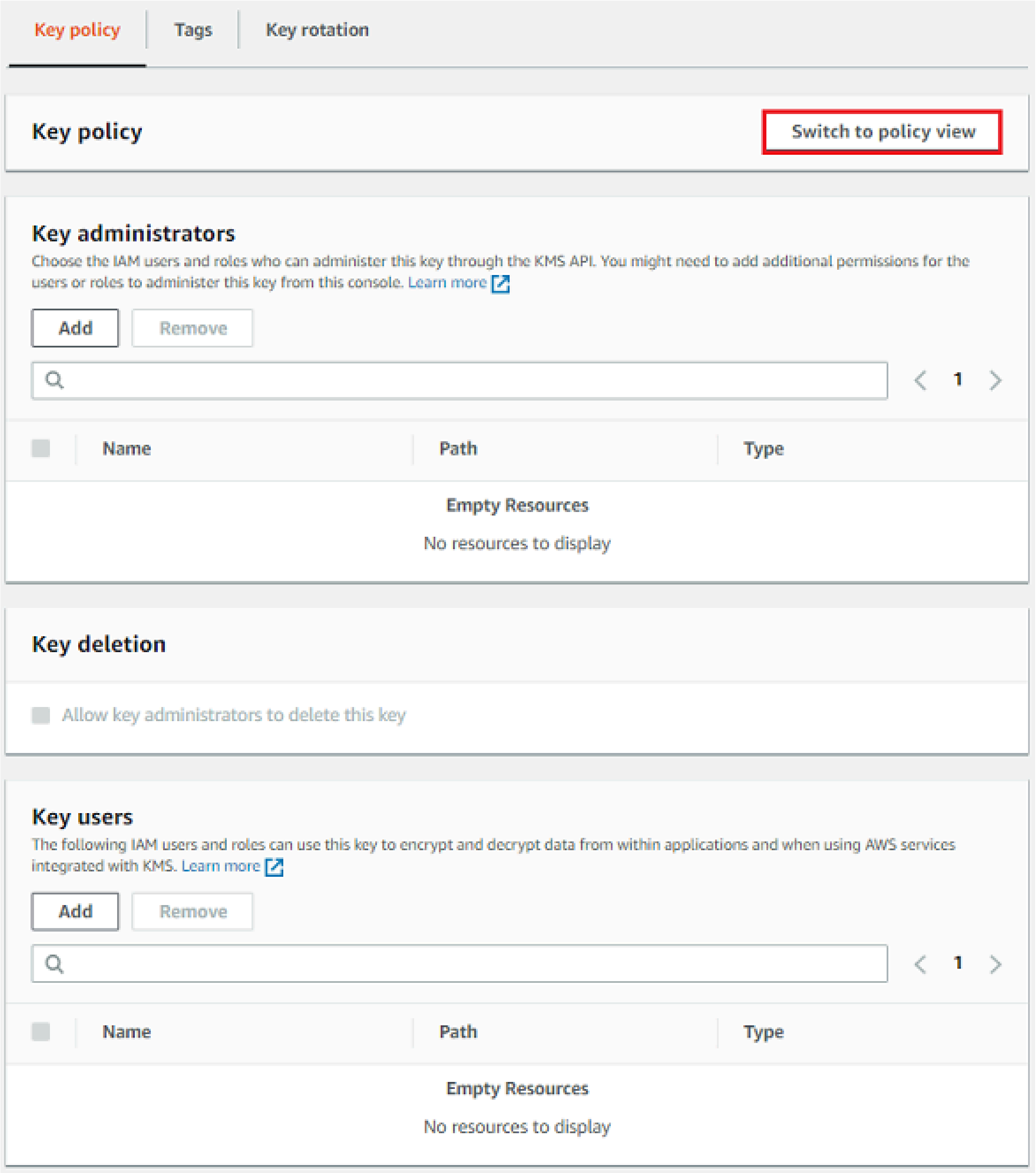
   The following example shows the policy view for the default key policy.



```
{
    "Version": "2012-10-17",
    "Id": "key-default-1",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        }
    ]
}
```

   Or, if you created the KMS key in the AWS Management Console, you will see the *default view* with sections for **Key administrators**, **Key deletion**, and **Key Users**. To see the key policy document, choose **Switch to policy view**.

   The following example shows the default view for the default key policy.



Key administrators

Choose the IAM users and roles who can administer this key through the KMS API. You might need to add additional permissions for the users or roles to administer this key from this console. Learn more

Key users

The following IAM users and roles can use this key to encrypt and decrypt data from within applications and when using AWS services integrated with KMS. Learn more

Viewing a key policy (AWS KMS API)

To get the key policy for a KMS key in your AWS account, use the GetKeyPolicy operation in the AWS KMS API. You cannot use this operation to view a key policy in a different account.

The following example uses the get-key-policy command in the AWS Command Line Interface (AWS CLI), but you can use any AWS SDK to make this request.

Note that the `PolicyName` parameter is required even though `default` is its only valid value. Also, this command requests the output in text, rather than JSON, to make it easier to view.

Before running this command, replace the example key ID with a valid one from your account.

```
$ aws kms get-key-policy --key-id 1234abcd-12ab-34cd-56ef-1234567890ab --policy-name
 default --output text
```

The response should be similar to the following one, which returns the default key policy.

```
{
  "Version" : "2012-10-17",
  "Id" : "key-consolepolicy-3",
  "Statement" : [ {
    "Sid" : "Enable IAM User Permissions",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::111122223333:root"
    },
    "Action" : "kms:*",
    "Resource" : "*"
  } ]
}
```

Changing a key policy

You can change the key policy for a KMS key in your AWS account by using the AWS Management Console or the PutKeyPolicy operation. You cannot use these techniques to change the key policy of a KMS key in a different AWS account.

When changing a key policy, keep in mind the following rules:

- You can view the key policy for an AWS managed key or a customer managed key, but you can only change the key policy for a customer managed key. The policies of AWS managed keys are created and managed by the AWS service that created the KMS key in your account. You cannot view or change the key policy for an AWS owned key.
- You can add or remove IAM users, IAM roles, and AWS accounts in the key policy, and change the actions that are allowed or denied for those principals. For more information about the ways to specify principals and permissions in a key policy, see Key policies.

- You cannot add IAM groups to a key policy, but you can add multiple IAM users and IAM roles. For more information, see Allowing multiple IAM principals to access a KMS key.
- If you add external AWS accounts to a key policy, you must also use IAM policies in the external accounts to give permissions to IAM users, groups, or roles in those accounts. For more information, see Allowing users in other accounts to use a KMS key.
- The resulting key policy document cannot exceed 32 KB (32,768 bytes).

Topics

How to change a key policy

You can change a key policy in three different ways as explained in the following sections.

Topics

Using the AWS Management Console default view

You can use the console to change a key policy with a graphical interface called the *default view*.

If the following steps don't match what you see in the console, it might mean that this key policy was not created by the console. Or it might mean that the key policy has been modified in a way that the console's default view does not support. In that case, follow the steps at Using the AWS Management Console policy view or Using the AWS KMS API.

1. View the key policy for a customer managed key as described in Viewing a key policy (console). (You cannot change the key policies of AWS managed keys.)
2. Decide what to change.
   - To add or remove key administrators, and to allow or prevent key administrators from deleting the KMS key, use the controls in the **Key administrators** section of the page. Key

administrators manage the KMS key, including enabling and disabling it, setting key policy, and enabling key rotation.

- To add or remove key users, and to allow or disallow external AWS accounts to use the KMS key, use the controls in the **Key users** section of the page. Key users can use the KMS key in cryptographic operations, such as encrypting, decrypting, re-encrypting, and generating data keys.

Using the AWS Management Console policy view

You can use the console to change a key policy document with the console's *policy view*.

1. View the key policy for a customer managed key as described in Viewing a key policy (console). (You cannot change the key policies of AWS managed keys.)
2. In the **Key Policy** section, choose **Switch to policy view**.
3. Edit the key policy document, and then choose **Save changes**.

Using the AWS KMS API

You can use the PutKeyPolicy operation to change the key policy of a KMS key in your AWS account. You cannot use this API on a KMS key in a different AWS account.

1. Use the GetKeyPolicy operation to get the existing key policy document, and then save the key policy document to a file. For sample code in multiple programming languages, see Getting a key policy.
2. Open the key policy document in your preferred text editor, edit the key policy document, and then save the file.
3. Use the PutKeyPolicy operation to apply the updated key policy document to the KMS key. For sample code in multiple programming languages, see Setting a key policy.

For an example of copying a key policy from one KMS key to another, see the GetKeyPolicy example in the AWS CLI Command Reference.

Allowing multiple IAM principals to access a KMS key

IAM groups are not valid principals in a key policy. To allow multiple users and roles to access a KMS key, do one of the following:

- Use an IAM role as the principal in the key policy. Multiple authorized users can assume the role as needed. For details, see IAM roles in the *IAM User Guide*.

  While you can list multiple IAM users in a key policy, this practice is not recommended because it requires that you update the key policy every time the list of authorized users changes. Also, IAM best practices discourage the use of IAM users with long-term credentials. For details, see Security best practices in IAM in the *IAM User Guide*.
- Use an IAM policy to give permission to an IAM group. To do this, ensure that the key policy includes the statement that enables IAM policies to allow access to the KMS key, create an IAM policy that allows access to the KMS key, and then attach that policy to an IAM group that contains the authorized IAM users. Using this approach, you don't need to change any policies when the list of authorized users changes. Instead, you only need to add or remove those users from the appropriate IAM group. For details, see IAM user groups in the *IAM User Guide*

For more information about how AWS KMS key policies and IAM policies work together, see Troubleshooting key access.

Permissions for AWS services in key policies

Many AWS services use AWS KMS keys to protect the resources they manage. When a service uses AWS owned keys or AWS managed keys, the service establishes and maintains the key policies for these KMS keys.

However, when you use a customer managed key with an AWS service, you set and maintain the key policy. That key policy must allow the service the minimum permissions that it requires to protect the resource on your behalf. We recommend that you follow the principle of least privilege: give the service only the permissions that it requires. You can do this effectively by learning which permissions the service needs and using AWS global condition keys and AWS KMS condition keys to refine the permissions.

To find the permissions that the service requires on a customer managed key, see the encryption documentation for the service. For example, for the permissions that Amazon Elastic Block Store (Amazon EBS) requires, see *Permissions for IAM users* in the Amazon EC2 User Guide and Amazon EC2 User Guide. For the permissions that Secrets Manager requires, see Authorizing use of the KMS key in the *AWS Secrets Manager User Guide*.

Implementing least privileged permissions

When you give an AWS service permission to use a KMS key, ensure that the permission is valid only for the resources that the service must access on your behalf. This least privilege strategy helps to prevent unauthorized use of a KMS key when requests are passed between AWS services.

To implement a least privilege strategy, use we recommend using AWS KMS encryption context condition keys and the global source ARN or source account condition keys.

Using encryption context condition keys

The most effective way to implement least privileged permissions when using AWS KMS resources is to include the kms:EncryptionContext:*context-key* or kms:EncryptionContextKeys condition keys in the policy that allows principals to call AWS KMS cryptographic operations. These condition keys are particularly effective because they associate the permission with the encryption context that is bound to the ciphertext when the resource is encrypted.

Use encryption context conditions keys only when the action in the policy statement is CreateGrant or an AWS KMS symmetric cryptographic operation that takes an `EncryptionContext` parameter, such as the operations like GenerateDataKey or Decrypt. (For a list of supported operations, see kms:EncryptionContext:*context-key* or kms:EncryptionContextKeys.) If you use these condition keys to allow other operations, such as DescribeKey, permission will be denied.

Set the value to the encryption context that the service uses when it encrypts the resource. This information is typically available in the Security chapter of the service documentation. For example, the encryption context for AWS Proton identifies the AWS Proton resource and its associated template. The AWS Secrets Manager encryption context identifies the secret and its version. The encryption context for Amazon Location identifies the tracker or collection.

The following example key policy statement allows Amazon Location Service to create grants on behalf of authorized users. This policy statement limits the permission by using the kms:ViaService, kms:CallerAccount, and `kms:EncryptionContext:`*`context-key`* condition keys to tie the permission to a particular tracker resource.

```
{
  "Sid": "Allow Amazon Location to create grants on behalf of authorized users",
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/LocationTeam"
  },
  "Action": "kms:CreateGrant",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ViaService": "geo.us-west-2.amazonaws.com",
      "kms:CallerAccount": "111122223333",
      "kms:EncryptionContext:aws:geo:arn": "arn:aws:geo:us-west-2:111122223333:tracker/
SAMPLE-Tracker"
    }
  }
}
```

Using `aws:SourceArn` or `aws:SourceAccount` condition keys

When the principal in a key policy statement is an AWS service principal, we strongly recommend that you use the aws:SourceArn or aws:SourceAccount global condition keys, in addition to the `kms:EncryptionContext:`*`context-key`* condition key. The ARN and account values are included in the authorization context only when a request comes to AWS KMS from another AWS service. This combination of conditions implements least privileged permissions and avoids a potential confused deputy scenario. Service principals are not typically used as principals in a key policy, but some AWS services, such as AWS CloudTrail, require it.

To use the `aws:SourceArn` or `aws:SourceAccount` global condition keys, set the value to the Amazon Resource Name (ARN) or account of the resource that is being encrypted. For example, in a key policy statement that gives AWS CloudTrail permission to encrypt a trail, set the value of `aws:SourceArn` to the ARN of the trail. Whenever possible, use `aws:SourceArn`, which is more specific. Set the value to the ARN or an ARN pattern with wildcard characters. If you don't know the ARN of the resource, use `aws:SourceAccount` instead.

> **Note**
>
> If a resource ARN includes characters that are not permitted in an AWS KMS key policy, you cannot use that resource ARN in the value of the `aws:SourceArn` condition key. Instead, use the `aws:SourceAccount` condition key. For details about key policy document rules, see Key policy format.

In the following example key policy, the principal who gets the permissions is the AWS CloudTrail service principal, `cloudtrail.amazonaws.com`. To implement least privilege, this policy uses the `aws:SourceArn` and `kms:EncryptionContext:`*`context-key`* condition keys. The policy statement allows CloudTrail to use the KMS key to generate the data key that it uses to encrypt a trail. The `aws:SourceArn` and `kms:EncryptionContext:`*`context-key`* conditions are evaluated independently. Any request to use the KMS key for the specified operation must satisfy both conditions.

To restrict the service's permission to the `finance` trail in the example account (111122223333) and `us-west-2` Region, this policy statement sets the `aws:SourceArn` condition key to the ARN of a particular trail. The condition statement uses the ArnEquals operator to ensure that every element in the ARN is evaluated independently when matching. The example also uses the `kms:EncryptionContext:`*`context-key`* condition key to limit the permission to trails in a particular account and Region.

Before using this key policy, replace the example account ID, Region, and trail name with valid values from your account.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow CloudTrail to encrypt logs",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudtrail.amazonaws.com"
      },
      "Action": "kms:GenerateDataKey",
      "Resource": "*",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": [
            "arn:aws:cloudtrail:us-west-2:111122223333:trail/finance"
          ]
        },
        "StringLike": {
          "kms:EncryptionContext:aws:cloudtrail:arn": [
            "arn:aws:cloudtrail:*:111122223333:trail/*"
          ]
        }
      }
    }
  ]
}
```

Using IAM policies with AWS KMS

You can use IAM policies, along with key policies, grants, and VPC endpoint policies, to control access to your AWS KMS keys in AWS KMS.

> **Note**
>
> To use an IAM policy to control access to a KMS key, the key policy for the KMS key must give the account permission to use IAM policies. Specifically, the key policy must include the policy statement that enables IAM policies.
> This section explains how to use IAM policies to control access to AWS KMS operations. For more general information about IAM, see the IAM User Guide.

All KMS keys must have a key policy. IAM policies are optional. To use an IAM policy to control access to a KMS key, the key policy for the KMS key must give the account permission to use IAM policies. Specifically, the key policy must include the policy statement that enables IAM policies.

IAM policies can control access to any AWS KMS operation. Unlike key policies, IAM policies can control access to multiple KMS keys and provide permissions for the operations of several related AWS services. But IAM policies are particularly useful for controlling access to operations, such as CreateKey, that can't be controlled by a key policy because they don't involve any particular KMS key.

If you access AWS KMS through an Amazon Virtual Private Cloud (Amazon VPC) endpoint, you can also use a VPC endpoint policy to limit access to your AWS KMS resources when using the endpoint. For example, when using the VPC endpoint, you might only allow the principals in your AWS account to access your customer managed keys. For details, see Controlling access to a VPC endpoint.

For help writing and formatting a JSON policy document, see the IAM JSON Policy Reference in the *IAM User Guide*.

Topics

- Permissions required to use the AWS KMS console
- AWS managed policy for power users
- IAM policy examples

Overview of IAM policies

You can use IAM policies in the following ways:

- **Attach a permissions policy to a role for federation or cross-account permissions** – You can attach an IAM policy to an IAM role to enable identity federation, allow cross-account permissions, or give permissions to applications running on EC2 instances. For more information about the various use cases for IAM roles, see IAM Roles in the *IAM User Guide*.
- **Attach a permissions policy to a user or a group** – You can attach a policy that allows a user or group of users to call AWS KMS operations. However, IAM best practices recommend that you use identities with temporary credentials, such as IAM roles, whenever possible.

The following example shows an IAM policy with AWS KMS permissions. This policy allows the IAM identities to which it is attached to list all KMS keys and aliases.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:ListKeys",
      "kms:ListAliases"
    ],
    "Resource": "*"
  }
}
```

Like all IAM policies, this policy doesn't have a `Principal` element. When you attach an IAM policy to an IAM identity, that identity gets the permissions specified in the policy.

For a table showing all of the AWS KMS API actions and the resources that they apply to, see the Permissions reference.

Best practices for IAM policies

Securing access to AWS KMS keys is critical to the security of all of your AWS resources. KMS keys are used to protect many of the most sensitive resources in your AWS account. Take the time to design the key policies, IAM policies, grants, and VPC endpoint policies that control access to your KMS keys.

In IAM policy statements that control access to KMS keys, use the least privileged principle. Give IAM principals only the permissions they need on only the KMS keys they must use or manage.

The following best practices apply to IAM policies that control access to AWS KMS keys and aliases. For general IAM policy best practice guidance, see Security best practices in IAM in the *IAM User Guide*.

Use key policies

Whenever possible, provide permissions in key policies that affect one KMS key, rather than in an IAM policy that can apply to many KMS keys, including those in other AWS accounts. This is particularly important for sensitive permissions like kms:PutKeyPolicy and kms:ScheduleKeyDeletion but also for cryptographic operations that determine how your data is protected.

Limit CreateKey permission

Give permission to create keys (kms:CreateKey) only to principals who need it. Principals who create a KMS key also set its key policy, so they can give themselves and others permission to use and manage the KMS keys they create. When you allow this permission, consider limiting it by using policy conditions. For example, you can use the kms:KeySpec condition to limit the permission to symmetric encryption KMS keys.

Specify KMS keys in an IAM policy

As a best practice, specify the key ARN of each KMS key to which the permission applies in the `Resource` element of the policy statement. This practice restricts the permission to the KMS keys that principal requires. For example, this `Resource` element lists only the KMS keys the principal needs to use.

```
"Resource": [
    "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321"
]
```

When specifying KMS keys is impractical, use a `Resource` value that limits access to KMS keys in a trusted AWS account and Region, such as `arn:aws:kms:*region*:*account*:key/*`. Or limit access to KMS keys in all Regions (*) of a trusted AWS account, such as `arn:aws:kms:*:*account*:key/*`.

You cannot use a key ID, alias name, or alias ARN to represent a KMS key in the `Resource` field of an IAM policy. If you specify an alias ARN, the policy applies to the alias, not to the KMS key. For information about IAM policies for aliases, see Controlling access to aliases

Avoid "Resource": "*" in an IAM policy

Use wildcard characters (*) judiciously. In a key policy, the wildcard character in the `Resource` element represents the KMS key to which the key policy is attached. But in an IAM policy, a wildcard character alone in the `Resource` element (`"Resource": "*"`) applies the permissions to all KMS keys in all AWS accounts that the principal's account has permission to use. This might include KMS keys in other AWS accounts, as well as KMS keys in the principal's account.

For example, to use a KMS key in another AWS account, a principal needs permission from the key policy of the KMS key in the external account, and from an IAM policy in their own account. Suppose that an arbitrary account gave your AWS account kms:Decrypt permission on their KMS keys. If so, an IAM policy in your account that gives a role `kms:Decrypt` permission on all KMS keys (`"Resource": "*"`) would satisfy the IAM part of the requirement. As a result, principals who can assume that role can now decrypt ciphertexts using the KMS key in the untrusted account. Entries for their operations appear in the CloudTrail logs of both accounts.

In particular, avoid using `"Resource": "*"` in a policy statement that allows the following API operations. These operations can be called on KMS keys in other AWS accounts.

- DescribeKey
- GetKeyRotationStatus
- Cryptographic operations (Encrypt, Decrypt, GenerateDataKey, GenerateDataKeyPair, GenerateDataKeyWithoutPlaintext, GenerateDataKeyPairWithoutPlaintext, GetPublicKey, ReEncrypt, Sign, Verify)
- CreateGrant, ListGrants, ListRetirableGrants, RetireGrant, RevokeGrant

When to use "Resource": "*"

In an IAM policy, use a wildcard character in the `Resource` element only for permissions that require it. Only the following permissions require the `"Resource": "*"` element.

- kms:CreateKey
- kms:GenerateRandom
- kms:ListAliases
- kms:ListKeys
- Permissions for custom key stores, such as kms:CreateCustomKeyStore and kms:ConnectCustomKeyStore.

> **Note**
>
> Permissions for alias operations (kms:CreateAlias, kms:UpdateAlias, kms:DeleteAlias) must be attached to the alias and the KMS key. You can use `"Resource": "*"` in an IAM policy to represent the aliases and the KMS keys, or specify the aliases and KMS keys in the `Resource` element. For examples, see Controlling access to aliases.

The examples in this topic provide more information and guidance for designing IAM policies for KMS keys. For general AWS KMS best practice guidance, see the AWS Key Management Service Best Practices (PDF). For IAM best practices for all AWS resources, see Security best practices in IAM in the *IAM User Guide*.

Specifying KMS keys in IAM policy statements

You can use an IAM policy to allow a principal to use or manage KMS keys. KMS keys are specified in the `Resource` element of the policy statement.

- To specify a KMS key in an IAM policy statement, you must use its key ARN. You cannot use a key id, alias name, or alias ARN to identify a KMS key in an IAM policy statement.

  For example: `"Resource": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"`

  To control access to a KMS key based on its aliases, use the kms:RequestAlias or kms:ResourceAliases condition keys. For details, see ABAC for AWS KMS.

Use an alias ARN as the resource only in a policy statement that controls access to alias operations, such as CreateAlias, UpdateAlias, or DeleteAlias. For details, see Controlling access to aliases.

- To specify multiple KMS keys in the account and Region, use wildcard characters (*) in the Region or resource ID positions of the key ARN.

  For example, to specify all KMS keys in the US West (Oregon) Region of an account, use `"Resource": "arn:aws:kms:us-west-2:111122223333:key/*"`. To specify all KMS keys in all Regions of the account, use `"Resource": "arn:aws:kms:*:111122223333:key/*"`.
- To represent all KMS keys, use a wildcard character alone (`"*"`). Use this format for operations that don't use any particular KMS key, namely CreateKey, GenerateRandom, ListAliases, and ListKeys.

When writing your policy statements, it's a best practice to specify only the KMS keys that the principal needs to use, rather than giving them access to all KMS keys.

For example, the following IAM policy statement allows the principal to call the DescribeKey, GenerateDataKey, Decrypt operations only on the KMS keys listed in the `Resource` element of the policy statement. Specifying KMS keys by key ARN, which is a best practice, ensures that the permissions are limited only to the specified KMS keys.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:DescribeKey",
      "kms:GenerateDataKey",
      "kms:Decrypt"
    ],
    "Resource": [
      "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321"
    ]
  }
}
```

To apply the permission to all KMS keys in a particular trusted AWS account, you can use wildcard characters (*) in the Region and key ID positions. For example, the following policy statement allows the principal to call the specified operations on all KMS keys in two trusted example accounts.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:DescribeKey",
      "kms:GenerateDataKey",
      "kms:GenerateDataKeyPair"
    ],
    "Resource": [
      "arn:aws:kms:*:111122223333:key/*",
      "arn:aws:kms:*:444455556666:key/*"
    ]
  }
}
```

You can also use a wildcard character (`"*"`) alone in the `Resource` element. Because it allows access to all KMS keys the account has permission to use, it's recommended primarily for operations without a particular KMS key and for `Deny` statements. You can also use it in policy statements that allow only less sensitive read-only operations. To determine whether an AWS KMS operation involves a particular KMS key, look for the **KMS key** value in the **Resources** column of the table in the section called “Permissions reference”.

For example, the following policy statement uses a `Deny` effect to prohibit the principals from using the specified operations on any KMS key. It uses a wildcard character in the `Resource` element to represent all KMS keys.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Deny",
    "Action": [
      "kms:CreateKey",
      "kms:PutKeyPolicy",
      "kms:CreateGrant",
      "kms:ScheduleKeyDeletion"
    ],
    "Resource": "*"
  }
}
```

The following policy statement uses a wildcard character alone to represent all KMS keys. But it allows only less sensitive read-only operations and operations that don't apply to any particular KMS key.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:CreateKey",
      "kms:ListKeys",
      "kms:ListAliases",
      "kms:ListResourceTags"
    ],
    "Resource": "*"
  }
}
```

Permissions required to use the AWS KMS console

To work with the AWS KMS console, users must have a minimum set of permissions that allow them to work with the AWS KMS resources in their AWS account. In addition to these AWS KMS permissions, users must also have permissions to list IAM users and IAM roles. If you create an IAM policy that is more restrictive than the minimum required permissions, the AWS KMS console won't function as intended for users with that IAM policy.

For the minimum permissions required to allow a user read-only access to the AWS KMS console, see Allow a user to view KMS keys in the AWS KMS console.

To allow users to work with the AWS KMS console to create and manage KMS keys, attach the **AWSKeyManagementServicePowerUser** managed policy to the user, as described in the following section.

You don't need to allow minimum console permissions for users that are working with the AWS KMS API through the AWS SDKs, AWS Command Line Interface or AWS Tools for PowerShell.

However, you do need to grant these users permission to use the API. For more information, see Permissions reference.

AWS managed policy for power users

You can use the `AWSKeyManagementServicePowerUser` managed policy to give IAM principals in your account the permissions of a power user. Power users can create KMS keys, use and manage the KMS keys they create, and view all KMS keys and IAM identities. Principals who have the `AWSKeyManagementServicePowerUser` managed policy can also get permissions from other sources, including key policies, other IAM policies, and grants.

`AWSKeyManagementServicePowerUser` is an AWS managed IAM policy. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

> **Note**
>
> Permissions in this policy that are specific to a KMS key, such as `kms:TagResource` and `kms:GetKeyRotationStatus`, are effective only when the key policy for that KMS key explicitly allows the AWS account to use IAM policies to control access to the key. To determine whether a permission is specific to a KMS key, see AWS KMS permissions and look for a value of **KMS key** in the **Resources** column.
> This policy gives a power user permissions on any KMS key with a key policy that permits the operation. For cross-account permissions, such as `kms:DescribeKey` and `kms:ListGrants`, this might include KMS keys in untrusted AWS accounts. For details, see Best practices for IAM policies and Allowing users in other accounts to use a KMS key. To determine whether a permission is valid on KMS keys in other accounts, see AWS KMS permissions and look for a value of **Yes** in the **Cross-account use** column.
> To allow principals to view the AWS KMS console without errors, the principal needs the tag:GetResources permission, which is not included in the `AWSKeyManagementServicePowerUser` policy. You can allow this permission in a separate IAM policy.

The AWSKeyManagementServicePowerUser managed IAM policy includes the following permissions.

- Allows principals to create KMS keys. Because this process includes setting the key policy, power users can give themselves and others permission to use and manage the KMS keys they create.

- Allows principals to create and delete aliases and tags on all KMS keys. Changing a tag or alias can allow or deny permission to use and manage the KMS key. For details, see ABAC for AWS KMS.
- Allows principals to get detailed information about all KMS keys, including their key ARN, cryptographic configuration, key policy, aliases, tags, and rotation status.
- Allows principals to list IAM users, groups, and roles.
- This policy does not allow principals to use or manage KMS keys that they didn't create. However, they can change aliases and tags on all KMS keys, which might allow or deny them permission to use or manage a KMS key.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:CreateAlias",
                "kms:CreateKey",
                "kms:DeleteAlias",
                "kms:Describe*",
                "kms:GenerateRandom",
                "kms:Get*",
                "kms:List*",
                "kms:TagResource",
                "kms:UntagResource",
                "iam:ListGroups",
                "iam:ListRoles",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

IAM policy examples

In this section, you can find example IAM policies that allow permissions for various AWS KMS actions.

⚠ **Important**

Some of the permissions in the following policies are allowed only when the KMS key's key policy also allows them. For more information, see Permissions reference.

For help writing and formatting a JSON policy document, see the IAM JSON Policy Reference in the *IAM User Guide*.

Examples

- Allow a user to view KMS keys in the AWS KMS console
- Allow a user to create KMS keys
- Allow a user to encrypt and decrypt with any KMS key in a specific AWS account
- Allow a user to encrypt and decrypt with any KMS key in a specific AWS account and Region
- Allow a user to encrypt and decrypt with specific KMS keys
- Prevent a user from disabling or deleting any KMS keys

Allow a user to view KMS keys in the AWS KMS console

The following IAM policy allows users read-only access to the AWS KMS console. Users with these permissions can view all KMS keys in their AWS account, but they cannot create or change any KMS keys.

To view KMS keys on the **AWS managed keys** and **Customer managed keys** pages, principals require kms:ListKeys, kms:ListAliases, and tag:GetResources permissions, even if the keys do not have tags or aliases. The remaining permissions, particularly kms:DescribeKey, are required to view optional KMS key table columns and data on the KMS key detail pages. The iam:ListUsers and iam:ListRoles permissions are required to display the key policy in default view without error. To view data on the **Custom key stores** page and details about KMS keys in custom key stores, principals also need kms:DescribeCustomKeyStores permission.

If you limit a user's console access to particular KMS keys, the console displays an error for each KMS key that is not visible.

This policy includes of two policy statements. The `Resource` element in the first policy statement allows the specified permissions on all KMS keys in all Regions of the example AWS account.

Console viewers don't need additional access because the AWS KMS console displays only KMS keys in the principal's account. This is true even if they have permission to view KMS keys in other AWS accounts. The remaining AWS KMS and IAM permissions require a `"Resource": "*"` element because they don't apply to any particular KMS key.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadOnlyAccessForAllKMSKeysInAccount",
      "Effect": "Allow",
      "Action": [
        "kms:GetPublicKey",
        "kms:GetKeyRotationStatus",
        "kms:GetKeyPolicy",
        "kms:DescribeKey",
        "kms:ListKeyPolicies",
        "kms:ListResourceTags",
        "tag:GetResources"
      ],
      "Resource": "arn:aws:kms:*:111122223333:key/*"
    },
    {
      "Sid": "ReadOnlyAccessForOperationsWithNoKMSKey",
      "Effect": "Allow",
      "Action": [
        "kms:ListKeys",
        "kms:ListAliases",
        "iam:ListRoles",
        "iam:ListUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

Allow a user to create KMS keys

The following IAM policy allows a user to create all types of KMS keys. The value of the `Resource` element is `*` because the `CreateKey` operation does not use any particular AWS KMS resources (KMS keys or aliases).

To restrict the user to particular types of KMS keys, use the kms:KeySpec, kms:KeyUsage, and kms:KeyOrigin condition keys.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "kms:CreateKey",
    "Resource": "*"
  }
}
```

Principals who create keys might need some related permissions.

- **kms:PutKeyPolicy** — Principals who have `kms:CreateKey` permission can set the initial key policy for the KMS key. However, the `CreateKey` caller must have kms:PutKeyPolicy permission, which lets them change the KMS key policy, or they must specify the `BypassPolicyLockoutSafetyCheck` parameter of `CreateKey`, which is not recommended. The `CreateKey` caller can get `kms:PutKeyPolicy` permission for the KMS key from an IAM policy or they can include this permission in the key policy of the KMS key that they're creating.
- **kms:TagResource** — To add tags to the KMS key during the `CreateKey` operation, the `CreateKey` caller must have kms:TagResource permission in an IAM policy. Including this permission in the key policy of the new KMS key isn't sufficient. However, if the `CreateKey` caller includes `kms:TagResource` in the initial key policy, they can add tags in a separate call after the KMS key is created.
- **kms:CreateAlias** — Principals who create a KMS key in the AWS KMS console must have kms:CreateAlias permission on the KMS key and on the alias. (The console makes two calls; one to `CreateKey` and one to `CreateAlias`). You must provide the alias permission in an IAM policy. You can provide the KMS key permission in a key policy or IAM policy. For details, see Controlling access to aliases.

In addition to `kms:CreateKey`, the following IAM policy provides `kms:TagResource` permission on all KMS keys in the AWS account and `kms:CreateAlias` permission on all aliases that the account. It also includes some useful read-only permissions that can be provided only in an IAM policy.

This IAM policy does not include `kms:PutKeyPolicy` permission or any other permissions that can be set in a key policy. It's a best practice to set these permissions in the key policy where they apply exclusively to one KMS key.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "IAMPermissionsForParticularKMSKeys",
      "Effect": "Allow",
      "Action": "kms:TagResource",
      "Resource": "arn:aws:kms:*:111122223333:key/*"
    },
    {
      "Sid": "IAMPermissionsForParticularAliases",
      "Effect": "Allow",
      "Action": "kms:CreateAlias",
      "Resource": "arn:aws:kms:*:111122223333:alias/*"
    },
    {
      "Sid": "IAMPermissionsForAllKMSKeys",
      "Effect": "Allow",
      "Action": [
        "kms:CreateKey",
        "kms:ListKeys",
        "kms:ListAliases"
      ],
      "Resource": "*"
    }
  ]
}
```

Allow a user to encrypt and decrypt with any KMS key in a specific AWS account

The following IAM policy allows a user to encrypt and decrypt data with any KMS key in AWS account 111122223333.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:Encrypt",
      "kms:Decrypt"
    ],
    "Resource": "arn:aws:kms:*:111122223333:key/*"
  }
}
```

Allow a user to encrypt and decrypt with any KMS key in a specific AWS account and Region

The following IAM policy allows a user to encrypt and decrypt data with any KMS key in AWS account 111122223333 in the US West (Oregon) Region.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:Encrypt",
      "kms:Decrypt"
    ],
    "Resource": [
      "arn:aws:kms:us-west-2:111122223333:key/*"
    ]
  }
}
```

Allow a user to encrypt and decrypt with specific KMS keys

The following IAM policy allows a user to encrypt and decrypt data with the two KMS keys specified in the `Resource` element. When specifying a KMS key in an IAM policy statement, you must use the key ARN of the KMS key.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:Encrypt",
      "kms:Decrypt"
    ],
    "Resource": [
      "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321"
    ]
  }
}
```

Prevent a user from disabling or deleting any KMS keys

The following IAM policy prevents a user from disabling or deleting any KMS keys, even when another IAM policy or a key policy allows these permissions. A policy that explicitly denies permissions overrides all other policies, even those that explicitly allow the same permissions. For more information, see Troubleshooting key access.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Deny",
    "Action": [
      "kms:DisableKey",
      "kms:ScheduleKeyDeletion"
    ],
    "Resource": "*"
  }
}
```

Grants in AWS KMS

A *grant* is a policy instrument that allows AWS principals to use KMS keys in cryptographic operations. It also can let them view a KMS key (`DescribeKey`) and create and manage grants. When authorizing access to a KMS key, grants are considered along with key policies and IAM policies. Grants are often used for temporary permissions because you can create one, use its permissions, and delete it without changing your key policies or IAM policies.

Grants are commonly used by AWS services that integrate with AWS KMS to encrypt your data at rest. The service creates a grant on behalf of a user in the account, uses its permissions, and retires the grant as soon as its task is complete. For details about how AWS services, use grants, see How AWS services use AWS KMS or the *Encryption at rest* topic in the service's user guide or developer guide.

For code examples that demonstrate how to work with grants in several programming languages, see Working with grants.

Topics

- About grants
- Grant concepts
- Best practices for AWS KMS grants
- Creating grants
- Managing grants

About grants

Grants are a very flexible and useful access control mechanism. When you create a grant for a KMS key, the grant allows the grantee principal to call the specified grant operations on the KMS key provided that all conditions specified in the grant are met.

- Each grant allows access to exactly one KMS key. You can create a grant for a KMS key in a different AWS account.
- A grant can allow access to a KMS key, but not deny access.
- Each grant has one grantee principal. The grantee principal can represent one or more identities in the same AWS account as the KMS key or in a different account.
- A grant can only allow grant operations. The grant operations must be supported by the KMS key in the grant. If you specify an unsupported operation, the CreateGrant request fails with a `ValidationError` exception.
- The grantee principal can use the permissions that the grant gives them without specifying the grant, just as they would if the permissions came from a key policy or IAM policy. However, because the AWS KMS API follows an eventual consistency model, when you create, retire, or revoke a grant, there might be a brief delay, before the change is available throughout AWS KMS. To use the permissions in a grant immediately, use a grant token.
- An authorized principal can delete the grant (retire or revoke it). Deleting a grant eliminates all permissions that the grant allowed. You do not have to figure out which policies to add or remove to undo the grant.
- AWS KMS limits the number of grants on each KMS key. For details, see Grants per KMS key: 50,000.

Be cautious when creating grants and when giving others permission to create grants. Permission to create grants has security implications, much like allowing the kms:PutKeyPolicy permission to set policies.

- Users with permission to create grants for a KMS key (`kms:CreateGrant`) can use a grant to allow users and roles, including AWS services, to use the KMS key. The principals can be identities in your own AWS account or identities in a different account or organization.
- Grants can allow only a subset of AWS KMS operations. You can use grants to allow principals to view the KMS key, use it in cryptographic operations, and create and retire grants. For details, see Grant operations. You can also use grant constraints to limit the permissions in a grant for a symmetric encryption key.
- Principals can get permission to create grants from a key policy or IAM policy. Principals who get `kms:CreateGrant` permission from a policy can create grants for any grant operation on the KMS key. These principals are not required to have the permission that they are granting on the key. When you allow `kms:CreateGrant` permission in a policy, you can use policy conditions to limit this permission.
- Principals can also get permission to create grants from a grant. These principal can only delegate the permissions that they were granted, even if they have other permissions from a policy. For details, see Granting CreateGrant permission.

For help with concepts related to grants, see Grant terminology.

Grant concepts

To use grants effectively, you'll need to understand the terms and concepts that AWS KMS uses.

Grant constraint

A condition that limits the permissions in the grant. Currently, AWS KMS supports grant constraints based on the encryption context in the request for a cryptographic operation. For details, see Using grant constraints.

Grant ID

The unique identifier of a grant for a KMS key. You can use a grant ID, along with a key identifier, to identify a grant in a RetireGrant or RevokeGrant request.

Grant operations

The AWS KMS operations that you can allow in a grant. If you specify other operations, the CreateGrant request fails with a `ValidationError` exception. These are also the operations that accept a grant token. For detailed information about these permissions, see the AWS KMS permissions.

These grant operations actually represent permission to use the operation. Therefore, for the `ReEncrypt` operation, you can specify `ReEncryptFrom`, `ReEncryptTo`, or both `ReEncrypt*`.

The grant operations are:

- Cryptographic operations
  - Decrypt
  - DeriveSharedSecret
  - Encrypt
  - GenerateDataKey
  - GenerateDataKeyPair
  - GenerateDataKeyPairWithoutPlaintext
  - GenerateDataKeyWithoutPlaintext
  - GenerateMac
  - ReEncryptFrom
  - ReEncryptTo
  - Sign
  - Verify
  - VerifyMac
- Other operations
  - CreateGrant
  - DescribeKey
  - GetPublicKey
  - RetireGrant

The grant operations that you allow must be supported by the KMS key in the grant. If you specify an unsupported operation, the CreateGrant request fails with a `ValidationError` exception. For example, grants for symmetric encryption KMS keys cannot allow the Sign, Verify, `GenerateMac` or `VerifyMac` operations. Grants for asymmetric KMS keys cannot allow any operations that generate data keys or data key pairs.

Grant token

The AWS KMS API follows an eventual consistency model. When you create a grant, there might be a brief delay before the change is available throughout AWS KMS. It typically takes less than a few seconds for the change to propagate throughout the system, but in some cases it can take several minutes. If you try to use a grant before it fully propagates through the system, you might get an access denied error. A grant token lets you refer to the grant and use the grant permissions immediately.

A *grant token* is a unique, nonsecret, variable-length, base64-encoded string that represents a grant. You can use the grant token to identify the grant in any grant operation. However, because the token value is a hash digest, it doesn't reveal any details about the grant.

A grant token is designed to be used only until the grant has fully propagated throughout AWS KMS. After that, the grantee principal can use the permission in the grant without providing a grant token or any other evidence of the grant. You can use a grant token at any time, but once the grant is eventually consistent, AWS KMS uses the grant to determine permissions, not the grant token.

For example, the following command calls the GenerateDataKey operation. It uses a grant token to represent the grant that gives the caller (the grantee principal) permission to call `GenerateDataKey` on the specified KMS key.

```
$ aws kms generate-data-key \
        --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
        --key-spec AES_256 \
        --grant-token $token
```

You can also use a grant token to identify a grant in operations that manage grants. For example, the retiring principal can use a grant token in a call to the RetireGrant operation.

```
$ aws kms retire-grant \
        --grant-token $token
```

`CreateGrant` is the only operation that returns a grant token. You cannot get a grant token from any other AWS KMS operation or from the CloudTrail log event for the CreateGrant operation. The ListGrants and ListRetirableGrants operations return the grant ID, but not a grant token.

For details, see Using a grant token.

Grantee principal

The identities that get the permissions specified in the grant. Each grant has one grantee principal, but the grantee principal can represent multiple identities.

The grantee principal can be any AWS principal, including an AWS account (root), an IAM user, an IAM role, a federated role or user, or an assumed role user. The grantee principal can be in the same account as the KMS key or a different account. However, the grantee principal cannot be a service principal, an IAM group, or an AWS organization.

> **Note**
>
> IAM best practices discourage the use of IAM users with long-term credentials. Whenever possible, use IAM roles, which provide temporary credentials. For details, see Security best practices in IAM in the *IAM User Guide*.

Retire (a grant)

Terminates a grant. You retire a grant when you finish using the permissions.

Revoking and retiring a grant both delete the grant. But retiring is done by a principal specified in the grant. Revoking is typically done by a key administrator. For details, see Retiring and revoking grants.

Retiring principal

A principal who can retire a grant. You can specify a retiring principal in a grant, but it is not required. The retiring principal can be any AWS principal, including AWS accounts, IAM users, IAM roles, federated users, and assumed role users. The retiring principal can be in the same account as the KMS key or a different account.

> **Note**
>
> IAM best practices discourage the use of IAM users with long-term credentials. Whenever possible, use IAM roles, which provide temporary credentials. For details, see Security best practices in IAM in the *IAM User Guide*.

In addition to retiring principal specified in the grant, a grant can be retired by the AWS account in which the grant was created. If the grant allows the `RetireGrant` operation, the grantee principal can retire the grant. Also, the AWS account or an AWS account that is the retiring principal can delegate the permission to retire a grant to an IAM principal in the same AWS account. For details, see Retiring and revoking grants.

Revoke (a grant)

Terminates a grant. You revoke a grant to actively deny the permissions that the grant allows.

Revoking and retiring a grant both delete the grant. But retiring is done by a principal specified in the grant. Revoking is typically done by a key administrator. For details, see Retiring and revoking grants.

Eventual consistency (for grants)

The AWS KMS API follows an eventual consistency model. When you create, retire, or revoke a grant, there might be a brief delay before the change is available throughout AWS KMS. It typically takes less than a few seconds for the change to propagate throughout the system, but in some cases it can take several minutes.

You might become aware of this brief delay if you get unexpected errors. For example, If you try to manage a new grant or use the permissions in a new grant before the grant is known throughout AWS KMS, you might get an access denied error. If you retire or revoke a grant, the grantee principal might still be able to use its permissions for a brief period until the grant is fully deleted. The typical strategy is to retry the request, and some AWS SDKs include automatic backoff and retry logic.

AWS KMS has features to mitigate this brief delay.

- To use the permissions in a new grant immediately, use a grant token. You can use a grant token to refer to a grant in any grant operation. For instructions, see Using a grant token.
- The CreateGrant operation has a `Name` parameter that prevents retry operations from creating duplicate grants.

> **Note**
>
> Grant tokens supersede the validity of the grant until all endpoints in the service have been updated with the new grant state. In most cases, eventual consistency will be achieved within five minutes.

For more information, see AWS KMS eventual consistency.

Best practices for AWS KMS grants

AWS KMS recommends the following best practices when creating, using, and managing grants.

- Limit the permissions in the grant to those that the grantee principal requires. Use the principle of least privileged access.
- Use a specific grantee principal, such as an IAM role, and give the grantee principal permission to use only the API operations that they require.
- Use the encryption context grant constraints to ensure that callers are using the KMS key for the intended purpose. For details about how to use the encryption context in a request to secure your data, see How to Protect the Integrity of Your Encrypted Data by Using AWS Key Management Service and EncryptionContext in the *AWS Security Blog*.

> **Tip**
>
> Use the EncryptionContextEqual grant constraint whenever possible. The EncryptionContextSubset grant constraint is more difficult to use correctly. If you need to use it, read the documentation carefully and test the grant constraint to make sure it works as intended.

- Delete duplicate grants. Duplicate grants have the same key ARN, API actions, grantee principal, encryption context, and name. If you retire or revoke the original grant but leave the duplicates, the leftover duplicate grants constitute unintended escalations of privilege. To avoid duplicating grants when retrying a `CreateGrant` request, use the Name parameter. To detect duplicate grants, use the ListGrants operation. If you accidentally create a duplicate grant, retire or revoke it as soon as possible.

> **Note**
>
> Grants for AWS managed keys might look like duplicates but have different grantee principals.
> The `GranteePrincipal` field in the `ListGrants` response usually contains the grantee principal of the grant. However, when the grantee principal in the grant is an AWS service, the `GranteePrincipal` field contains the service principal, which might represent several different grantee principals.

- Remember that grants do not automatically expire. Retire or revoke the grant as soon as the permission is no longer needed. Grants that are not deleted might create a security risk for encrypted resources.

Creating grants

Before creating a grant, learn about the options for customizing your grant. You can use *grant constraints* to limit the permissions in the grant. Also, learn about granting `CreateGrant` permission. Principals who get permission to create grants from a grant are limited in the grants that they can create.

Topics

- Creating a grant
- Using grant constraints
- Granting CreateGrant permission

Creating a grant

To create a grant, call the CreateGrant operation. Specify a KMS key, a grantee principal, and a list of allowed grant operations. You can also designate an optional retiring principal. To customize the grant, use optional `Constraints` parameters to define grant constraints.

When you create, retire, or revoke a grant, there might be a brief delay, usually less than five minutes, before the change is available throughout AWS KMS. For more information, see Eventual consistency (for grants).

For example, the following CreateGrant command creates a grant that allows users who are authorized to assume the keyUserRole role to call the Decrypt operation on the specified symmetric KMS key. The grant uses the RetiringPrincipal parameter to designate a principal that can retire the grant. It also includes a grant constraint that allows the permission only when the encryption context in the request includes "Department": "IT".

```
$  aws kms create-grant \
    --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
    --grantee-principal arn:aws:iam::111122223333:role/keyUserRole \
    --operations Decrypt \
    --retiring-principal arn:aws:iam::111122223333:role/adminRole \
    --constraints EncryptionContextSubset={Department=IT}
```

If your code retries the CreateGrant operation, or uses an AWS SDK that automatically retries requests, use the optional Name parameter to prevent the creation of duplicate grants. If AWS KMS gets a CreateGrant request for a grant with the same properties as an existing grant, including the name, it recognizes the request as a retry, and does not create a new grant. You cannot use the Name value to identify the grant in any AWS KMS operations.

Important

Do not include confidential or sensitive information in the grant name. It may appear in plain text in CloudTrail logs and other output.

```
$ aws kms create-grant \
    --name IT-1234abcd-keyUserRole-decrypt \
    --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
    --grantee-principal arn:aws:iam::111122223333:role/keyUserRole \
    --operations Decrypt \
    --retiring-principal arn:aws:iam::111122223333:role/adminRole \
    --constraints EncryptionContextSubset={Department=IT}
```

For code examples that demonstrate how to work with grants in several programming languages, see Working with grants.

Using grant constraints

Grant constraints set conditions on the permissions that the grant gives to the grantee principal. Grant constraints take the place of condition keys in a key policy or IAM policy. Each grant constraint value can include up to 8 encryption context pairs. The encryption context value in each grant constraint cannot exceed 384 characters.

⚠ **Important**

Do not include confidential or sensitive information in this field. This field may be displayed in plaintext in CloudTrail logs and other output.

AWS KMS supports two grant constraints, `EncryptionContextEquals` and `EncryptionContextSubset`, both of which establish requirements for the encryption context in a request for a cryptographic operation.

The encryption context grant constraints are designed to be used with grant operations that have an encryption context parameter.

- Encryption context constraints are valid only in a grant for a symmetric encryption KMS key. Cryptographic operations with other KMS keys don't support an encryption context.
- The encryption context constraint is ignored for `DescribeKey` and `RetireGrant` operations. `DescribeKey` and `RetireGrant` don't have an encryption context parameter, but you can include these operations in a grant that has an encryption context constraint.
- You can use an encryption context constraint in a grant for the `CreateGrant` operation. The encryption context constraint requires that any grants created with the `CreateGrant` permission have an equally strict or stricter encryption context constraint.

AWS KMS supports the following encryption context grant constraints.

EncryptionContextEquals

Use `EncryptionContextEquals` to specify the exact encryption context for permitted requests.

`EncryptionContextEquals` requires that the encryption context pairs in the request are an exact, case-sensitive match for the encryption context pairs in the grant constraint. The pairs can appear in any order, but the keys and values in each pair cannot vary.

For example, if the `EncryptionContextEquals` grant constraint requires the `"Department": "IT"` encryption context pair, the grant allows requests of the specified type only when the encryption context in the request is exactly `"Department": "IT"`.

EncryptionContextSubset

Use `EncryptionContextSubset` to require that requests include particular encryption context pairs.

`EncryptionContextSubset` requires that the request include all encryption context pairs in the grant constraint (an exact, case-sensitive match), but the request can also have additional encryption context pairs. The pairs can appear in any order, but the keys and values in each pair cannot vary.

For example, if the `EncryptionContextSubset` grant constraint requires the `Department=IT` encryption context pair, the grant allows requests of the specified type when the encryption context in the request is `"Department": "IT"`, or includes `"Department": "IT"` along with other encryption context pairs, such as `"Department": "IT","Purpose": "Test"`.

To specify an encryption context constraint in a grant for a symmetric encryption KMS key, use the `Constraints` parameter in the CreateGrant operation. The grant that this command creates gives users who are authorized to assume the `keyUserRole` role permission to call the Decrypt operation. But that permission is effective only when the encryption context in the `Decrypt` request is a `"Department": "IT"` encryption context pair.

```
$ aws kms create-grant \
    --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
    --grantee-principal arn:aws:iam::111122223333:role/keyUserRole \
    --operations Decrypt \
    --retiring-principal arn:aws:iam::111122223333:role/adminRole \
    --constraints EncryptionContextEquals={Department=IT}
```

The resulting grant looks like the following one. Notice that the permission granted to the `keyUserRole` role is effective only when the `Decrypt` request uses the same encryption context pair specified in the grant constraint. To find the grants on a KMS key, use the ListGrants operation.

```
$ aws kms list-grants --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
{
    "Grants": [
        {
            "Name": "",
            "IssuingAccount": "arn:aws:iam::111122223333:root",
            "GrantId": 
 "abcde1237f76e4ba7987489ac329fbfba6ad343d6f7075dbd1ef191f0120514a",
            "Operations": [
                "Decrypt"
            ],
            "GranteePrincipal": "arn:aws:iam::111122223333:role/keyUserRole",
            "Constraints": {
                "EncryptionContextEquals": {
                    "Department": "IT"
                }
            },
            "CreationDate": 1568565290.0,
            "KeyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "RetiringPrincipal": "arn:aws:iam::111122223333:role/adminRole"
        }
    ]
}
```

To satisfy the `EncryptionContextEquals` grant constraint, the encryption context in the request for the `Decrypt` operation must be a `"Department": "IT"` pair. A request like the following from the grantee principal would satisfy the `EncryptionContextEquals` grant constraint.

```
$ aws kms decrypt \
    --key-id arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab\
    --ciphertext-blob fileb://encrypted_msg \
    --encryption-context Department=IT
```

When the grant constraint is `EncryptionContextSubset`, the encryption context pairs in the request must include the encryption context pairs in the grant constraint, but the request can also include other encryption context pairs. The following grant constraint requires that one of encryption context pairs in the request is `"Deparment": "IT"`.

```
"Constraints": {
   "EncryptionContextSubset": {
       "Department": "IT"
   }
}
```

The following request from the grantee principal would satisfy both of the `EncryptionContextEqual` and `EncryptionContextSubset` grant constraints in this example.

```
$ aws kms decrypt \
    --key-id arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab \
    --ciphertext-blob fileb://encrypted_msg \
    --encryption-context Department=IT
```

However, a request like the following from the grantee principal would satisfy the `EncryptionContextSubset` grant constraint, but it would fail the `EncryptionContextEquals` grant constraint.

```
$ aws kms decrypt \
    --key-id arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab \
    --ciphertext-blob fileb://encrypted_msg \
    --encryption-context Department=IT,Purpose=Test
```

AWS services often use encryption context constraints in the grants that give them permission to use KMS keys in your AWS account. For example, Amazon DynamoDB uses a grant like the following one to get permission to use the AWS managed key for DynamoDB in your account. The `EncryptionContextSubset` grant constraint in this grant makes the permissions in the grant effective only when the encryption context in the request includes `"subscriberID": "111122223333"` and `"tableName": "Services"` pairs. This grant constraint means that the grant allows DynamoDB to use the specified KMS key only for a particular table in your AWS account.

To get this output, run the ListGrants operation on the AWS managed key for DynamoDB in your account.

```
$ aws kms list-grants --key-id 0987dcba-09fe-87dc-65ba-ab0987654321

{
    "Grants": [
        {
            "Operations": [
                "Decrypt",
                "Encrypt",
                "GenerateDataKey",
                "ReEncryptFrom",
                "ReEncryptTo",
                "RetireGrant",
                "DescribeKey"
            ],
            "IssuingAccount": "arn:aws:iam::111122223333:root",
            "Constraints": {
                "EncryptionContextSubset": {
                    "aws:dynamodb:tableName": "Services",
                    "aws:dynamodb:subscriberId": "111122223333"
                }
            },
            "CreationDate": 1518567315.0,
            "KeyId": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321",
            "GranteePrincipal": "dynamodb.us-west-2.amazonaws.com",
            "RetiringPrincipal": "dynamodb.us-west-2.amazonaws.com",
            "Name": "8276b9a6-6cf0-46f1-b2f0-7993a7f8c89a",
            "GrantId":
 "1667b97d27cf748cf05b487217dd4179526c949d14fb3903858e25193253fe59"
        }
    ]
}
```

Granting CreateGrant permission

A grant can include permission to call the `CreateGrant` operation. But when a grantee principal gets permission to call `CreateGrant` from a grant, rather than from a policy, that permission is limited.

- The grantee principal can only create grants that allow some or all of the operations in the parent grant.
- The grant constraints in the grants they create must be at least as strict as those in the parent grant.

These limitations don't apply to principals who get `CreateGrant` permission from a policy, although their permissions can be limited by policy conditions.

For example, consider a grant that allows the grantee principal to call the `GenerateDataKey`, `Decrypt`, and `CreateGrant` operations. We call a grant that allow `CreateGrant` permission a *parent grant*.

```
# The original grant in a ListGrants response.
{
    "Grants": [
        {
            "KeyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "CreationDate": 1572216195.0,
            "GrantId":
 "abcde1237f76e4ba7987489ac329fbfba6ad343d6f7075dbd1ef191f0120514a",
            "Operations": [
                "GenerateDataKey",
                "Decrypt",
                "CreateGrant
            ]
            "RetiringPrincipal": "arn:aws:iam::111122223333:role/adminRole",
            "Name": "",
            "IssuingAccount": "arn:aws:iam::111122223333:root",
            "GranteePrincipal": "arn:aws:iam::111122223333:role/keyUserRole",
            "Constraints": {
                "EncryptionContextSubset": {
                    "Department": "IT"
                }
            },
        }
    ]
}
```

The grantee principal, exampleUser, can use this permission to create a grant that includes any subset of the operations specified in the original grant, such as `CreateGrant` and `Decrypt`. The *child grant* cannot include other operations, such as `ScheduleKeyDeletion` or `ReEncrypt`.

Also, the grant constraints in child grants must be as restrictive or more restrictive than those in the parent grant. For example, the child grant can add pairs to an `EncryptionContextSubset` constraint in the parent grant, but it cannot remove them. The child grant can change an `EncryptionContextSubset` constraint to an `EncryptionContextEquals` constraint, but not the reverse.

For example, the grantee principal can use the `CreateGrant` permission that it got from the parent grant to create the following child grant. The operations in the child grant are a subset of the operations in the parent grant and the grant constraints are more restrictive.

```
# The child grant in a ListGrants response.
{
    "Grants": [
        {
            "KeyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "CreationDate": 1572249600.0,
            "GrantId":
 "fedcba9999c1e2e9876abcde6e9d6c9b6a1987650000abcee009abcdef40183f",
            "Operations": [
                "CreateGrant"
                "Decrypt"
            ]
            "RetiringPrincipal": "arn:aws:iam::111122223333:user/exampleUser",
            "Name": "",
            "IssuingAccount": "arn:aws:iam::111122223333:root",
            "GranteePrincipal": "arn:aws:iam::111122223333:user/anotherUser",
            "Constraints": {
IAM best practices discourage the use of IAM users with long-term credentials. Whenever
                      possible, use IAM roles, which provide temporary credentials. For
 details,
                      see Security best practices in IAM in the IAM User Guide.
        "EncryptionContextEquals": {
                    "Department": "IT"
                }
            },
        }
    ]
}
```

The grantee principal in the child grant, `anotherUser`, can use their `CreateGrant` permission to create grants. However, the grants that `anotherUser` creates must include the operations in its parent grant or a subset, and the grant constraints must be the same or stricter.

Managing grants

Principals with the required permissions can view, use and delete (retire or revoke) grants. To refine permissions for creating and managing grants, AWS KMS supports several policy conditions that you can use in key policies and IAM policies.

Topics

- Controlling access to grants

- Viewing grants
- Using a grant token
- Retiring and revoking grants

Controlling access to grants

You can control access to the operations that create and manage grants in key policies, IAM policies, and in grants. Principals who get `CreateGrant` permission from a grant have more limited grant permissions.

API operation	Key policy or IAM policy	Grant
CreateGrant	✓	✓
ListGrants	✓	-
ListRetirableGrants	✓	-
Retire Grants	(Limited. See Retiring and revoking grants)	✓
RevokeGrant	✓	-

When you use a key policy or IAM policy to control access to operations that create and manage grants, you can use one or more of the following policy conditions to limit the permission. AWS KMS supports all of the following grant-related condition keys. For detailed information and examples, see AWS KMS condition keys.

kms:GrantConstraintType

Allows principals to create a grant only when the grant includes the specified grant constraint.

kms:GrantIsForAWSResource

Allows principals to call `CreateGrant`, `ListGrants`, or `RevokeGrant` only when an AWS service that is integrated with AWS KMS sends the request on the principal's behalf.

kms:GrantOperations

Allows principals to create a grant, but limits the grant to the specified operations.

kms:GranteePrincipal

Allows principals to create a grant only for the specified grantee principal.

kms:RetiringPrincipal

Allows principals to create a grant only when the grant specifies a particular retiring principal.

Viewing grants

To view the grant, use the ListGrants operation. You must specify the KMS key to which the grants apply. You can also filter the grant list by grant ID or grantee principal. For more examples, see Viewing a grant.

To view all grants in the AWS account and Region with a particular retiring principal, use ListRetirableGrants. The responses include details about each grant.

Note

The `GranteePrincipal` field in the `ListGrants` response usually contains the grantee principal of the grant. However, when the grantee principal in the grant is an AWS service, the `GranteePrincipal` field contains the service principal, which might represent several different grantee principals.

For example, the following command lists all of the grants for a KMS key.

```
$  aws kms list-grants --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
{
    "Grants": [
        {
            "KeyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "CreationDate": 1572216195.0,
            "GrantId":
 "abcde1237f76e4ba7987489ac329fbfba6ad343d6f7075dbd1ef191f0120514a",
            "Constraints": {
                "EncryptionContextSubset": {
                    "Department": "IT"
                }
            },
            "RetiringPrincipal": "arn:aws:iam::111122223333:role/adminRole",
            "Name": "",
            "IssuingAccount": "arn:aws:iam::111122223333:root",
            "GranteePrincipal": "arn:aws:iam::111122223333:user/exampleUser",
            "Operations": [
                "Decrypt"
            ]
        }
    ]
}
```

Using a grant token

The AWS KMS API follows an eventual consistency model. When you create a grant, the grant might not be effective immediately. There might be a brief delay before the change is available throughout AWS KMS. It typically takes less than a few seconds for the change to propagate throughout the system, but in some cases it can take several minutes. Once the change has fully propagated throughout the system, the grantee principal can use the permissions in the grant without specifying the grant token or any evidence of the grant. However, if a grant that is so new that it is not yet known to all of AWS KMS, the request might fail with an `AccessDeniedException` error.

To use the permissions in a new grant immediately, use the grant token for the grant. Save the grant token that the CreateGrant operation returns. Then submit the grant token in the request for the AWS KMS operation. You can submit a grant token to any AWS KMS grant operation and you can submit multiple grant tokens in the same request.

The following example uses the `CreateGrant` operation to create a grant that allows the GenerateDataKey and Decrypt operations. It saves the grant token that `CreateGrant` returns in the `token` variable. Then, in a call to the `GenerateDataKey` operation, it uses the grant token in the `token` variable.

```
# Create a grant; save the grant token
$ token=$(aws kms create-grant \
    --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
    --grantee-principal arn:aws:iam::111122223333:user/appUser \
    --retiring-principal arn:aws:iam::111122223333:user/acctAdmin \
    --operations GenerateDataKey Decrypt \
    --query GrantToken \
    --output text)
```

```
# Use the grant token in a request
$ aws kms generate-data-key \
    --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
    --key-spec AES_256 \
    --grant-tokens $token
```

Principals with permission can also use a grant token to retire a new grant even before the grant is available throught AWS KMS. (The RevokeGrant operation doesn't accept a grant token.) For details, see Retiring and revoking grants.

```
# Retire the grant
$ aws kms retire-grant --grant-token $token
```

Retiring and revoking grants

To delete a grant, retire or revoke it.

The RetireGrant and RevokeGrant operations are very similar to each other. Both operations delete a grant, which eliminates the permissions the grant allows. The primary difference between these operations is how they are authorized.

RevokeGrant

Like most AWS KMS operations, access to the RevokeGrant operation is controlled through key policies and IAM policies. The RevokeGrant API can be called by any principal with kms:RevokeGrant permission. This permission is included in the standard permissions given to key administrators. Typically, administrators revoke a grant to deny permissions the grant allows.

RetireGrant

The grant determines who can retire it. This design allows you to control the lifecycle of a grant without changing key policies or IAM policies. Typically, you retire a grant when you are done using its permissions.

A grant can be retired by an optional retiring principal specified in the grant. The grantee principal can also retire the grant, but only if they are also a retiring principal or the grant includes the RetireGrant operation. As a backup, the AWS account in which the grant was created can retire the grant.

There is a `kms:RetireGrant` permission that can be used in IAM policies, but it has limited utility. Principals specified in the grant can retire a grant without the `kms:RetireGrant` permission. The `kms:RetireGrant` permission alone does not allow principals to retire a grant. The `kms:RetireGrant` permission is not effective in a key policy.

- To deny permission to retire a grant, you can use a `Deny` action with the `kms:RetireGrant` permission.
- The AWS account that owns the KMS key can delegate the `kms:RetireGrant` permission to IAM principal in the account.
- If the retiring principal is a different AWS account, administrators in the other account can use `kms:RetireGrant` to delegate permission to retire the grant to an IAM principal in that account.

The AWS KMS API follows an eventual consistency model. When you create, retire, or revoke a grant, there might be a brief delay before the change is available throughout AWS KMS. It typically takes less than a few seconds for the change to propagate throughout the system, but in some cases it can take several minutes. If you need to delete a new grant immediately, before it is available throughout AWS KMS, use a grant token to retire the grant. You cannot use a grant token to revoke a grant.

Connecting to AWS KMS through a VPC endpoint

You can connect directly to AWS KMS through a private interface endpoint in your virtual private cloud (VPC). When you use an interface VPC endpoint, communication between your VPC and AWS KMS is conducted entirely within the AWS network.

AWS KMS supports Amazon Virtual Private Cloud (Amazon VPC) endpoints powered by AWS PrivateLink. Each VPC endpoint is represented by one or more Elastic Network Interfaces (ENIs) with private IP addresses in your VPC subnets.

The interface VPC endpoint connects your VPC directly to AWS KMS without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. The instances in your VPC do not need public IP addresses to communicate with AWS KMS.

Regions

AWS KMS supports VPC endpoints and VPC endpoint policies in all AWS Regions in which AWS KMS is supported.

Topics

Considerations for AWS KMS VPC endpoints

Before you set up an interface VPC endpoint for AWS KMS, review the Interface endpoint properties and limitations topic in the *AWS PrivateLink Guide*.

AWS KMS support for a VPC endpoint includes the following.

- You can use your VPC endpoint to call all AWS KMS API operations from your VPC.
- You can create an interface VPC endpoint that connects to an AWS KMS region endpoint or an AWS KMS FIPS endpoint.
- You can use AWS CloudTrail logs to audit your use of KMS keys through the VPC endpoint. For details, see Logging your VPC endpoint.

Creating a VPC endpoint for AWS KMS

You can create a VPC endpoint for AWS KMS by using the Amazon VPC console or the Amazon VPC API. For more information, see Create an interface endpoint in the *AWS PrivateLink Guide*.

- To create a VPC endpoint for AWS KMS, use the following service name:

  ```
  com.amazonaws.region.kms
  ```

  For example, in the US West (Oregon) Region (`us-west-2`), the service name would be:

  ```
  com.amazonaws.us-west-2.kms
  ```

- To create a VPC endpoint that connects to an AWS KMS FIPS endpoint, use the following service name:

```
com.amazonaws.region.kms-fips
```

For example, in the US West (Oregon) Region (`us-west-2`), the service name would be:

```
com.amazonaws.us-west-2.kms-fips
```

To make it easier to use the VPC endpoint, you can enable a private DNS name for your VPC endpoint. If you select the **Enable DNS Name** option, the standard AWS KMS DNS hostname resolves to your VPC endpoint. For example, `https://kms.us-west-2.amazonaws.com` would resolve to a VPC endpoint connected to service name `com.amazonaws.us-west-2.kms`.

This option makes it easier to use the VPC endpoint. The AWS SDKs and AWS CLI use the standard AWS KMS DNS hostname by default, so you do not need to specify the VPC endpoint URL in applications and commands.

For more information, see Accessing a service through an interface endpoint in the *AWS PrivateLink Guide*.

Connecting to an AWS KMS VPC endpoint

You can connect to AWS KMS through the VPC endpoint by using an AWS SDK, the AWS CLI or AWS Tools for PowerShell. To specify the VPC endpoint, use its DNS name.

For example, this list-keys command uses the `endpoint-url` parameter to specify the VPC endpoint. To use a command like this, replace the example VPC endpoint ID with one in your account.

```
$ aws kms list-keys --endpoint-url https://vpce-1234abcdf5678c90a-09p7654s-us-
east-1a.ec2.us-east-1.vpce.amazonaws.com
```

If you enabled private hostnames when you created your VPC endpoint, you do not need to specify the VPC endpoint URL in your CLI commands or application configuration. The standard AWS KMS DNS hostname resolves to your VPC endpoint. The AWS CLI and SDKs use this hostname by default, so you can begin using the VPC endpoint to connect to an AWS KMS regional endpoint without changing anything in your scripts and applications.

To use private hostnames, the `enableDnsHostnames` and `enableDnsSupport` attributes of your VPC must be set to `true`. To set these attributes, use the ModifyVpcAttribute operation. For details, see View and update DNS attributes for your VPC in the *Amazon VPC User Guide*.

Controlling access to a VPC endpoint

To control access to your VPC endpoint for AWS KMS, attach a *VPC endpoint policy* to your VPC endpoint. The endpoint policy determines whether principals can use the VPC endpoint to call AWS KMS operations on AWS KMS resources.

You can create a VPC endpoint policy when you create your endpoint, and you can change the VPC endpoint policy at any time. Use the VPC management console, or the CreateVpcEndpoint or ModifyVpcEndpoint operations. You can also create and change a VPC endpoint policy by using an AWS CloudFormation template. For help using the VPC management console, see Create an interface endpoint and Modifying an interface endpoint in the *AWS PrivateLink Guide*.

> **Note**
>
> AWS KMS supports VPC endpoint policies beginning in July 2020. VPC endpoints for AWS KMS that were created before that date have the default VPC endpoint policy, but you can change it at any time.

For help writing and formatting a JSON policy document, see the IAM JSON Policy Reference in the *IAM User Guide*.

Topics

About VPC endpoint policies

For an AWS KMS request that uses a VPC endpoint to be successful, the principal requires permissions from two sources:

- A key policy, IAM policy, or grant must give principal permission to call the operation on the resource (KMS key or alias).
- A VPC endpoint policy must give the principal permission to use the endpoint to make the request.

For example, a key policy might give a principal permission to call Decrypt on a particular KMS key. However, the VPC endpoint policy might not allow that principal to call `Decrypt` on that KMS key by using the endpoint.

Or a VPC endpoint policy might allow a principal to use the endpoint to call DisableKey on certain KMS keys. But if the principal doesn't have those permissions from a key policy, IAM policy, or grant, the request fails.

Default VPC endpoint policy

Every VPC endpoint has a VPC endpoint policy, but you are not required to specify the policy. If you don't specify a policy, the default endpoint policy allows all operations by all principals on all resources over the endpoint.

However, for AWS KMS resources, the principal must also have permission to call the operation from a key policy, IAM policy, or grant. Therefore, in practice, the default policy says that if a principal has permission to call an operation on a resource, they can also call it by using the endpoint.

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Principal": "*",
      "Resource": "*"
    }
  ]
}
```

To allow principals to use the VPC endpoint for only a subset of their permitted operations, create or update the VPC endpoint policy.

Creating a VPC endpoint policy

A VPC endpoint policy determines whether a principal has permission to use the VPC endpoint to perform operations on a resource. For AWS KMS resources, the principal must also have permission to perform the operations from a key policy, IAM policy, or grant.

Each VPC endpoint policy statement requires the following elements:

- The principal that can perform actions
- The actions that can be performed
- The resources on which actions can be performed

The policy statement doesn't specify the VPC endpoint. Instead, it applies to any VPC endpoint to which the policy is attached. For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

The following is an example of a VPC endpoint policy for AWS KMS. When attached to a VPC endpoint, this policy allows `ExampleUser` to use the VPC endpoint to call the specified operations on the specified KMS keys. Before using a policy like this one, replace the example principal and key ARN with valid values from your account.

```
{
   "Statement":[
      {
         "Sid": "AllowDecryptAndView",
         "Principal": {"AWS": "arn:aws:iam::111122223333:user/ExampleUser"},
         "Effect":"Allow",
         "Action": [
             "kms:Decrypt",
             "kms:DescribeKey",
             "kms:ListAliases",
             "kms:ListKeys"
          ],
         "Resource": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
      }
   ]
}
```

AWS CloudTrail logs all operations that use the VPC endpoint. However, your CloudTrail logs don't include operations requested by principals in other accounts or operations for KMS keys in other accounts.

As such, you might want to create a VPC endpoint policy that prevents principals in external accounts from using the VPC endpoint to call any AWS KMS operations on any keys in the local account.

The following example uses the aws:PrincipalAccount global condition key to deny access to all principals for all operations on all KMS keys unless the principal is in the local account. Before using a policy like this one, replace the example account ID with a valid one.

```
{
  "Statement": [
    {
      "Sid": "AccessForASpecificAccount",
      "Principal": {"AWS": "*"},
      "Action": "kms:*",
      "Effect": "Deny",
      "Resource": "arn:aws:kms:*:111122223333:key/*",
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalAccount": "111122223333"
        }
      }
    }
  ]
}
```

Viewing a VPC endpoint policy

To view the VPC endpoint policy for an endpoint, use the VPC management console or the DescribeVpcEndpoints operation.

The following AWS CLI command gets the policy for the endpoint with the specified VPC endpoint ID.

Before using this command, replace the example endpoint ID with a valid one from your account.

```
$ aws ec2 describe-vpc-endpoints \
--query 'VpcEndpoints[?VpcEndpointId==`vpce-1234abcdf5678c90a`].[PolicyDocument]'
--output text
```

Using a VPC endpoint in a policy statement

You can control access to AWS KMS resources and operations when the request comes from VPC or uses a VPC endpoint. To do so, use one of the following global condition keys in a key policy or IAM policy.

- Use the `aws:sourceVpce` condition key to grant or restrict access based on the VPC endpoint.
- Use the `aws:sourceVpc` condition key to grant or restrict access based on the VPC that hosts the private endpoint.

> **Note**
>
> Use caution when creating key policies and IAM policies based on your VPC endpoint. If a policy statement requires that requests come from a particular VPC or VPC endpoint, requests from integrated AWS services that use an AWS KMS resource on your behalf might fail. For help, see Using VPC endpoint conditions in policies with AWS KMS permissions. Also, the `aws:sourceIP` condition key is not effective when the request comes from an Amazon VPC endpoint. To restrict requests to a VPC endpoint, use the `aws:sourceVpce` or `aws:sourceVpc` condition keys. For more information, see Identity and access management for VPC endpoints and VPC endpoint services in the *AWS PrivateLink Guide*.

You can use these global condition keys to control access to AWS KMS keys (KMS keys), aliases, and to operations like CreateKey that don't depend on any particular resource.

For example, the following sample key policy allows a user to perform some cryptographic operations with a KMS key only when the request uses the specified VPC endpoint. When a user makes a request to AWS KMS, the VPC endpoint ID in the request is compared to the `aws:sourceVpce` condition key value in the policy. If they do not match, the request is denied.

To use a policy like this one, replace the placeholder AWS account ID and VPC endpoint IDs with valid values for your account.

```
{
    "Id": "example-key-1",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable IAM policies",
            "Effect": "Allow",
            "Principal": {"AWS":["111122223333"]},
            "Action": ["kms:*"],
            "Resource": "*"
        },
        {
            "Sid": "Restrict usage to my VPC endpoint",
            "Effect": "Deny",
            "Principal": "*",
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*"
            ],
            "Resource": "*",
            "Condition": {
                "StringNotEquals": {
                    "aws:sourceVpce": "vpce-1234abcdf5678c90a"
                }
            }
        }

    ]
}
```

You can also use the `aws:sourceVpc` condition key to restrict access to your KMS keys based on the VPC in which VPC endpoint resides.

The following sample key policy allows commands that manage the KMS key only when they come from `vpc-12345678`. In addition, it allows commands that use the KMS key for cryptographic operations only when they come from `vpc-2b2b2b2b`. You might use a policy like this one if an application is running in one VPC, but you use a second, isolated VPC for management functions.

To use a policy like this one, replace the placeholder AWS account ID and VPC endpoint IDs with valid values for your account.

```
{
    "Id": "example-key-2",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Allow administrative actions from vpc-12345678",
            "Effect": "Allow",
            "Principal": {"AWS": "111122223333"},
            "Action": [
                "kms:Create*","kms:Enable*","kms:Put*","kms:Update*",
                "kms:Revoke*","kms:Disable*","kms:Delete*",
                "kms:TagResource", "kms:UntagResource"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:sourceVpc": "vpc-12345678"
                }
            }
        },
        {
            "Sid": "Allow key usage from vpc-2b2b2b2b",
            "Effect": "Allow",
            "Principal": {"AWS": "111122223333"},
            "Action": [
                "kms:Encrypt","kms:Decrypt","kms:GenerateDataKey*"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:sourceVpc": "vpc-2b2b2b2b"
                }
            }
        },
        {
            "Sid": "Allow read actions from everywhere",
            "Effect": "Allow",
            "Principal": {"AWS": "111122223333"},
            "Action": [
                "kms:Describe*","kms:List*","kms:Get*"
            ],
            "Resource": "*",
        }
    ]
}
```

Logging your VPC endpoint

AWS CloudTrail logs all operations that use the VPC endpoint. When a request to AWS KMS uses a VPC endpoint, the VPC endpoint ID appears in the AWS CloudTrail log entry that records the request. You can use the endpoint ID to audit the use of your AWS KMS VPC endpoint.

However, your CloudTrail logs don't include operations requested by principals in other accounts or requests for AWS KMS operations on KMS keys and aliases in other accounts. Also, to protect your VPC, requests that are denied by a VPC endpoint policy, but otherwise would have been allowed, are not recorded in AWS CloudTrail.

For example, this sample log entry records a GenerateDataKey request that used the VPC endpoint. The `vpcEndpointId` field appears at the end of the log entry.

```
{
  "eventVersion":"1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::111122223333:user/Alice",
    "accessKeyId": "EXAMPLE_KEY_ID",
    "accountId": "111122223333",
    "userName": "Alice"
  },
  "eventTime":"2018-01-16T05:46:57Z",
  "eventSource":"kms.amazonaws.com",
  "eventName":"GenerateDataKey",
  "awsRegion":"eu-west-1",
  "sourceIPAddress":"172.01.01.001",
  "userAgent":"aws-cli/1.14.23 Python/2.7.12 Linux/4.9.75-25.55.amzn1.x86_64
 botocore/1.8.27",
  "requestParameters":{
    "keyId":"1234abcd-12ab-34cd-56ef-1234567890ab",
    "numberOfBytes":128
  },
  "responseElements":null,
  "requestID":"a9fff0bf-fa80-11e7-a13c-afcabff2f04c",
  "eventID":"77274901-88bc-4e3f-9bb6-acf1c16f6a7c",
  "readOnly":true,
  "resources":[{
    "ARN":"arn:aws:kms:eu-
west-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "accountId":"111122223333",
    "type":"AWS::KMS::Key"
  }],
  "eventType":"AwsApiCall",
  "recipientAccountId":"111122223333",
  "vpcEndpointId": "vpce-1234abcdf5678c90a"
}
```

Condition keys for AWS KMS

You can specify conditions in the key policies and IAM policies that control access to AWS KMS resources. The policy statement is effective only when the conditions are true. For example, you might want a policy statement to take effect only after a specific date. Or, you might want a policy statement to control access only when a specific value appears in an API request.

To specify conditions, you use *condition keys* in the `Condition` element of a policy statement with IAM condition operators. Some condition keys apply generally to AWS; others are specific to AWS KMS.

Condition key values must adhere to the character and encoding rules for AWS KMS key policies and IAM policies. For details about key policy document rules, see Key policy format. For details about IAM policy document rules, see IAM name requirements in the *IAM User Guide*..

Topics

AWS global condition keys

AWS defines global condition keys, a set of policy conditions keys for all AWS services that use IAM for access control. AWS KMS supports all global condition keys. You can use them in AWS KMS key policies and IAM policies.

For example, you can use the aws:PrincipalArn global condition key to allow access to an AWS KMS key (KMS key) only when the principal in the request is represented by the Amazon Resource Name (ARN) in the condition key value. To support attribute-based access control (ABAC) in AWS KMS, you can use the aws:ResourceTag/*tag-key* global condition key in an IAM policy to allow access to KMS keys with a particular tag.

To help prevent an AWS service from being used as a confused deputy in a policy where the principal is an AWS service principal, you can use the aws:SourceArn or aws:SourceAccount global condition keys. For details, see Using `aws:SourceArn` or `aws:SourceAccount` condition keys.

For information about AWS global condition keys, including the types of requests in which they are available, see AWS Global Condition Context Keys in the *IAM User Guide*. For examples of using global condition keys in IAM policies, see Controlling Access to Requests and Controlling Tag Keys in the *IAM User Guide*.

The following topics provide special guidance for using condition keys based on IP addresses and VPC endpoints.

Topics

- Using the IP address condition in policies with AWS KMS permissions
- Using VPC endpoint conditions in policies with AWS KMS permissions

Using the IP address condition in policies with AWS KMS permissions

You can use AWS KMS to protect your data in an integrated AWS service. But use caution when specifying the IP address condition operators or the `aws:SourceIp` condition key in the same policy statement that allows or denies access to AWS KMS. For example, the policy in AWS: Denies Access to AWS Based on the Source IP restricts AWS actions to requests from the specified IP range.

Consider this scenario:

1. You attach a policy like the one shown at AWS: Denies Access to AWS Based on the Source IP to an IAM identity. You set the value of the `aws:SourceIp` condition key to the range of IP addresses for the user's company. This IAM identity has other policies attached that allow it to use Amazon EBS, Amazon EC2, and AWS KMS.
2. The identity attempts to attach an encrypted EBS volume to an EC2 instance. This action fails with an authorization error even though the user has permission to use all the relevant services.

Step 2 fails because the request to AWS KMS to decrypt the volume's encrypted data key comes from an IP address that is associated with the Amazon EC2 infrastructure. To succeed, the request must come from the IP address of the originating user. Because the policy in step 1 explicitly denies all requests from IP addresses other than those specified, Amazon EC2 is denied permission to decrypt the EBS volume's encrypted data key.

Also, the `aws:sourceIP` condition key is not effective when the request comes from an Amazon VPC endpoint. To restrict requests to a VPC endpoint, including an AWS KMS VPC endpoint, use the `aws:sourceVpce` or `aws:sourceVpc` condition keys. For more information, see VPC Endpoints - Controlling the Use of Endpoints in the *Amazon VPC User Guide*.

Using VPC endpoint conditions in policies with AWS KMS permissions

AWS KMS supports Amazon Virtual Private Cloud (Amazon VPC) endpoints that are powered by AWS PrivateLink. You can use the following global condition keys in key policies and IAM policies to control access to AWS KMS resources when the request comes from a VPC or uses a VPC endpoint. For details, see Using a VPC endpoint in a policy statement.

- `aws:SourceVpc` limits access to requests from the specified VPC.
- `aws:SourceVpce` limits access to requests from the specified VPC endpoint.

If you use these condition keys to control access to KMS keys, you might inadvertently deny access to AWS services that use AWS KMS on your behalf.

Take care to avoid a situation like the IP address condition keys example. If you restrict requests for a KMS key to a VPC or VPC endpoint, calls to AWS KMS from an integrated service, such as Amazon S3 or Amazon EBS, might fail. This can happen even if the source request ultimately originates in the VPC or from the VPC endpoint.

AWS KMS condition keys

AWS KMS provides a set of condition keys that you can use in key policies and IAM policies. These condition keys are specific to AWS KMS. For example, you can use the `kms:EncryptionContext:`*`context-key`* condition key to require a particular encryption context when controlling access to a symmetric encryption KMS key.

Conditions for an API operation request

Many AWS KMS condition keys control access to a KMS key based on the value of a parameter in the request for an AWS KMS operation. For example, you can use the kms:KeySpec condition key in an IAM policy to allow use of the CreateKey operation only when the value of the `KeySpec` parameter in the `CreateKey` request is `RSA_4096`.

This type of condition works even when the parameter doesn't appear in the request, such as when you use the parameter's default value. For example you can use the kms:KeySpec condition key to allow users to use the `CreateKey` operation only when the value of the `KeySpec` parameter is `SYMMETRIC_DEFAULT`, which is the default value. This condition allows requests that have the `KeySpec` parameter with the `SYMMETRIC_DEFAULT` value and requests that have no `KeySpec` parameter.

Conditions for KMS keys used in API operations

Some AWS KMS condition keys can control access to operations based on a property of the KMS key that is used in the operation. For example, you can use the kms:KeyOrigin condition to allow principals to call GenerateDataKey on a KMS key only when the `Origin` of the KMS key is `AWS_KMS`. To find out if a condition key can be used in this way, see the description of the condition key.

The operation must be a *KMS key resource operation*, that is, an operation that is authorized for a particular KMS key. To identify the KMS key resource operations, in the Actions and Resources Table, look for a value of `KMS key` in the `Resources` column for the operation. If you use this type of condition key with an operation that is not authorized for a particular KMS key resource, like ListKeys, the permission is not effective because the condition can never be satisfied. There is no KMS key resource involved in authorizing the `ListKeys` operation and no `KeySpec` property.

The following topics describe each AWS KMS condition key and include example policy statements that demonstrate policy syntax.

Using set operators with condition keys

When a policy condition compares two set of values, such as the set of tags in a request and the set of tags in a policy, you need tell AWS how to compare the sets. IAM defines two set operators, `ForAnyValue` and `ForAllValues`, for this purpose. Use set operators only with *multi-valued condition keys*, which require them. Do not use set operators with *single-valued condition keys*. As always, test your policy statements thoroughly before using them in a production environment.

Condition keys are single-valued or multi-valued. To determine whether an AWS KMS condition key is single-valued or multi-valued, see the **Value type** column in the condition key description.

- *Single-valued* condition keys have at most one value in the authorization context (the request or resource). For example, because each API call can originate from only one AWS account, kms:CallerAccount is a single valued condition key. Do not use a set operator with a single-valued condition key.

- *Multi-valued* condition keys have multiple values in the authorization context (the request or resource). For example, because each KMS key can have multiple aliases, kms:ResourceAliases can have multiple values. Multi-valued condition keys require a set operator.

Note that the difference between single-valued and multi-valued condition keys depends on the number of values in the authorization context; not the number of values in the policy condition.

⚠ **Warning**

Using a set operator with a single-valued condition key can create a policy statement that is overly permissive (or overly restrictive). Use set operators only with multi-valued condition keys.

If you create or update a policy that includes a `ForAllValues` set operator with the kms:EncryptionContext:*context-key* or `aws:RequestTag/`*`tag-key`* condition keys, AWS KMS returns the following error message:

`OverlyPermissiveCondition: Using the ForAllValues set operator with a single-valued condition key matches requests without the specified [encryption context or tag] or with an unspecified [encryption context or tag]. To fix, remove ForAllValues.`

For detailed information about the `ForAnyValue` and `ForAllValues` set operators, see Using multiple keys and values in the *IAM User Guide*. For information about the risk of using the `ForAllValues` set operator with a single-valued condition, see Security Warning – ForAllValues with single valued key in the *IAM User Guide*.

Topics

- kms:ExpirationModel
- kms:GrantConstraintType
- kms:GrantIsForAWSResource
- kms:GrantOperations
- kms:GranteePrincipal
- kms:KeyAgreementAlgorithm
- kms:KeyOrigin
- kms:KeySpec
- kms:KeyUsage
- kms:MacAlgorithm
- kms:MessageType
- kms:MultiRegion
- kms:MultiRegionKeyType
- kms:PrimaryRegion
- kms:ReEncryptOnSameKey
- kms:RequestAlias
- kms:ResourceAliases
- kms:ReplicaRegion
- kms:RetiringPrincipal
- kms:RotationPeriodInDays
- kms:ScheduleKeyDeletionPendingWindowInDays
- kms:SigningAlgorithm
- kms:ValidTo
- kms:ViaService
- kms:WrappingAlgorithm
- kms:WrappingKeySpec

kms:BypassPolicyLockoutSafetyCheck

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
`kms:BypassPolicyLockoutSafetyCheck`	Boolean	Single-valued	`CreateKey` `PutKeyPolicy`	IAM policies only Key policies and IAM policies

The `kms:BypassPolicyLockoutSafetyCheck` condition key controls access to the CreateKey and PutKeyPolicy operations based on the value of the `BypassPolicyLockoutSafetyCheck` parameter in the request.

The following example IAM policy statement prevents users from bypassing the policy lockout safety check by denying them permission to create KMS keys when the value of the `BypassPolicyLockoutSafetyCheck` parameter in the `CreateKey` request is `true.`

```
{
  "Effect": "Deny",
  "Action": [
    "kms:CreateKey",
    "kms:PutKeyPolicy"
  ],
  "Resource": "*",
  "Condition": {
    "Bool": {
      "kms:BypassPolicyLockoutSafetyCheck": true
    }
  }
}
```

You can also use the `kms:BypassPolicyLockoutSafetyCheck` condition key in an IAM policy or key policy to control access to the `PutKeyPolicy` operation. The following example policy statement from a key policy prevents users from bypassing the policy lockout safety check when changing the policy of a KMS key.

Instead of using an explicit `Deny`, this policy statement uses `Allow` with the Null condition operator to allow access only when the request does not include the

`BypassPolicyLockoutSafetyCheck` parameter. When the parameter is not used, the default value is `false`. This slightly weaker policy statement can be overridden in the rare case that a bypass is necessary.

```
{
  "Effect": "Allow",
  "Action": "kms:PutKeyPolicy",
  "Resource": "*",
  "Condition": {
    "Null": {
      "kms:BypassPolicyLockoutSafetyCheck": true
    }
  }
}
```

See also

- kms:KeySpec
- kms:KeyOrigin
- kms:KeyUsage

kms:CallerAccount

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
`kms:CallerAccount`	String	Single-valued	KMS key resource operations Custom key store operations	Key policies and IAM policies

You can use this condition key to allow or deny access to all identities (users and roles) in an AWS account. In key policies, you use the `Principal` element to specify the identities to which the policy statement applies. The syntax for the `Principal` element does not provide a way to specify all identities in an AWS account. But you can achieve this effect by combining this condition key with a `Principal` element that specifies all AWS identities.

You can use it to control access to any *KMS key resource operation*, that is, any AWS KMS operation that uses a particular KMS key. To identify the KMS key resource operations, in the Actions and Resources Table, look for a value of `KMS key` in the `Resources` column for the operation. It is also valid for operations that manage custom key stores.

For example, the following key policy statement demonstrates how to use the `kms:CallerAccount` condition key. This policy statement is in the key policy for the AWS managed key for Amazon EBS. It combines a `Principal` element that specifies all AWS identities with the `kms:CallerAccount` condition key to effectively allow access to all identities in AWS account 111122223333. It contains an additional AWS KMS condition key (`kms:ViaService`) to further limit the permissions by only allowing requests that come through Amazon EBS. For more information, see kms:ViaService.

```
{
  "Sid": "Allow access through EBS for all principals in the account that are
 authorized to use EBS",
  "Effect": "Allow",
  "Principal": {"AWS": "*"},
  "Condition": {
    "StringEquals": {
      "kms:CallerAccount": "111122223333",
      "kms:ViaService": "ec2.us-west-2.amazonaws.com"
    }
  },
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:CreateGrant",
    "kms:DescribeKey"
  ],
  "Resource": "*"
}
```

kms:CustomerMasterKeySpec (deprecated)

The `kms:CustomerMasterKeySpec` condition key is deprecated. Instead, use the kms:KeySpec condition key.

The `kms:CustomerMasterKeySpec` and `kms:KeySpec` condition keys work the same way. Only the names differ. We recommend that you use `kms:KeySpec`. However, to avoid breaking changes, AWS KMS supports both condition keys.

kms:CustomerMasterKeyUsage (deprecated)

The `kms:CustomerMasterKeyUsage` condition key is deprecated. Instead, use the kms:KeyUsage condition key.

The `kms:CustomerMasterKeyUsage` and `kms:KeyUsage` condition keys work the same way. Only the names differ. We recommend that you use `kms:KeyUsage`. However, to avoid breaking changes, AWS KMS supports both condition keys.

kms:DataKeyPairSpec

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
`kms:DataKeyPairSpec`	String	Single-valued	`GenerateDataKeyPair` `GenerateDataKeyPairWithoutPlaintext`	Key policies and IAM policies

You can use this condition key to control access to the GenerateDataKeyPair and GenerateDataKeyPairWithoutPlaintext operations based on the value of the `KeyPairSpec` parameter in the request. For example, you can allow users to generate only particular types of data key pairs.

The following example key policy statement uses the `kms:DataKeyPairSpec` condition key to allow users to use the KMS key to generate only RSA data key pairs.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
  },
  "Action": [
    "kms:GenerateDataKeyPair",
    "kms:GenerateDataKeyPairWithoutPlaintext"
  ],
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "kms:DataKeyPairSpec": "RSA*"
    }
  }
}
```

See also

- kms:KeySpec
- the section called “kms:EncryptionAlgorithm”
- the section called “kms:EncryptionContext:*context-key*”
- the section called “kms:EncryptionContextKeys”

kms:EncryptionAlgorithm

AWS KMS condition keys	Condition type	Value type	API operations	Policy type

`kms:EncryptionAlgorithm`	String	Single-valued	`Decrypt`
			`Encrypt`
			`GenerateDataKey`
			`GenerateDataKeyPair`
				Key policies and IAM policies

AWS KMS condition keys	Condition type	Value type	API operations	Policy type

			`GenerateDataKeyPairWithoutPlaintext`
			`GenerateDataKeyWithoutPlaintext`
			`ReEncrypt`	

You can use the `kms:EncryptionAlgorithm` condition key to control access to cryptographic operations based on the encryption algorithm that is used in the operation. For the Encrypt, Decrypt, and ReEncrypt operations, it controls access based on the value of the EncryptionAlgorithm parameter in the request. For operations that generate data keys and data key pairs, it controls access based on the encryption algorithm that is used to encrypt the data key.

This condition key has no effect on operations performed outside of AWS KMS, such as encrypting with the public key in an asymmetric KMS key pair outside of AWS KMS.

EncryptionAlgorithm parameter in a request

To allow users to use only a particular encryption algorithm with a KMS key, use a policy statement with a `Deny` effect and a `StringNotEquals` condition operator. For example, the following example key policy statement prohibits principals who can assume the `ExampleRole` role from using this KMS key in the specified cryptographic operations unless the encryption algorithm in the request is `RSAES_OAEP_SHA_256`, an asymmetric encryption algorithm used with RSA KMS keys.

Unlike a policy statement that allows a user to use a particular encryption algorithm, a policy statement with a double-negative like this one prevents other policies and grants for this KMS key from allowing this role to use other encryption algorithms. The `Deny` in this key policy statement takes precedence over any key policy or IAM policy with an `Allow` effect, and it takes precedence over all grants for this KMS key and its principals.

```
{
  "Sid": "Allow only one encryption algorithm with this asymmetric KMS key",
  "Effect": "Deny",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
  },
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*"
  ],
  "Resource": "*",
  "Condition": {
    "StringNotEquals": {
      "kms:EncryptionAlgorithm": "RSAES_OAEP_SHA_256"
    }
  }
}
```

Encryption algorithm used for the operation

You can also use the `kms:EncryptionAlgorithm` condition key to control access to operations based on the encryption algorithm used in the operation, even when the algorithm isn't specified in the request. This allows you to require or forbid the `SYMMETRIC_DEFAULT` algorithm, which might not be specified in a request because it's the default value.

This feature lets you use the `kms:EncryptionAlgorithm` condition key to control access to the operations that generate data keys and data key pairs. These operations use only symmetric encryption KMS keys and the `SYMMETRIC_DEFAULT` algorithm.

For example, this IAM policy limits its principals to symmetric encryption. It denies access to any KMS key in the example account for cryptographic operations unless the encryption algorithm specified in the request or used in the operation is SYMMETRIC_DEFAULT. Including `GenerateDataKey*` adds GenerateDataKey, GenerateDataKeyWithoutPlaintext, GenerateDataKeyPair, and GenerateDataKeyPairWithoutPlaintext to the permissions. The condition has no effect on these operations because they always use a symmetric encryption algorithm.

```
{
  "Sid": "AllowOnlySymmetricAlgorithm",
  "Effect": "Deny",
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*"
  ],
  "Resource": "arn:aws:kms:us-west-2:111122223333:key/*",
  "Condition": {
    "StringNotEquals": {
      "kms:EncryptionAlgorithm": "SYMMETRIC_DEFAULT"
    }
  }
}
```

See also

- the section called “kms:MacAlgorithm”
- kms:SigningAlgorithm

kms:EncryptionContext:*context-key*

AWS KMS condition keys	Condition type	Value type	API operations	Policy type

`kms:EncryptionContext:` *`context-key`*	String	Single-valued	`CreateGrant`
			`Encrypt`
			`Decrypt`
			`GenerateDataKey`
			`GenerateDataKeyPair`
			`GenerateDataKeyPairWithoutPlaintext`
			`GenerateDataKeyWit`
				Key policies and IAM policies

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
			houtPlain text ReEncrypt RetireGrant	

You can use the `kms:EncryptionContext:context-key` condition key to control access to a symmetric encryption KMS key based on the encryption context in a request for a cryptographic operation. Use this condition key to evaluate both the key and the value in the encryption context pair. To evaluate only the encryption context keys or require an encryption context regardless of keys or values, use the kms:EncryptionContextKeys condition key.

Note

Condition key values must conform to the character rules for key policies and IAM policies. Some characters that are valid in an encryption context are not valid in policies. You might not be able to use this condition key to express all valid encryption context values. For details about key policy document rules, see Key policy format. For details about IAM policy document rules, see IAM name requirements in the *IAM User Guide*.

You cannot specify an encryption context in a cryptographic operation with an asymmetric KMS key or an HMAC KMS key. Asymmetric algorithms and MAC algorithms do not support an encryption context.

To use the kms:EncryptionContext:*context-key* condition key, replace the *`context-key`* placeholder with the encryption context key. Replace the *`context-value`* placeholder with the encryption context value.

```
"kms:EncryptionContext:context-key": "context-value"
```

For example, the following condition key specifies an encryption context in which the key is `AppName` and the value is `ExampleApp` (`AppName = ExampleApp`).

```
"kms:EncryptionContext:AppName": "ExampleApp"
```

This is a single-valued condition key. The key in the condition key specifies a particular encryption context key (*context-key*). Although you can include multiple encryption context pairs in each API request, the encryption context pair with the specified *context-key* can have only one value. For example, the `kms:EncryptionContext:Department` condition key applies only to encryption context pairs with a `Department` key, and any given encryption context pair with the `Department` key can have only one value.

Do not use a set operator with the `kms:EncryptionContext:`*`context-key`* condition key. If you create a policy statement with an `Allow` action, the `kms:EncryptionContext:`*`context-key`* condition key, and the `ForAllValues` set operator, the condition allows requests with no encryption context and requests with encryption context pairs that are not specified in the policy condition.

> ⚠ **Warning**
>
> Do not use a `ForAnyValue` or `ForAllValues` set operator with this single-valued condition key. These set operators can create a policy condition that does not require values you intend to require and allows values you intend to forbid.
> If you create or update a policy that includes a `ForAllValues` set operator with the kms:EncryptionContext:*context-key*, AWS KMS returns the following error message:
> `OverlyPermissiveCondition:EncryptionContext: Using the ForAllValues set operator with a single-valued condition key matches requests without the specified encryption context or with an unspecified encryption context. To fix, remove ForAllValues.`

To require a particular encryption context pair, use the `kms:EncryptionContext:`*`context-key`* condition key with the `StringEquals` operator .

The following example key policy statement allows principals who can assume the role to use the KMS key in a `GenerateDataKey` request only when the encryption context in the request includes the `AppName:ExampleApp` pair. Other encryption context pairs are permitted.

The key name is not case sensitive. The case sensitivity of the value is determined by the condition operator, such as `StringEquals`. For details, see Case sensitivity of the encryption context condition.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
  },
  "Action": "kms:GenerateDataKey",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:EncryptionContext:AppName": "ExampleApp"
    }
  }
}
```

To require an encryption context pair and forbid all other encryption context pairs, use both kms:EncryptionContext:*context-key* and [kms:EncryptionContextKeys](#) in the policy statement. The following key policy statement uses the `kms:EncryptionContext:AppName` condition to require the `AppName=ExampleApp` encryption context pair in the request. It also uses a `kms:EncryptionContextKeys` condition key with the `ForAllValues` set operator to allow only the `AppName` encryption context key.

The `ForAllValues` set operator limits encryption context keys in the request to `AppName`. If the `kms:EncryptionContextKeys` condition with the `ForAllValues` set operator was used alone in a policy statement, this set operator would allow requests with no encryption context. However, if the request had no encryption context, the `kms:EncryptionContext:AppName` condition would fail. For details about the `ForAllValues` set operator, see [Using multiple keys and values](#) in the *IAM User Guide*.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/KeyUsers"
  },
  "Action": "kms:GenerateDataKey",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:EncryptionContext:AppName": "ExampleApp"
    },
    "ForAllValues:StringEquals": {
      "kms:EncryptionContextKeys": [
        "AppName"
      ]
    }
  }
}
```

You can also use this condition key to deny access to a KMS key for a particular operation. The following example key policy statement uses a `Deny` effect to forbid the principal from using the KMS key if the encryption context in the request includes a `Stage=Restricted` encryption context pair. This condition allows a request with other encryption context pairs, including encryption context pairs with the `Stage` key and other values, such as `Stage=Test`.

```
{
  "Effect": "Deny",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
  },
  "Action": "kms:GenerateDataKey",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:EncryptionContext:Stage": "Restricted"
    }
  }
}
```

Using multiple encryption context pairs

You can require or forbid multiple encryption context pairs. You can also require one of several encryption context pairs. For details about the logic used to interpret these conditions, see Creating a condition with multiple keys or values in the IAM User Guide.

> **Note**
>
> Earlier versions of this topic displayed policy statements that used the `ForAnyValue` and `ForAllValues` set operators with the kms:EncryptionContext:*context-key* condition key. Using a set operator with a single-valued condition key can result in policies that allow requests with no encryption context and unspecified encryption context pairs.
> For example, a policy condition with the `Allow` effect, the `ForAllValues` set operator, and the `"kms:EncryptionContext:Department": "IT"` condition key does not

> limit the encryption context to the "Department=IT" pair. It allows requests with no encryption context and requests with unspecified encryption context pairs, such as `Stage=Restricted`.
> Please review your policies and eliminate the set operator from any condition with kms:EncryptionContext:*context-key*. Attempts to create or update a policy with this format fail with an `OverlyPermissiveCondition` exception. To resolve the error, delete the set operator.

To require multiple encryption context pairs, list the pairs in the same condition. The following example key policy statement requires two encryption context pairs, `Department=IT` and `Project=Alpha`. Because the conditions have different keys (`kms:EncryptionContext:Department` and `kms:EncryptionContext:Project`), they are implicitly connected by an AND operator. Other encryption context pairs are permitted, but not required.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
  },
  "Action": "kms:Decrypt",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:EncryptionContext:Department": "IT",
      "kms:EncryptionContext:Project": "Alpha"
    }
  }
}
```

To require one encryption context pair OR another pair, place each condition key in a separate policy statement. The following example key policy requires `Department=IT` *or* `Project=Alpha` pairs, or both. Other encryption context pairs are permitted, but not required.

```
{
 "Effect": "Allow",
 "Principal": {
  "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
 },
 "Action": "kms:GenerateDataKey",
 "Resource": "*",
 "Condition": {
  "StringEquals": {
   "kms:EncryptionContext:Department": "IT"
  }
 }
},
{
 "Effect": "Allow",
 "Principal": {
  "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
 },
 "Action": "kms:GenerateDataKey",
 "Resource": "*",
 "Condition": {
  "StringEquals": {
   "kms:EncryptionContext:Project": "Alpha"
  }
 }
}
```

To require particular encryption pairs and exclude all other encryption context pairs, use both kms:EncryptionContext:*context-key* and `kms:EncryptionContextKeys` in the policy statement. The following key policy statement uses the kms:EncryptionContext:*context-key* condition to require an encryption context with both `Department=IT` *and* `Project=Alpha` pairs. It uses a `kms:EncryptionContextKeys` condition key with the `ForAllValues` set operator to allow only the `Department` and `Project` encryption context keys.

The `ForAllValues` set operator limits encryption context keys in the request to `Department` and `Project`. If it were used alone in a condition, this set operator would allow requests with no encryption context, but in this configuration, the kms:EncryptionContext:*context-key* in this condition would fail.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
  },
  "Action": "kms:GenerateDataKey",
  "Resource": "*",
  "Condition": {
      "StringEquals": {
        "kms:EncryptionContext:Department": "IT",
        "kms:EncryptionContext:Project": "Alpha"
      },
      "ForAllValues:StringEquals": {
        "kms:EncryptionContextKeys": [
          "Department",
          "Project"
        ]
      }
    }
}
```

You can also forbid multiple encryption context pairs. The following example key policy statement uses a `Deny` effect to forbid the principal from using the KMS keys if the encryption context in the request includes a `Stage=Restricted` or `Stage=Production`.pair.

Multiple values (`Restricted` and `Production`) for the same key (`kms:EncryptionContext:Stage`) are implicitly connected by a OR. For details, see Evaluation logic for conditions with multiple keys or values in the *IAM User Guide*.

```
{
  "Effect": "Deny",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
  },
  "Action": "kms:GenerateDataKey",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:EncryptionContext:Stage": [
         "Restricted",
         "Production"
      ]
    }
  }
}
```

Case sensitivity of the encryption context condition

The encryption context that is specified in a decryption operation must be an exact, case-sensitive match for the encryption context that is specified in the encryption operation. Only the order of pairs in an encryption context with multiple pair can vary.

However, in policy conditions, the condition key is not case sensitive. The case sensitivity of the condition value is determined by the policy condition operator that you use, such as `StringEquals` or `StringEqualsIgnoreCase`.

As such, the condition key, which consists of the `kms:EncryptionContext:` prefix and the *`context-key`* replacement, is not case sensitive. A policy that uses this condition does not check the case of either element of the condition key. The case sensitivity of the value, that is, the *`context-value`* replacement, is determined by the policy condition operator.

For example, the following policy statement allows the operation when the encryption context includes an `Appname` key, regardless of its capitalization. The `StringEquals` condition requires that `ExampleApp` be capitalized as it is specified.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
  },
  "Action": "kms:Decrypt",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:EncryptionContext:Appname": "ExampleApp"
    }
  }
}
```

To require a case-sensitive encryption context key, use the kms:EncryptionContextKeys policy condition with a case-sensitive condition operator, such as `StringEquals`. In this policy condition, because the encryption context key is the value in this policy condition, its case sensitivity is determined by the condition operator.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
  },
  "Action": "kms:GenerateDataKey",
  "Resource": "*",
  "Condition": {
    "ForAnyValue:StringEquals": {
      "kms:EncryptionContextKeys": "AppName"
    }
  }
}
```

To require a case-sensitive evaluation of both the encryption context key and value, use the `kms:EncryptionContextKeys` and kms:EncryptionContext:*context-key* policy conditions together in the same policy statement. The case-sensitive condition operator (such as `StringEquals`) always applies to the value of the condition. The encryption context key (such as `AppName`) is the value of the `kms:EncryptionContextKeys` condition. The encryption context value (such as `ExampleApp`) is the value of the kms:EncryptionContext:*context-key* condition.

For example, in the following example key policy statement, because the `StringEquals` operator is case sensitive, both the encryption context key and the encryption context value are case sensitive.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
  },
  "Action": "kms:GenerateDataKey",
  "Resource": "*",
  "Condition": {
    "ForAnyValue:StringEquals": {
      "kms:EncryptionContextKeys": "AppName"
    },
    "StringEquals": {
      "kms:EncryptionContext:AppName": "ExampleApp"
    }
  }
}
```

Using variables in an encryption context condition

The key and value in an encryption context pair must be simple literal strings. They cannot be integers or objects, or any type that is not fully resolved. If you use a different type, such as an integer or float, AWS KMS interprets it as a literal string.

```
"encryptionContext": {
    "department": "10103.0"
}
```

However, the value of the `kms:EncryptionContext:*context-key*` condition key can be an IAM policy variable. These policy variables are resolved at runtime based on values in the request. For example, `aws:CurrentTime` resolves to the time of the request and `aws:username` resolves to the friendly name of the caller.

You can use these policy variables to create a policy statement with a condition that requires very specific information in an encryption context, such as the caller's user name. Because it contains a variable, you can use the same policy statement for all users who can assume the role. You don't have to write a separate policy statement for each user.

Consider a situation where you want to all users who can assume a role to use the same KMS key to encrypt and decrypt their data. However, you want to allow them to decrypt only the data that they encrypted. Start by requiring that every request to AWS KMS include an encryption context where the key is `user` and the value is the caller's AWS user name, such as the following one.

```
"encryptionContext": {
    "user": "bob"
}
```

Then, to enforce this requirement, you can use a policy statement like the one in the following example. This policy statement gives the `TestTeam` role permission to encrypt and decrypt data with the KMS key. However, the permission is valid only when the encryption context in the request includes a `"user": "*<username>*"` pair. To represent the user name, the condition uses the `aws:username` policy variable.

When the request is evaluated, the caller's user name replaces the variable in the condition. As such, the condition requires an encryption context of `"user": "bob"` for "bob" and `"user": "alice"` for "alice."

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/TestTeam"
  },
  "Action": [
    "kms:Decrypt",
    "kms:Encrypt"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:EncryptionContext:user": "${aws:username}"
    }
  }
}
```

You can use an IAM policy variable only in the value of the `kms:EncryptionContext:`*`context-key`* condition key. You cannot use a variable in the key.

You can also use provider-specific context keys in variables. These context keys uniquely identify users who logged into AWS by using web identity federation.

Like all variables, these variables can be used only in the `kms:EncryptionContext:`*`context-key`* policy condition, not in the actual encryption context. And they can be used only in the value of the condition, not in the key.

For example, the following key policy statement is similar to the previous one. However, the condition requires an encryption context where the key is `sub` and the value uniquely identifies a user logged into an Amazon Cognito user pool. For details about identifying users and roles in Amazon Cognito, see IAM Roles in the Amazon Cognito Developer Guide.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/TestTeam"
  },
  "Action": [
    "kms:Decrypt",
    "kms:Encrypt"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
       "kms:EncryptionContext:sub": "${cognito-identity.amazonaws.com:sub}"
    }
  }
}
```

See also

- the section called “kms:EncryptionContextKeys”
- the section called “kms:GrantConstraintType”

kms:EncryptionContextKeys

AWS KMS condition keys	Condition type	Value type	API operations	Policy type

`kms:EncryptionContextKeys`	String (list)	Multi-valued	`CreateGrant`
			`Decrypt`
			`Encrypt`
			`GenerateDataKey`
			`GenerateDataKeyPair`
			`GenerateDataKeyPairWithoutPlaintext`
			`GenerateDataKeyWithoutPlaintext`
			`ReEncrypt`
				Key policies and IAM policies

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
			`RetireGrant`	

You can use the `kms:EncryptionContextKeys` condition key to control access to a symmetric encryption KMS key based on the encryption context in a request for a cryptographic operation. Use this condition key to evaluate only the key in each encryption context pair. To evaluate both the key and the value in the encryption context, use the `kms:EncryptionContext:`*`context-key`* condition key.

You cannot specify an encryption context in a cryptographic operation with an asymmetric KMS key or an HMAC KMS key. Asymmetric algorithms and MAC algorithms do not support an encryption context.

> **Note**
>
> Condition key values, including an encryption context key, must conform to the character and encoding rules for AWS KMS key policies. You might not be able to use this condition key to express all valid encryption context keys. For details about key policy document rules, see Key policy format. For details about IAM policy document rules, see IAM name requirements in the *IAM User Guide*.

This is a multi-valued condition key. You can specify multiple encryption context pairs in each API request. `kms:EncryptionContextKeys` compares the encryption context keys in the request to the set of encryption context keys in the policy. To determine how these sets are compared, you must provide a `ForAnyValue` or `ForAllValues` set operator in the policy condition. For details about the set operators, see Using multiple keys and values in the IAM User Guide.

- `ForAnyValue`: At least one encryption context key in the request must match an encryption context key in the policy condition. Other encryption context keys are permitted. If the request has no encryption context, the condition is not satisfied.
- `ForAllValues`: Every encryption context key in the request must match an encryption context key in the policy condition. This set operator limits the encryption context keys to those in the policy condition. It doesn't require any encryption context keys, but it forbids unspecified encryption context keys.

The following example key policy statement uses the kms:EncryptionContextKeys condition key with the ForAnyValue set operator. This policy statement allows use of a KMS key for the specified operations, but only when at least one of the encryption context pairs in the request includes the AppName key, regardless of its value.

For example, this key policy statement allows a GenerateDataKey request with two encryption context pairs, AppName=Helper and Project=Alpha, because the first encryption context pair satisfies the condition. A request with only Project=Alpha or with no encryption context would fail.

Because the StringEquals condition operation is case sensitive, this policy statement requires the spelling and case of the encryption context key. But you can use a condition operator that ignores the case of the key, such as StringEqualsIgnoreCase.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
  },
  "Action": [
    "kms:Encrypt",
    "kms:GenerateDataKey*"
  ],
  "Resource": "*",
  "Condition": {
    "ForAnyValue:StringEquals": {
      "kms:EncryptionContextKeys": "AppName"
    }
  }
}
```

You can also use the kms:EncryptionContextKeys condition key to require an encryption context (any encryption context) in cryptographic operations that use the KMS key;.

The following example key policy statement uses the kms:EncryptionContextKeys condition key with the Null condition operator to allow access to a KMS key only when encryption context in the API request is not null. This condition does not check the keys or values of the encryption context. It only verifies that the encryption context exists.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
  },
  "Action": [
    "kms:Encrypt",
    "kms:GenerateDataKey*"
  ],
  "Resource": "*",
  "Condition": {
    "Null": {
      "kms:EncryptionContextKeys": false
    }
  }
}
```

See also

- kms:EncryptionContext:*context-key*
- kms:GrantConstraintType

kms:ExpirationModel

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
`kms:ExpirationModel`	String	Single-valued	`ImportKeyMaterial`	Key policies and IAM policies

The `kms:ExpirationModel` condition key controls access to the ImportKeyMaterial operation based on the value of the ExpirationModel parameter in the request.

`ExpirationModel` is an optional parameter that determines whether the imported key material expires. Valid values are `KEY_MATERIAL_EXPIRES` and `KEY_MATERIAL_DOES_NOT_EXPIRE`. `KEY_MATERIAL_EXPIRES` is the default value.

The expiration date and time is determined by the value of the ValidTo parameter. The `ValidTo` parameter is required unless the value of the `ExpirationModel` parameter is `KEY_MATERIAL_DOES_NOT_EXPIRE`. You can also use the kms:ValidTo condition key to require a particular expiration date as a condition for access.

The following example policy statement uses the `kms:ExpirationModel` condition key to allow users to import key material into a KMS key only when the request includes the `ExpirationModel` parameter and its value is `KEY_MATERIAL_DOES_NOT_EXPIRE`.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
  },
  "Action": "kms:ImportKeyMaterial",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ExpirationModel": "KEY_MATERIAL_DOES_NOT_EXPIRE"
    }
  }
}
```

You can also use the `kms:ExpirationModel` condition key to allow users to import key material only when the key material expires. The following example key policy statement uses the `kms:ExpirationModel` condition key with the Null condition operator to allow users to import key material only when the request does not have an `ExpirationModel` parameter. The default value for ExpirationModel is `KEY_MATERIAL_EXPIRES`.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
  },
  "Action": "kms:ImportKeyMaterial",
  "Resource": "*",
  "Condition": {
    "Null": {
      "kms:ExpirationModel": true
    }
  }
}
```

See also

- kms:ValidTo

- kms:WrappingAlgorithm
- kms:WrappingKeySpec

kms:GrantConstraintType

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
kms:GrantConstraintType	String	Single-valued	CreateGrant RetireGrant	Key policies and IAM policies

You can use this condition key to control access to the CreateGrant operation based on the type of grant constraint in the request.

When you create a grant, you can optionally specify a grant constraint to allow the operations that the grant permit only when a particular encryption context is present. The grant constraint can be one of two types: `EncryptionContextEquals` or `EncryptionContextSubset`. You can use this condition key to check that the request contains one type or the other.

> ⚠ **Important**
>
> Do not include confidential or sensitive information in this field. This field may be displayed in plaintext in CloudTrail logs and other output.

The following example key policy statement uses the `kms:GrantConstraintType` condition key to allow users to create grants only when the request includes an `EncryptionContextEquals` grant constraint. The example shows a policy statement in a key policy.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
  },
  "Action": "kms:CreateGrant",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:GrantConstraintType": "EncryptionContextEquals"
    }
  }
}
```

See also

- kms:EncryptionContext:*context-key*
- kms:EncryptionContextKeys
- kms:GrantIsForAWSResource
- kms:GrantOperations
- kms:GranteePrincipal
- kms:RetiringPrincipal

kms:GrantIsForAWSResource

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
kms:GrantIsForAWSResource	Boolean	Single-valued	CreateGrant ListGrants RevokeGrant	Key policies and IAM policies

Allows or denies permission for the CreateGrant, ListGrants, or RevokeGrant operations only when an AWS service integrated with AWS KMS calls the operation on the user's behalf. This policy condition doesn't allow the user to call these grant operations directly.

The following example key policy statement uses the `kms:GrantIsForAWSResource` condition key. It allows AWS services that are integrated with AWS KMS, such as Amazon EBS, to create grants on this KMS key on behalf of the specified principal.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
  },
  "Action": "kms:CreateGrant",
  "Resource": "*",
  "Condition": {
    "Bool": {
      "kms:GrantIsForAWSResource": true
    }
  }
}
```

See also

- kms:GrantConstraintType
- kms:GrantOperations
- kms:GranteePrincipal
- kms:RetiringPrincipal

kms:GrantOperations

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
`kms:GrantOperations`	String	Multi-valued	`CreateGrant`	Key policies and IAM policies

You can use this condition key to control access to the CreateGrant operation based on the grant operations in the request. For example, you can allow users to create grants that delegate permission to encrypt but not decrypt. For more information about grants, see Using grants.

This is a multi-valued condition key. `kms:GrantOperations` compares the set of grant operations in the `CreateGrant` request to the set of grant operations in the policy. To determine how these sets are compared, you must provide a `ForAnyValue` or `ForAllValues` set operator in the policy condition. For details about the set operators, see Using multiple keys and values in the IAM User Guide.

- `ForAnyValue`: At least one grant operation in the request must match one of the grant operations in the policy condition. Other grant operations are permitted.

- ForAllValues: Every grant operation in the request must match a grant operation in the policy condition. This set operator limits the grant operations to those specified in the policy condition. It doesn't require any grant operations, but it forbids unspecified grant operations.

  ForAllValues also returns true when there are no grant operations in the request, but `CreateGrant` doesn't permit it. If the `Operations` parameter is missing or has a null value, the `CreateGrant` request fails.

The following example key policy statement uses the `kms:GrantOperations` condition key to to create grants only when the grant operations are `Encrypt`, `ReEncryptTo`, or both. If the grant includes any other operations, the `CreateGrant` request fails.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
  },
  "Action": "kms:CreateGrant",
  "Resource": "*",
  "Condition": {
    "ForAllValues:StringEquals": {
      "kms:GrantOperations": [
        "Encrypt",
        "ReEncryptTo"
      ]
    }
  }
}
```

If you change the set operator in the policy condition to `ForAnyValue`, the policy statement would require that at least one of the grant operations in the grant is `Encrypt` or `ReEncryptTo`, but it would allow other grant operations, such as `Decrypt` or `ReEncryptFrom`.

See also

- kms:GrantConstraintType
- kms:GrantIsForAWSResource
- kms:GranteePrincipal
- kms:RetiringPrincipal

kms:GranteePrincipal

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
kms:GranteePrincipal	String	Single-valued	CreateGrant	IAM and key policies

You can use this condition key to control access to the CreateGrant operation based on the value of the GranteePrincipal parameter in the request. For example, you can to create grants to use a KMS key only when the grantee principal in the CreateGrant request matches the principal specified in the condition statement.

To specify the grantee principal, use the Amazon Resource Name (ARN) of an AWS principal. Valid principals include AWS accounts, IAM users, IAM roles, federated users, and assumed role users. For help with the ARN syntax for a principal, see IAM ARNs in the *IAM User Guide*.

The following example key policy statement uses the kms:GranteePrincipal condition key to to create grants for a KMS key only when the grantee principal in the grant is the LimitedAdminRole.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
  },
  "Action": "kms:CreateGrant",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:GranteePrincipal": "arn:aws:iam::111122223333:role/LimitedAdminRole"
    }
  }
}
```

See also

- kms:GrantConstraintType

- kms:GrantIsForAWSResource
- kms:GrantOperations
- kms:RetiringPrincipal

kms:KeyAgreementAlgorithm

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
kms:KeyAgreementAlgorithm	String	Single-valued	DeriveSharedSecret	Key policies and IAM policies

You can use the kms:KeyAgreementAlgorithm condition key to control access to the DeriveSharedSecret operation based on the value of the KeyAgreementAlgorithm parameter in the request. The only valid value for KeyAgreementAlgorithm is ECDH.

For example, the following key policy statement uses the kms:KeyAgreementAlgorithm condition key to deny all access to DeriveSharedSecret unless the KeyAgreementAlgorithm is ECDH.

```
{
       "Effect": "Deny",
       "Principal": {
         "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
       },
       "Action": "kms:DeriveSharedSecret",
       "Resource": "*",
       "Condition": {
            "StringNotEquals": {
               "kms:KeyAgreementAlgorithm": "ECDH"
         }
       }
}
```

See also

- the section called “kms:KeyUsage”

kms:KeyOrigin

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
kms:KeyOrigin	String	Single-valued	CreateKey KMS key resource operations	IAM policies Key policies and IAM policies

The kms:KeyOrigin condition key controls access to operations based on the value of the Origin property of the KMS key that is created by or used in the operation. It works as a resource condition or a request condition.

You can use this condition key to control access to the CreateKey operation based on the value of the Origin parameter in the request. Valid values for Origin are AWS_KMS, AWS_CLOUDHSM, and EXTERNAL.

For example, you can to create a KMS key only when the key material is generated in AWS KMS (AWS_KMS), only when the key material is generated in an AWS CloudHSM cluster that is associated with a custom key store (AWS_CLOUDHSM), or only when the key material is imported from an external source (EXTERNAL).

The following example key policy statement uses the kms:KeyOrigin condition key to to create a KMS key only when AWS KMS creates the key material.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
      },
      "Action": "kms:CreateKey",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:KeyOrigin": "AWS_KMS"
        }
      }
    },
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:GenerateDataKey",
        "kms:GenerateDataKeyWithoutPlaintext",
        "kms:GenerateDataKeyPair",
        "kms:GenerateDataKeyPairWithoutPlaintext",
        "kms:ReEncrypt*"
      ],
      "Resource": "arn:aws:kms:us-west-2:111122223333:key/*",
      "Condition": {
        "StringEquals": {
          "kms:KeyOrigin": "AWS_CLOUDHSM"
        }
      }
    }
  ]
}
```

You can also use the `kms:KeyOrigin` condition key to control access to operations that use or manage a KMS key based on the `Origin` property of the KMS key used for the operation. The operation must be a *KMS key resource operation*, that is, an operation that is authorized for a particular KMS key. To identify the KMS key resource operations, in the Actions and Resources Table, look for a value of `KMS key` in the `Resources` column for the operation.

For example, the following IAM policy allows principals to perform the specified KMS key resource operations, but only with KMS keys in the account that were created in a custom key store.

```
{
  "Effect": "Allow",
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:GenerateDataKey",
    "kms:GenerateDataKeyWithoutPlaintext",
    "kms:GenerateDataKeyPair",
    "kms:GenerateDataKeyPairWithoutPlaintext",
    "kms:ReEncrypt*"
  ],
  "Resource": "arn:aws:kms:us-west-2:111122223333:key/*",
  "Condition": {
    "StringEquals": {
      "kms:KeyOrigin": "AWS_CLOUDHSM"
    }
  }
}
```

See also

- kms:BypassPolicyLockoutSafetyCheck
- kms:KeySpec
- kms:KeyUsage

kms:KeySpec

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
`kms:KeySpec`	String	Single-valued	`CreateKey` KMS key resource operations	IAM policies Key policies and IAM policies

The `kms:KeySpec` condition key controls access to operations based on the value of the `KeySpec` property of the KMS key that is created by or used in the operation.

You can use this condition key in an IAM policy to control access to the CreateKey operation based on the value of the KeySpec parameter in a `CreateKey` request. For example, you can use this condition to allow users to create only symmetric encryption KMS keys or only HMAC KMS keys.

The following example IAM policy statement uses the `kms:KeySpec` condition key to allow the principals to create only RSA asymmetric KMS keys. The permission is valid only when the `KeySpec` in the request begins with `RSA_`.

```
{
  "Effect": "Allow",
  "Action": "kms:CreateKey",
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "kms:KeySpec": "RSA_*"
    }
  }
}
```

You can also use the `kms:KeySpec` condition key to control access to operations that use or manage a KMS key based on the `KeySpec` property of the KMS key used for the operation. The operation must be a *KMS key resource operation*, that is, an operation that is authorized for a particular KMS key. To identify the KMS key resource operations, in the Actions and Resources Table, look for a value of `KMS key` in the `Resources` column for the operation.

For example, the following IAM policy allows principals to perform the specified KMS key resource operations, but only with symmetric encryption KMS keys in the account.

```
{
  "Effect": "Allow",
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:DescribeKey"
  ],
  "Resource": "arn:aws:kms:us-west-2:111122223333:key/*",
  "Condition": {
    "StringEquals": {
      "kms:KeySpec": "SYMMETRIC_DEFAULT"
    }
  }
}
```

See also

- kms:BypassPolicyLockoutSafetyCheck
- kms:CustomerMasterKeySpec (deprecated)
- kms:DataKeyPairSpec
- kms:KeyOrigin
- kms:KeyUsage

kms:KeyUsage

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
kms:KeyUsage	String	Single-valued	CreateKey KMS key resource operations	IAM policies Key policies and IAM policies

The kms:KeyUsage condition key controls access to operations based on the value of the KeyUsage property of the KMS key that is created by or used in the operation.

You can use this condition key to control access to the CreateKey operation based on the value of the KeyUsage parameter in the request. Valid values for KeyUsage are ENCRYPT_DECRYPT, SIGN_VERIFY, GENERATE_VERIFY_MAC, and KEY_AGREEMENT.

For example, you can to create a KMS key only when the KeyUsage is ENCRYPT_DECRYPT or deny a user permission when the KeyUsage is SIGN_VERIFY.

The following example IAM policy statement uses the kms:KeyUsage condition key to to create a KMS key only when the KeyUsage is ENCRYPT_DECRYPT.

```
{
  "Effect": "Allow",
  "Action": "kms:CreateKey",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:KeyUsage": "ENCRYPT_DECRYPT"
    }
  }
}
```

You can also use the `kms:KeyUsage` condition key to control access to operations that use or manage a KMS key based on the `KeyUsage` property of the KMS key in the operation. The operation must be a *KMS key resource operation*, that is, an operation that is authorized for a particular KMS key. To identify the KMS key resource operations, in the Actions and Resources Table, look for a value of `KMS key` in the `Resources` column for the operation.

For example, the following IAM policy allows principals to perform the specified KMS key resource operations, but only with KMS keys in the account that are used for signing and verification.

```
{
  "Effect": "Allow",
  "Action": [
    "kms:CreateGrant",
    "kms:DescribeKey",
    "kms:GetPublicKey",
    "kms:ScheduleKeyDeletion"
  ],
  "Resource": "arn:aws:kms:us-west-2:111122223333:key/*",
  "Condition": {
    "StringEquals": {
      "kms:KeyUsage": "SIGN_VERIFY"
    }
  }
}
```

See also

- kms:BypassPolicyLockoutSafetyCheck
- kms:CustomerMasterKeyUsage (deprecated)
- kms:KeyOrigin
- kms:KeySpec

kms:MacAlgorithm

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
`kms:MacAlgorithm`	String	Single-valued	`GenerateMac` `VerifyMac`	Key policies and IAM policies

You can use the `kms:MacAlgorithm` condition key to control access to the GenerateMac and VerifyMac operations based on the value of the `MacAlgorithm` parameter in the request.

The following example key policy allows users who can assume the `testers` role to use the HMAC KMS key to generate and verify HMAC tags only when the MAC algorithm in the request is `HMAC_SHA_384` or `HMAC_SHA_512`. This policy uses two separate policy statements each with its own condition. If you specify more than one MAC algorithm in a single condition statement, the condition requires both algorithms, instead of one or the other.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/testers"
      },
      "Action": [
        "kms:GenerateMac",
        "kms:VerifyMac"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:MacAlgorithm": "HMAC_SHA_384"
        }
      }
    },
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/testers"
      },
      "Action": [
        "kms:GenerateMac",
        "kms:VerifyMac"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:MacAlgorithm": "HMAC_SHA_512"
        }
      }
    }
  ]
}
```

See also

- the section called “kms:EncryptionAlgorithm”
- kms:SigningAlgorithm

kms:MessageType

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
`kms:MessageType`	String	Single-valued	`Sign` `Verify`	Key policies and IAM policies

The `kms:MessageType` condition key controls access to the Sign and Verify operations based on the value of the `MessageType` parameter in the request. Valid values for `MessageType` are `RAW` and `DIGEST`.

For example, the following key policy statement uses the `kms:MessageType` condition key to to use an asymmetric KMS key to sign a message, but not a message digest.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
  },
  "Action": "kms:Sign",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:MessageType": "RAW"
    }
  }
}
```

See also

- the section called “kms:SigningAlgorithm”

kms:MultiRegion

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
`kms:MultiRegion`	Boolean	Single-valued	`CreateKey` KMS key resource operations	Key policies and IAM policies

You can use this condition key to allow operations only on single-Region keys or only on multi-Region keys. The `kms:MultiRegion` condition key controls access to AWS KMS operations on KMS keys and to the CreateKey operation based on the value of the `MultiRegion` property of the KMS key. Valid values are `true` (multi-Region), and `false` (single-Region). All KMS keys have a `MultiRegion` property.

For example, the following IAM policy statement uses the `kms:MultiRegion` condition key to allow principals to create only single-Region keys.

```
{
  "Effect": "Allow",
  "Action": "kms:CreateKey",
  "Resource": "*",
  "Condition": {
    "Bool": {
      "kms:MultiRegion": false
    }
  }
}
```

kms:MultiRegionKeyType

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
kms:MultiRegionKeyType	String	Single-valued	CreateKey KMS key resource operations	Key policies and IAM policies

You can use this condition key to allow operations only on multi-Region primary keys or only on multi-Region replica keys. The kms:MultiRegionKeyType condition key controls access to AWS KMS operations on KMS keys and the CreateKey operation based on the MultiRegionKeyType property of the KMS key. The valid values are PRIMARY and REPLICA. Only multi-Region keys have a MultiRegionKeyType property.

Typically, you use the kms:MultiRegionKeyType condition key in an IAM policy to control access to multiple KMS keys. However, because a given multi-Region key can change to primary or replica, you might want to use this condition in a key policy to allow an operation only when the particular multi-Region key is a primary or replica key.

For example, the following IAM policy statement uses the kms:MultiRegionKeyType condition key to allow principals to schedule and cancel key deletion only on multi-Region replica keys in the specified AWS account.

```
{
  "Effect": "Allow",
  "Action": [
    "kms:ScheduleKeyDeletion",
    "kms:CancelKeyDeletion"
  ],
  "Resource": "arn:aws:kms:*:111122223333:key/*",
  "Condition": {
    "StringEquals": {
      "kms:MultiRegionKeyType": "REPLICA"
    }
  }
}
```

To allow or deny access to all multi-Region keys, you can use both values or a null value with `kms:MultiRegionKeyType`. However, the kms:MultiRegion condition key is recommended for that purpose.

kms:PrimaryRegion

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
`kms:PrimaryRegion`	String (list)	Single-valued	`UpdatePrimaryRegion`	Key policies and IAM policies

You can use this condition key to limit the destination Regions in an UpdatePrimaryRegion operation. These are AWS Regions that can host your multi-Region primary keys.

The `kms:PrimaryRegion` condition key controls access to the UpdatePrimaryRegion operation based on the value of the `PrimaryRegion` parameter. The `PrimaryRegion` parameter specifies the AWS Region of the multi-Region replica key that is being promoted to primary. The value of the condition is one or more AWS Region names, such as `us-east-1` or `ap-southeast-2`, or Region name patterns, such as `eu-*`

For example, the following key policy statement uses the `kms:PrimaryRegion` condition key to allow principals to update the primary region of a multi-Region key to one of the four specified Regions.

```
{
  "Effect": "Allow",
  "Action": "kms:UpdatePrimaryRegion",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/Developer"
  },
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:PrimaryRegion": [
         "us-east-1",
         "us-west-2",
         "eu-west-3",
         "ap-southeast-2"
      ]
    }
  }
}
```

kms:ReEncryptOnSameKey

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
kms:ReEncryptOnSameKey	Boolean	Single-valued	ReEncrypt	Key policies and IAM policies

You can use this condition key to control access to the ReEncrypt operation based on whether the request specifies a destination KMS key that is the same one used for the original encryption.

For example, the following key policy statement uses the kms:ReEncryptOnSameKey condition key to to reencrypt only when the destination KMS key is the same one used for the original encryption.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
  },
  "Action": "kms:ReEncrypt*",
  "Resource": "*",
  "Condition": {
    "Bool": {
      "kms:ReEncryptOnSameKey": true
    }
  }
}
```

kms:RequestAlias

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
kms:RequestAlias	String (list)	Single-valued	Cryptographic operations DescribeKey GetPublicKey	Key policies and IAM policies

You can use this condition key to allow an operation only when the request uses a particular alias to identify the KMS key. The kms:RequestAlias condition key controls access to a KMS key used in a cryptographic operation, GetPublicKey, or DescribeKey based on the alias that identifies that KMS key in the request. (This policy condition has no effect on the GenerateRandom operation because the operation doesn't use a KMS key or alias.)

This condition supports attribute-based access control (ABAC) in AWS KMS, which lets you control access to KMS keys based on the tags and aliases of a KMS key. You can use tags and aliases to allow or deny access to a KMS key without changing policies or grants. For details, see ABAC for AWS KMS.

To specify the alias in this policy condition, use an alias name, such as alias/project-alpha, or an alias name pattern, such as alias/*test*. You cannot specify an alias ARN in the value of this condition key.

To satisfy this condition, the value of the KeyId parameter in the request must be a matching alias name or alias ARN. If the request uses a different key identifier, it does not satisfy the condition, even if identifies the same KMS key.

For example, the following key policy statement allows the principal to call the GenerateDataKey operation on the KMS key. However this is permitted only when the value of the KeyId parameter in the request is `alias/finance-key` or an alias ARN with that alias name, such as `arn:aws:kms:us-west-2:111122223333:alias/finance-key`.

```
{
  "Sid": "Key policy using a request alias condition",
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/developer"
  },
  "Action": "kms:GenerateDataKey",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:RequestAlias": "alias/finance-key"
    }
  }
}
```

You cannot use this condition key to control access to alias operations, such as CreateAlias or DeleteAlias. For information about controlling access to alias operations, see Controlling access to aliases.

kms:ResourceAliases

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
`kms:ResourceAliases`	String (list)	Multi-valued	KMS key resource operations	IAM policies only

Use this condition key to control access to a KMS key based on the aliases that are associated with the KMS key. The operation must be a *KMS key resource operation*, that is, an operation that is authorized for a particular KMS key. To identify the KMS key resource operations, in the Actions and Resources Table, look for a value of `KMS key` in the `Resources` column for the operation.

This condition supports attribute-based access control (ABAC) in AWS KMS. With ABAC, you can control access to KMS keys based on the tags that are assigned to a KMS key and the aliases that are associated with a KMS key. You can use tags and aliases to allow or deny access to a KMS key without changing policies or grants. For details, see ABAC for AWS KMS.

An alias must be unique in an AWS account and Region, but this condition lets you control access to multiple KMS keys in the same Region (using the `StringLike` comparison operator) or to multiple KMS keys in different AWS Regions of each account.

> **Note**
>
> The kms:ResourceAliases condition is effective only when the KMS key conforms to the aliases per KMS key quota. If a KMS key exceeds this quota, principals who are authorized to use the KMS key by the `kms:ResourceAliases` condition are denied access to the KMS key.

To specify the alias in this policy condition, use an alias name, such as `alias/project-alpha`, or an alias name pattern, such as `alias/*test*`. You cannot specify an alias ARN in the value of this condition key. To satisfy the condition, the KMS key used in the operation must have the specified alias. It does not matter whether or how the KMS key is identified in the request for the operation.

This is a multivalued condition key that compares the set of aliases associated with a KMS key to the set of aliases in the policy. To determine how these sets are compared, you must provide a `ForAnyValue` or `ForAllValues` set operator in the policy condition. For details about the set operators, see Using multiple keys and values in the IAM User Guide.

- ForAnyValue: At least one alias associated with the KMS key must match an alias in the policy condition. Other aliases are permitted. If the KMS key has no aliases, the condition is not satisfied.
- ForAllValues: Every alias associated with the KMS key must match an alias in the policy. This set operator limits the aliases associated with the KMS key to those in the policy condition. It doesn't require any aliases, but it forbids unspecified aliases.

For example, the following IAM policy statement allows the principal to call the GenerateDataKey operation on any KMS key in the specified AWS account that is associated with the `finance-key` alias. (The key policies of the affected KMS keys must also allow the principal's account to use them for this operation.) To indicate that the condition is satisfied when one of the many aliases that might be associated with the KMS key is `alias/finance-key`, the condition uses the `ForAnyValue` set operator.

Because the `kms:ResourceAliases` condition is based on the resource, not the request, a call to `GenerateDataKey` succeeds for any KMS key associated with the `finance-key` alias, even if the request uses a key ID or key ARN to identify the KMS key.

```
{
  "Sid": "AliasBasedIAMPolicy",
  "Effect": "Allow",
  "Action": "kms:GenerateDataKey",
  "Resource": [
    "arn:aws:kms:*:111122223333:key/*",
    "arn:aws:kms:*:444455556666:key/*"
  ],
  "Condition": {
    "ForAnyValue:StringEquals": {
      "kms:ResourceAliases": "alias/finance-key"
    }
  }
}
```

The following example IAM policy statement allows the principal to enable and disable KMS keys but only when all aliases of the KMS keys include "`Test`." This policy statement uses two conditions. The condition with the `ForAllValues` set operator requires that all aliases associated with the KMS key include "Test". The condition with the `ForAnyValue` set operator requires that the KMS key have at least one alias with "Test." Without the `ForAnyValue` condition, this policy statement would have allowed the principal to use KMS keys that had no aliases.

```
{
  "Sid": "AliasBasedIAMPolicy",
  "Effect": "Allow",
  "Action": [
    "kms:EnableKey",
    "kms:DisableKey"
  ],
  "Resource": "arn:aws:kms:*:111122223333:key/*",
  "Condition": {
    "ForAllValues:StringLike": {
      "kms:ResourceAliases": [
        "alias/*Test*"
      ]
    },
    "ForAnyValue:StringLike": {
      "kms:ResourceAliases": [
        "alias/*Test*"
      ]
    }
  }
}
```

kms:ReplicaRegion

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
`kms:ReplicaRegion`	String (list)	Single-valued	`ReplicateKey`	Key policies and IAM policies

You can use this condition key to limit the AWS Regions in which a principal can replicate a multi-Region key. The `kms:ReplicaRegion` condition key controls access to the ReplicateKey operation based on the value of the ReplicaRegion parameter in the request. This parameter specifies the AWS Region for the new replica key.

The value of the condition is one or more AWS Region names, such as `us-east-1` or `ap-southeast-2`, or name patterns, such as `eu-*`. For a list of the names of AWS Regions that AWS KMS supports, see AWS Key Management Service endpoints and quotas in the AWS General Reference.

For example, the following key policy statement uses the `kms:ReplicaRegion` condition key to allow principals to call the ReplicateKey operation only when the value of the `ReplicaRegion` parameter is one of the specified Regions.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/Administrator"
  },
  "Action": "kms:ReplicateKey"
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ReplicaRegion": [
         "us-east-1",
         "eu-west-3",
          "ap-southeast-2"
        ]
      }
    }
}
```

This condition key controls access only to the ReplicateKey operation. To control access to the UpdatePrimaryRegion operation, use the kms:PrimaryRegion condition key.

kms:RetiringPrincipal

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
`kms:RetiringPrincipal`	String (list)	Single-valued	`CreateGrant`	Key policies and IAM policies

You can use this condition key to control access to the CreateGrant operation based on the value of the RetiringPrincipal parameter in the request. For example, you can to create grants to use a KMS key only when the `RetiringPrincipal` in the `CreateGrant` request matches the `RetiringPrincipal` in the condition statement.

To specify the retiring principal, use the Amazon Resource Name (ARN) of an AWS principal. Valid principals include AWS accounts, IAM users, IAM roles, federated users, and assumed role users. For help with the ARN syntax for a principal, see IAM ARNs in the *IAM User Guide*.

The following example key policy statement allows a user to create grants for the KMS key. The `kms:RetiringPrincipal` condition key restricts the permission to `CreateGrant` requests where the retiring principal in the grant is the `LimitedAdminRole`.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
  },
  "Action": "kms:CreateGrant",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:RetiringPrincipal": "arn:aws:iam::111122223333:role/LimitedAdminRole"
    }
  }
}
```

See also

- kms:GrantConstraintType
- kms:GrantIsForAWSResource
- kms:GrantOperations
- kms:GranteePrincipal

kms:RotationPeriodInDays

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
`kms:RotationPeriodInDays`	Numeric	Single-valued	`EnableKeyRotation`	Key policies and IAM policies

You can use this condition key to limit the values that principals can specify in the `RotationPeriodInDays` parameter of a EnableKeyRotation request.

The `RotationPeriodInDays` specifies the number of days between each automatic key rotation date. AWS KMS allows you to specify a rotation period between 90 and 2560 days, but you can use the `kms:RotationPeriodInDays` condition key to further constrain the rotation period, such as enforcing a minimum rotation period within the valid range.

For example, the following key policy statement uses the `kms:RotationPeriodInDays` condition key to prevent principals from enabling key rotation if the rotation period is less than or equal to 180 days.

```
{
  "Effect": "Deny",
  "Action": "kms:EnableKeyRotation",
  "Principal": "*",
  "Resource": "*",
  "Condition" : {
      "NumericLessThanEquals" : {
        "kms:RotationPeriodInDays" : "180"
      }
  }
}
```

kms:ScheduleKeyDeletionPendingWindowInDays

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
`kms:ScheduleKeyDeletionPendingWindowInDays`	Numeric	Single-valued	`ScheduleKeyDeletion`	Key policies and IAM policies

You can use this condition key to limit the values that principals can specify in the `PendingWindowInDays` parameter of a ScheduleKeyDeletion request.

The `PendingWindowInDays` specifies the number of days that AWS KMS will wait before deleting a key. AWS KMS allows you to specify a waiting period between 7 and 30 days, but you can use the `kms:ScheduleKeyDeletionPendingWindowInDays` condition key to further constrain the waiting period, such as enforcing a minimum waiting period within the valid range.

For example, the following key policy statement uses the `kms:ScheduleKeyDeletionPendingWindowInDays` condition key to prevent principals from scheduling key deletion if the waiting period is less than or equal to 21 days.

```
{
  "Effect": "Deny",
  "Action": "kms:ScheduleKeyDeletion",
  "Principal": "*",
  "Resource": "*",
  "Condition" : {
      "NumericLessThanEquals" : {
        "kms:ScheduleKeyDeletionPendingWindowInDays" : "21"
      }
  }
}
```

kms:SigningAlgorithm

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
`kms:SigningAlgorithm`	String	Single-valued	`Sign` `Verify`	Key policies and IAM policies

You can use the `kms:SigningAlgorithm` condition key to control access to the Sign and Verify operations based on the value of the SigningAlgorithm parameter in the request. This condition key has no effect on operations performed outside of AWS KMS, such as verifying signatures with the public key in an asymmetric KMS key pair outside of AWS KMS.

The following example key policy allows users who can assume the `testers` role to use the KMS key to sign messages only when the signing algorithm used for the request is an RSASSA_PSS algorithm, such as RSASSA_PSS_SHA512.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/testers"
  },
  "Action": "kms:Sign",
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "kms:SigningAlgorithm": "RSASSA_PSS*"
    }
  }
}
```

See also

- kms:EncryptionAlgorithm
- the section called “kms:MacAlgorithm”

- the section called “kms:MessageType”

kms:ValidTo

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
`kms:ValidTo`	Timestamp	Single-valued	`ImportKeyMaterial`	Key policies and IAM policies

The `kms:ValidTo` condition key controls access to the ImportKeyMaterial operation based on the value of the ValidTo parameter in the request, which determines when the imported key material expires. The value is expressed in Unix time.

By default, the `ValidTo` parameter is required in an `ImportKeyMaterial` request. However, if the value of the ExpirationModel parameter is `KEY_MATERIAL_DOES_NOT_EXPIRE`, the `ValidTo` parameter is invalid. You can also use the kms:ExpirationModel condition key to require the `ExpirationModel` parameter or a specific parameter value.

The following example policy statement allows a user to import key material into a KMS key. The `kms:ValidTo` condition key limits the permission to `ImportKeyMaterial` requests where the `ValidTo` value is less than or equal to `1546257599.0` (December 31, 2018 11:59:59 PM).

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
  },
  "Action": "kms:ImportKeyMaterial",
  "Resource": "*",
  "Condition": {
    "NumericLessThanEquals": {
      "kms:ValidTo": "1546257599.0"
    }
  }
}
```

See also

- kms:ExpirationModel
- kms:WrappingAlgorithm
- kms:WrappingKeySpec

kms:ViaService

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
kms:ViaService	String	Single-valued	KMS key resource operations	Key policies and IAM policies

The kms:ViaService condition key limits use of an KMS key to requests from specified AWS services. You can specify one or more services in each kms:ViaService condition key. The operation must be a *KMS key resource operation*, that is, an operation that is authorized for a particular KMS key. To identify the KMS key resource operations, in the Actions and Resources Table, look for a value of KMS key in the Resources column for the operation.

For example, the following key policy statement uses the kms:ViaService condition key to allow a customer managed key to be used for the specified actions only when the request comes from Amazon EC2 or Amazon RDS in the US West (Oregon) region on behalf of ExampleRole.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
  },
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:CreateGrant",
    "kms:ListGrants",
    "kms:DescribeKey"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ViaService": [
        "ec2.us-west-2.amazonaws.com",
        "rds.us-west-2.amazonaws.com"
      ]
    }
  }
}
```

You can also use a `kms:ViaService` condition key to deny permission to use a KMS key when the request comes from particular services. For example, the following policy statement from a key policy uses a `kms:ViaService` condition key to prevent a customer managed key from being used for `Encrypt` operations when the request comes from AWS Lambda on behalf of `ExampleRole`.

```
{
  "Effect": "Deny",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
  },
  "Action": [
    "kms:Encrypt"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ViaService": [
          "lambda.us-west-2.amazonaws.com"
      ]
    }
  }
}
```

⚠ **Important**

When you use the `kms:ViaService` condition key, the service makes the request on behalf of a principal in the AWS account. These principals must have the following permissions:

- Permission to use the KMS key. The principal needs to grant these permissions to the integrated service so the service can use the customer managed key on behalf of the principal. For more information, see How AWS services use AWS KMS.
- Permission to use the integrated service. For details about giving users access to an AWS service that integrates with AWS KMS, consult the documentation for the integrated service.

All AWS managed keys use a `kms:ViaService` condition key in their key policy document. This condition allows the KMS key to be used only for requests that come from the service that created the KMS key. To see the key policy for an AWS managed key, use the GetKeyPolicy operation.

The `kms:ViaService` condition key is valid in IAM and key policy statements. The services that you specify must be integrated with AWS KMS and support the `kms:ViaService` condition key.

Services that support the `kms:ViaService` condition key

The following table lists AWS services that are integrated with AWS KMS and support the use of the `kms:ViaService` condition key in customer managed keys The services in this table might not be available in all regions. Use the `.amazonaws.com` suffix of the AWS KMS ViaService name in all AWS partitions.

Note

You might need to scroll horizontally or vertically to see all of the data in this table.

Service name	AWS KMS ViaService name
AWS App Runner	`apprunner.` *`AWS_region`* `.amazonaws.com`
AWS AppFabric	`appfabric.` *`AWS_region`* `.amazonaws.com`
Amazon AppFlow	`appflow.`*`AWS_region`* `.amazonaws.com`
AWS Application Migration Service	`mgn.`*`AWS_region`* `.amazonaws.com`

Service name	AWS KMS ViaService name
Amazon Athena	athena.*AWS_region*.amazonaws.com
AWS Audit Manager	auditmanager.*AWS_region*.amazonaws.com
Amazon Aurora	rds.*AWS_region*.amazonaws.com
AWS Backup	backup.*AWS_region*.amazonaws.com
AWS Backup Gateway	backup-gateway.*AWS_region*.amazonaws.com
Amazon Chime SDK	chimevoiceconnector.*AWS_region*.amazonaws.com
AWS CodeArtifact	codeartifact.*AWS_region*.amazonaws.com
Amazon CodeGuru Reviewer	codeguru-reviewer.*AWS_region*.amazonaws.com
Amazon Comprehend	comprehend.*AWS_region*.amazonaws.com
Amazon Connect	connect.*AWS_region*.amazonaws.com
Amazon Connect Customer Profiles	profile.*AWS_region*.amazonaws.com
Amazon Q in Connect	wisdom.*AWS_region*.amazonaws.com
AWS Database Migration Service (AWS DMS)	dms.*AWS_region*.amazonaws.com
AWS Directory Service	directoryservice.*AWS_region*.amazonaws.com
Amazon DynamoDB	dynamodb.*AWS_region*.amazonaws.com

Service name	AWS KMS ViaService name
Amazon DocumentDB	`docdb-elastic. AWS_region .amazonaws.com`
Amazon EC2 Systems Manager (SSM)	`ssm.AWS_region .amazonaws.com`
Amazon Elastic Block Store (Amazon EBS)	`ec2.AWS_region .amazonaws.com` (EBS only)
Amazon Elastic Container Registry (Amazon ECR)	`ecr.AWS_region .amazonaws.com`
Amazon Elastic File System (Amazon EFS)	`elasticfilesystem. AWS_region .amazonaws.com`
Amazon ElastiCache	Include both ViaService names in the condition key value: • `elasticache. AWS_region .amazonaws.com` • `dax.AWS_region .amazonaws.com`
AWS Elemental MediaTailor	`mediatailor. AWS_region .amazonaws.com`
AWS Entity Resolution	`entityresolution. AWS_region .amazonaws.com`
Amazon EventBridge	`events.AWS_region .amazonaws.com`
Amazon FinSpace	`finspace. AWS_region .amazonaws.com`
Amazon Forecast	`forecast. AWS_region .amazonaws.com`
Amazon FSx	`fsx.AWS_region .amazonaws.com`
AWS Glue	`glue.AWS_region .amazonaws.com`

Service name	AWS KMS ViaService name
AWS Ground Station	`groundstation. AWS_region .amazonaws.com`
Amazon GuardDuty	`malware-protection. AWS_region .amazonaws.com`
AWS HealthLake	`healthlake. AWS_region .amazonaws.com`
AWS IoT SiteWise	`iotsitewise. AWS_region .amazonaws.com`
Amazon Kendra	`kendra.AWS_region .amazonaws.com`
Amazon Keyspaces (for Apache Cassandra)	`cassandra. AWS_region .amazonaws.com`
Amazon Kinesis	`kinesis.AWS_region .amazonaws.com`
Amazon Data Firehose	`firehose. AWS_region .amazonaws.com`
Amazon Kinesis Video Streams	`kinesisvideo. AWS_region .amazonaws.com`
AWS Lambda	`lambda.AWS_region .amazonaws.com`
Amazon Lex	`lex.AWS_region .amazonaws.com`
AWS License Manager	`license-manager. AWS_region .amazonaws.com`
Amazon Location Service	`geo.AWS_region .amazonaws.com`
Amazon Lookout for Equipment	`lookoutequipment. AWS_region .amazonaws.com`

Service name	AWS KMS ViaService name
Amazon Lookout for Metrics	`lookoutmetrics.` *`AWS_region`* `.amazonaws.com`
Amazon Lookout for Vision	`lookoutvision.` *`AWS_region`* `.amazonaws.com`
Amazon Macie	`macie.`*`AWS_region`* `.amazonaws.com`
AWS Mainframe Modernization	`m2.`*`AWS_region`* `.amazonaws.com`
AWS Mainframe Modernization Application Testing	`apptest.`*`AWS_region`* `.amazonaws.com`
Amazon Managed Blockchain	`managedblockchain.` *`AWS_region`* `.amazonaws.com`
Amazon Managed Streaming for Apache Kafka (Amazon MSK)	`kafka.`*`AWS_region`* `.amazonaws.com`
Amazon Managed Workflows for Apache Airflow (MWAA)	`airflow.`*`AWS_region`* `.amazonaws.com`
Amazon MemoryDB for Redis	`memorydb.` *`AWS_region`* `.amazonaws.com`
Amazon Monitron	`monitron.` *`AWS_region`* `.amazonaws.com`
Amazon MQ	`mq.`*`AWS_region`* `.amazonaws.com`
Amazon Neptune	`rds.`*`AWS_region`* `.amazonaws.com`
Amazon Nimble Studio	`nimble.`*`AWS_region`* `.amazonaws.com`
AWS HealthOmics	`omics.`*`AWS_region`* `.amazonaws.com`
Amazon OpenSearch Service	`es.`*`AWS_region`* `.amazonaws.com` , `aoss.`*`AWS_region`* `.amazonaws.com`

Service name	AWS KMS ViaService name
AWS Proton	`proton.AWS_region .amazonaws.com`
Amazon Quantum Ledger Database (Amazon QLDB)	`qldb.AWS_region .amazonaws.com`
Amazon RDS Performance Insights	`rds.AWS_region .amazonaws.com`
Amazon Redshift	`redshift. AWS_region .amazonaws.com`
Amazon Redshift query editor V2	`sqlworkbench. AWS_region .amazonaws.com`
Amazon Redshift Serverless	`redshift-serverless. AWS_region .amazonaws.com`
Amazon Rekognition	`rekognition. AWS_region .amazonaws.com`
Amazon Relational Database Service (Amazon RDS)	`rds.AWS_region .amazonaws.com`
Amazon Replicated Data Store	`ards.AWS_region .amazonaws.com`
Amazon SageMaker	`sagemaker. AWS_region .amazonaws.com`
AWS Secrets Manager	`secretsmanager. AWS_region .amazonaws.com`
Amazon Security Lake	`securitylake. AWS_region .amazonaws.com`
Amazon Simple Email Service (Amazon SES)	`ses.AWS_region .amazonaws.com`
Amazon Simple Notification Service (Amazon SNS)	`sns.AWS_region .amazonaws.com`

Service name	AWS KMS ViaService name
Amazon Simple Queue Service (Amazon SQS)	sqs.*AWS_region* .amazonaws.com
Amazon Simple Storage Service (Amazon S3)	s3.*AWS_region* .amazonaws.com
AWS Snowball	importexport. *AWS_region* .amazonaws.com
AWS Step Functions	states.*AWS_region* .amazonaws.com
AWS Storage Gateway	storagegateway. *AWS_region* .amazonaws.com
AWS Systems Manager Incident Manager	ssm-incidents. *AWS_region* .amazonaws.com
AWS Systems Manager Incident Manager Contacts	ssm-contacts. *AWS_region* .amazonaws.com
Amazon Timestream	timestream. *AWS_region* .amazonaws.com
Amazon Translate	translate. *AWS_region* .amazonaws.com
AWS Verified Access	verified-access. *AWS_region* .amazonaws.com
Amazon WorkMail	workmail. *AWS_region* .amazonaws.com
Amazon WorkSpaces	workspaces. *AWS_region* .amazonaws.com
Amazon WorkSpaces Thin Client	thinclient. *AWS_region* .amazonaws.com
Amazon WorkSpaces Web	workspaces-web. *AWS_region* .amazonaws.com

Service name	AWS KMS ViaService name
AWS X-Ray	xray.*AWS_region* .amazonaws.com

kms:WrappingAlgorithm

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
kms:WrappingAlgorithm	String	Single-valued	GetParametersForImport	Key policies and IAM policies

This condition key controls access to the GetParametersForImport operation based on the value of the WrappingAlgorithm parameter in the request. You can use this condition to require principals to use a particular algorithm to encrypt key material during the import process. Requests for the required public key and import token fail when they specify a different wrapping algorithm.

The following example key policy statement uses the kms:WrappingAlgorithm condition key to give the example user permission to call the GetParametersForImport operation, but prevents them from using the RSAES_OAEP_SHA_1 wrapping algorithm. When the WrappingAlgorithm in the GetParametersForImport request is RSAES_OAEP_SHA_1, the operation fails.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
  },
  "Action": "kms:GetParametersForImport",
  "Resource": "*",
  "Condition": {
    "StringNotEquals": {
      "kms:WrappingAlgorithm": "RSAES_OAEP_SHA_1"
    }
  }
}
```

See also

- kms:ExpirationModel
- kms:ValidTo
- kms:WrappingKeySpec

kms:WrappingKeySpec

AWS KMS condition keys	Condition type	Value type	API operations	Policy type
kms:WrappingKeySpec	String	Single-valued	GetParametersForImport	Key policies and IAM policies

This condition key controls access to the GetParametersForImport operation based on the value of the WrappingKeySpec parameter in the request. You can use this condition to require principals to use a particular type of public key during the import process. If the request specifies a different key type, it fails.

Because the only valid value for the WrappingKeySpec parameter value is RSA_2048, preventing users from using this value effectively prevents them from using the GetParametersForImport operation.

The following example policy statement uses the kms:WrappingAlgorithm condition key to require that the WrappingKeySpec in the request is RSA_4096.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
  },
  "Action": "kms:GetParametersForImport",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:WrappingKeySpec": "RSA_4096"
    }
  }
}
```

See also

- kms:ExpirationModel
- kms:ValidTo
- kms:WrappingAlgorithm

AWS KMS condition keys for AWS Nitro Enclaves

AWS Nitro Enclaves is an Amazon EC2 capability that lets you create isolated compute environments called enclaves to protect and process highly sensitive data. AWS KMS provides condition keys to support AWS Nitro Enclaves. These conditions keys are effective only for requests to AWS KMS for a Nitro Enclave.

When you call the Decrypt, DeriveSharedSecret, GenerateDataKey, GenerateDataKeyPair, or GenerateRandom API operations with the signed attestation document from an enclave, these APIs encrypt the plaintext in the response under the public key from the attestation document, and return ciphertext instead of plaintext. This ciphertext can be decrypted only by using the private key in the enclave. For more information, see How AWS Nitro Enclaves uses AWS KMS.

The following condition keys let you limit the permissions for these operations based on the contents of the signed attestation document. Before allowing an operation, AWS KMS compares the attestation document from the enclave to the values in these AWS KMS condition keys.

kms:RecipientAttestation:ImageSha384

AWS KMS Condition Keys	Condition Type	Value type	API Operations	Policy Type

`kms:RecipientAttestation:ImageSha384`	String	Single-valued	`Decrypt`
			`DeriveSharedSecret`
			`GenerateDataKey`
			`GenerateDataKeyPair`
				Key policies and IAM policies

AWS KMS Condition Keys	Condition Type	Value type	API Operations	Policy Type
			GenerateRandom	

The kms:RecipientAttestation:ImageSha384 condition key controls access to Decrypt, DeriveSharedSecret, GenerateDataKey, GenerateDataKeyPair, and GenerateRandom with a KMS key when the image digest from the signed attestation document in the request matches the value in the condition key. The ImageSha384 value corresponds to PCR0 in the attestation document. This condition key is effective only when the Recipient parameter in the request specifies a signed attestation document for an AWS Nitro enclave.

This value is also included in CloudTrail events for requests to AWS KMS for Nitro enclaves.

> **Note**
>
> This condition key is valid in key policy statements and IAM policy statements even though it does not appear in the IAM console or the IAM *Service Authorization Reference*.

For example, the following key policy statement allows the data-processing role to use the KMS key for Decrypt, DeriveSharedSecret, GenerateDataKey, GenerateDataKeyPair, and GenerateRandom operations. The kms:RecipientAttestation:ImageSha384 condition key allows the operations only when the image digest value (PCR0) of the attestation document in the request matches the image digest value in the condition. This condition key is effective only when the Recipient parameter in the request specifies a signed attestation document for an AWS Nitro enclave.

If the request does not include a valid attestation document from an AWS Nitro enclave, permission is denied because this condition is not satisfied.

```
{
  "Sid" : "Enable enclave data processing",
  "Effect" : "Allow",
  "Principal" : {
    "AWS" : "arn:aws:iam::111122223333:role/data-processing"
  },
  "Action": [
    "kms:Decrypt",
    "kms:DeriveSharedSecret",
    "kms:GenerateDataKey",
    "kms:GenerateDataKeyPair",
    "kms:GenerateRandom"
  ],
  "Resource" : "*",
  "Condition": {
    "StringEqualsIgnoreCase": {
      "kms:RecipientAttestation:ImageSha384":
 "9fedcba8abcdef7abcdef6abcdef5abcdef4abcdef3abcdef2abcdef1abcdef1abcdef0abcdef1abcdef2abcdef3a
    }
  }
}
```

kms:RecipientAttestation:PCR<PCR_ID>

AWS KMS Condition Keys	Condition Type	Value type	API Operations	Policy Type

`kms:RecipientAttestation:PCR<PCR_ID>`	String	Single-valued	`Decrypt`
			`DeriveSharedSecret`
			`GenerateDataKey`
			`GenerateDataKeyPair`
			`GenerateRandom`
				Key policies and IAM policies

The `kms:RecipientAttestation:PCR<PCR_ID>` condition key controls access to `Decrypt`, `DeriveSharedSecret`, `GenerateDataKey`, `GenerateDataKeyPair`, and `GenerateRandom` with a KMS key only when the platform configuration registers (PCRs) from the signed attestation document in the request match the PCRs in the condition key. This condition key is effective only when the `Recipient` parameter in the request specifies a signed attestation document from an AWS Nitro enclave.

This value is also included in CloudTrail events that represent requests to AWS KMS for Nitro enclaves.

> **Note**
>
> This condition key is valid in key policy statements and IAM policy statements even though it does not appear in the IAM console or the IAM *Service Authorization Reference*.

To specify a PCR value, use the following format. Concatenate the PCR ID to the condition key name. The PCR value must be a lower-case hexadecimal string of up to 96 bytes.

```
"kms:RecipientAttestation:PCRPCR_ID": "PCR_value"
```

For example, the following condition key specifies a particular value for PCR1, which corresponds to the hash of the kernel used for the enclave and the bootstrap process.

```
kms:RecipientAttestation:PCR1:
 "0x1abcdef2abcdef3abcdef4abcdef5abcdef6abcdef7abcdef8abcdef9abcdef8abcdef7abcdef6abcdef5abcdef
```

The following example key policy statement allows the `data-processing` role to use the KMS key for the Decrypt operation.

The `kms:RecipientAttestation:PCR` condition key in this statement allows the operation only when the PCR1 value in the signed attestation document in the request matches `kms:RecipientAttestation:PCR1` value in the condition. Use the `StringEqualsIgnoreCase` policy operator to require a case-insensitive comparison of the PCR values.

If the request does not include an attestation document, permission is denied because this condition is not satisfied.

```
{
  "Sid" : "Enable enclave data processing",
  "Effect" : "Allow",
  "Principal" : {
    "AWS" : "arn:aws:iam::111122223333:role/data-processing"
  },
  "Action": "kms:Decrypt",
  "Resource" : "*",
  "Condition": {
    "StringEqualsIgnoreCase": {
      "kms:RecipientAttestation:PCR1":
 "0x1de4f2dcf774f6e3b679f62e5f120065b2e408dcea327bd1c9dddaea6664e7af7935581474844767453082c6f15
    }
  }
}
```

ABAC for AWS KMS

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. AWS KMS supports ABAC by allowing you to control access to your customer managed keys based on the tags and aliases associated with the KMS keys. The tag and alias condition keys that enable ABAC in AWS KMS provide a powerful and flexible way to authorize principals to use KMS keys without editing policies or managing grants. But you should use these feature with care so principals aren't inadvertently allowed or denied access.

If you use ABAC, be aware that permission to manage tags and aliases is now an access control permission. Be sure that you know the existing tags and aliases on all KMS keys before you deploy a policy that depends on tags or aliases. Take reasonable precautions when adding, deleting, and updating aliases, and when tagging and untagging keys. Give permissions to manage tags and aliases only to principals who need them, and limit the tags and aliases they can manage.

Notes

When using ABAC for AWS KMS, be cautious about giving principals permission to manage tags and aliases. Changing a tag or alias might allow or deny permission to a KMS key. Key administrators who don't have permission to change key policies or create grants can control access to KMS keys if they have permission to manage tags or aliases.
It might take up to five minutes for tag and alias changes to affect KMS key authorization. Recent changes might be visible in API operations before they affect authorization.
To control access to a KMS key based on its alias, you must use a condition key. You cannot use an alias to represent a KMS key in the `Resource` element of a policy statement. When an alias appears in the `Resource` element, the policy statement applies to the alias, not to the associated KMS key.

Learn more

- For details about AWS KMS support for ABAC, including examples, see Using aliases to control access to KMS keys and Using tags to control access to KMS keys.
- For more general information about using tags to control access to AWS resources, see What is ABAC for AWS? and Controlling Access to AWS Resources Using Resource Tags in the *IAM User Guide*.

ABAC condition keys for AWS KMS

To authorize access to KMS keys based on their tags and aliases, use the following condition keys in a key policy or IAM policy.

ABAC condition key	Description	Policy type	AWS KMS operations
aws:ResourceTag	Tag (key and value) on the KMS key matches the tag (key and value) or tag pattern in the policy	IAM policy only	KMS key resource operations [2]
aws:RequestTag/*tag-key*	Tag (key and value) in the request matches the tag (key and value) or tag pattern in the policy	Key policy and IAM policies[1]	TagResource, UntagResource
aws:TagKeys	Tag keys in the request match the tag keys in the policy	Key policy and IAM policies[1]	TagResource, UntagResource
kms:ResourceAliases	Aliases associated with the KMS key match the aliases or alias patterns in the policy	IAM policy only	KMS key resource operations [2]

ABAC condition key	Description	Policy type	AWS KMS operations
kms:RequestAlias	Alias that represents the KMS key in the request matches the alias or alias patterns in the policy.	Key policy and IAM policies[1]	Cryptographic operations, DescribeKey, GetPublicKey

[1] Any condition key that can be used in a key policy can also be used in an IAM policy, but only if the key policy allows it.

[2] A *KMS key resource operation* is an operation authorized for a particular KMS key. To identify the KMS key resource operations, in the AWS KMS permissions table, look for a value of KMS key in the `Resources` column for the operation.

For example, you can use these condition keys to create the following policies.

- An IAM policy with `kms:ResourceAliases` that allows permission to use KMS keys with a particular alias or alias pattern. This is a bit different from policies that rely on tags: Although you can use alias patterns in a policy, each alias must be unique in an AWS account and Region. This allows you to apply a policy to a select set of KMS keys without listing the key ARNs of the KMS keys in the policy statement. To add or remove KMS keys from the set, change the alias of the KMS key.
- A key policy with `kms:RequestAlias` that allows principals to use a KMS key in a `Encrypt` operation, but only when the `Encrypt` request uses that alias to identify the KMS key.
- An IAM policy with `aws:ResourceTag/*tag-key*` that denies permission to use KMS keys with a particular tag key and tag value. This lets you apply a policy to a select set of KMS keys without listing the key ARNs of the KMS keys in the policy statement. To add or remove KMS keys from the set, tag or untag the KMS key.
- An IAM policy with `aws:RequestTag/*tag-key*` that allows principals to delete only `"Purpose"="Test"` KMS key tags.
- An IAM policy with `aws:TagKeys` that denies permission to tag or untag a KMS key with a `Restricted` tag key.

ABAC makes access management flexible and scalable. For example, you can use the `aws:ResourceTag/*tag-key*` condition key to create an IAM policy that allows principals to use a KMS key for specified operations only when the KMS key has a `Purpose=Test` tag. The policy applies to all KMS keys in all Regions of the AWS account.

When attached to a user or role, the following IAM policy allows principals to use all existing KMS keys with a `Purpose=Test` tag for the specified operations. To provide this access to new or existing KMS keys, you don't need to change the policy. Just attach the `Purpose=Test` tag to the KMS keys. Similarly, to remove this access from KMS keys with a `Purpose=Test` tag, edit or delete the tag.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AliasBasedIAMPolicy",
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:Encrypt",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "arn:aws:kms:*:111122223333:key/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Purpose": "Test"
        }
      }
    }
  ]
}
```

However, if you use this feature, be careful when managing tags and aliases. Adding, changing, or deleting a tag or alias can inadvertently allow or deny access to a KMS key. Key administrators who don't have permission to change key policies or create grants can control access to KMS keys if they have permission to manage tags and aliases. To mitigate this risk, consider limiting permissions to manage tags and aliases. For example, you might want to allow only select principals to manage `Purpose=Test` tags. For details, see Using aliases to control access to KMS keys and Using tags to control access to KMS keys.

Tags or aliases?

AWS KMS supports ABAC with tags and aliases. Both options provide a flexible, scalable access control strategy, but they're slightly different from each other.

You might decide to use tags or use aliases based on your particular AWS use patterns. For example, if you have already given tagging permissions to most administrators, it might be easier to control an authorization strategy based on aliases. Or, if you are close to the quota for aliases per KMS key, you might prefer an authorization strategy based on tags.

The following benefits are of general interest.

Benefits of tag-based access control

- Same authorization mechanism for different types of AWS resources.

  You can use the same tag or tag key to control access to multiple resource types, such as an Amazon Relational Database Service (Amazon RDS) cluster, an Amazon Elastic Block Store (Amazon EBS) volume, and a KMS key. This feature enables several different authorization models that are more flexible than traditional role-based access control.
- Authorize access to a group of KMS keys.

  You can use tags to manage access to a group of KMS keys in the same AWS account and Region. Assign the same tag or tag key to the KMS keys that you choose. Then create a simple, easy-to-maintain policy statement that is based on the tag or tag key. To add or remove a KMS key from your authorization group, add or remove the tag; you don't need to edit the policy.

Benefits of alias-based access control

- Authorize access to cryptographic operations based on aliases.

  Most request-based policy conditions for attributes, including aws:RequestTag/*tag-key*, affect only operations that add, edit, or delete the attribute. But the kms:RequestAlias condition key controls access to cryptographic operations based on the alias used to identify the KMS key in the request. For example, you can give a principal permission to use a KMS key in a `Encrypt` operation but only when the value of the `KeyId` parameter is `alias/restricted-key-1`. To satisfy this condition requires all of the following:
  - The KMS key must be associated with that alias.
  - The request must use the alias to identify the KMS key.

  - The principal must have permission to use the KMS key subject to the `kms:RequestAlias` condition.

  This is particularly useful if your applications commonly use alias names or alias ARNs to refer to KMS keys.

- Provide very limited permissions.

  An alias must be unique in an AWS account and Region. As a result, giving principals access to a KMS key based on an alias can be much more restrictive than giving them access based on a tag. Unlike aliases, tags can be assigned to multiple KMS keys in the same account and Region. If you choose, you can use an alias pattern, such as `alias/test*`, to give principals access to a group of KMS keys in the same account and Region. However, allowing or denying access to a particular alias allows very strict control on KMS keys.

Troubleshooting ABAC for AWS KMS

Controlling access to KMS keys based on their tags and aliases is convenient and powerful. However, it's prone to a few predictable errors that you'll want to prevent.

Access changed due to tag change

If a tag is deleted or its value is changed, principals who have access to a KMS key based only on that tag will be denied access to the KMS key. This can also happen when a tag that is included in a deny policy statement is added to a KMS key. Adding a policy-related tag to a KMS key can allow access to principals who should be denied access to a KMS key.

For example, suppose that a principal has access to a KMS key based on the `Project=Alpha` tag, such as the permission provided by the following example IAM policy statement.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "IAMPolicyWithResourceTag",
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKeyWithoutPlaintext",
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:ap-southeast-1:111122223333:key/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Project": "Alpha"
        }
      }
    }
  ]
}
```

If the tag is deleted from that KMS key or the tag value is changed, the principal no longer has permission to use the KMS key for the specified operations. This might become evident when the principal tries to read or write data in an AWS service that uses a customer managed key To trace the tag change, review your CloudTrail logs for TagResource or UntagResource entries.

To restore access without updating the policy, change the tags on the KMS key. This action has minimal impact other than a brief period while it is taking effect throughout AWS KMS. To prevent an error like this one, give tagging and untagging permissions only to principals who need it and limit their tagging permissions to tags they need to manage. Before changing a tag, search policies to detect access that depends on the tag, and get KMS keys in all Regions that have the tag. You might consider creating an Amazon CloudWatch alarm when particular tags are changed.

Access change due to alias change

If an alias is deleted or associated with a different KMS key, principals who have access to the KMS key based only on that alias will be denied access to the KMS key. This can also happen when an alias that is associated with a KMS key is included in a deny policy statement. Adding a policy-related alias to a KMS key can also allow access to principals who should be denied access to a KMS key.

For example, the following IAM policy statement uses the kms:ResourceAliases condition key to allow access to KMS keys in different Regions of the account with any of the specified aliases.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AliasBasedIAMPolicy",
      "Effect": "Allow",
      "Action": [
        "kms:List*",
        "kms:Describe*",
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:*:111122223333:key/*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "kms:ResourceAliases": [
            "alias/ProjectAlpha",
            "alias/ProjectAlpha_Test",
            "alias/ProjectAlpha_Dev"
          ]
        }
      }
    }
  ]
}
```

To trace the alias change, review your CloudTrail logs for CreateAlias, UpdateAlias, and DeleteAlias entries.

To restore access without updating the policy, change the alias associated with the KMS key. Because each alias can be associated with only one KMS key in an account and Region, managing aliases is a bit more difficult than managing tags. Restoring access to some principals on one KMS key can deny the same or other principals access to a different KMS key.

To prevent this error, give alias management permissions only to principals who need it and limit their alias-management permissions to aliases they need to manage. Before updating or deleting an alias, search policies to detect access that depends on the alias, and find KMS keys in all Regions that are associated with the alias.

Access denied due to alias quota

Users who are authorized to use a KMS key by an kms:ResourceAliases condition will get an `AccessDenied` exception if the KMS key exceeds the default aliases per KMS key quota for that account and Region.

To restore access, delete aliases that are associated with the KMS key so it complies with the quota. Or use an alternate mechanism to give users access to the KMS key.

Delayed authorization change

Changes that you make to tags and aliases might take up to five minutes to affect the authorization of KMS keys. As a result, a tag or alias change might be reflected in the responses from API operations before they affect authorization. This delay is likely to be longer than the brief eventual consistency delay that affects most AWS KMS operations.

For example, you might have an IAM policy that allows certain principals to use any KMS key with a `"Purpose"="Test"` tag. Then you add the `"Purpose"="Test"` tag to a KMS key. Although the TagResource operation completes and ListResourceTags response confirms that the tag is assigned to the KMS key, the principals might not have access to the KMS key for up to five minutes.

To prevent errors, build this expected delay into your code.

Failed requests due to alias updates

When you update an alias, you associate an existing alias with a different KMS key.

Decrypt and ReEncrypt requests that specify the alias name or alias ARN might fail because the alias is now associated with a KMS key that didn't encrypt the ciphertext. This situation typically returns an `IncorrectKeyException` or `NotFoundException`. Or if the request has no `KeyId` or `DestinationKeyId` parameter, the operation might fail with `AccessDenied` exception because the caller no longer has access to the KMS key that encrypted the ciphertext.

You can trace the change by looking at CloudTrail logs for CreateAlias, UpdateAlias, and DeleteAlias log entries. You can also use the value of the `LastUpdatedDate` field in the ListAliases response to detect a change.

For example, the following ListAliases example response shows that the `ProjectAlpha_Test` alias in the `kms:ResourceAliases` condition was updated. As a result, the principals who have access based on the alias lose access to the previously associated KMS key. Instead, they have access to the newly associated KMS key.

```
$ aws kms list-aliases --query 'Aliases[?starts_with(AliasName, `alias/ProjectAlpha`)]'

{
    "Aliases": [
        {
            "AliasName": "alias/ProjectAlpha_Test",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/ProjectAlpha_Test",
            "TargetKeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
            "CreationDate": 1566518783.394,
            "LastUpdatedDate": 1605308931.903
        },
        {
            "AliasName": "alias/ProjectAlpha_Restricted",
            "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/
ProjectAlpha_Restricted",
            "TargetKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
            "CreationDate": 1553410800.010,
            "LastUpdatedDate": 1553410800.010
        }
    ]
}
```

The remedy for this change isn't simple. You can update the alias again to associate it with the original KMS key. However, before you act, you need to consider the effect of that change on the currently associated KMS key. If principals used the latter KMS key in cryptographic operations, they might need continued access to it. In this case, you might want to update the policy to ensure that principals have permission to use both of the KMS keys.

You can prevent an error like this one: Before updating an alias, search policies to detect access that depends on the alias. Then get KMS keys in all Regions that are associated with the alias. Give alias management permissions only to principals who need it and limit their alias-management permissions to aliases they need to manage.

Allowing users in other accounts to use a KMS key

You can allow users or roles in a different AWS account to use a KMS key in your account. Cross-account access requires permission in the key policy of the KMS key and in an IAM policy in the external user's account.

Cross-account permission is effective only for the following operations:

- Cryptographic operations
- CreateGrant
- DescribeKey
- GetKeyRotationStatus
- GetPublicKey
- ListGrants
- RetireGrant
- RevokeGrant

If you give a user in a different account permission for other operations, those permissions have no effect. For example, if you give a principal in a different account kms:ListKeys permission in an IAM policy, or kms:ScheduleKeyDeletion permission on a KMS key in a key policy, the user's attempts to call those operations on your resources still fail.

For details about using KMS keys in different accounts for AWS KMS operations, see the **Cross-account use** column in the AWS KMS permissions and Using KMS keys in other accounts. There is also a **Cross-account use** section in each API description in the AWS Key Management Service API Reference.

> ⚠ **Warning**
>
> Be cautious about giving principals permissions to use your KMS keys. Whenever possible, follow the *least privilege* principle. Give users access only to the KMS keys they need for only the operations they require.
> Also, be cautious about using any unfamiliar KMS key, especially a KMS key in a different account. Malicious users might give you permissions to use their KMS key to get information about you or your account.
> For information about using policies to protect the resources in your account, see Best practices for IAM policies.

To give permission to use a KMS key to users and roles in another account, you must use two different types of policies:

- The **key policy** for the KMS key must give the external account (or users and roles in the external account) permission to use the KMS key. The key policy is in the account that owns the KMS key.
- **IAM policies** in the external account must delegate the key policy permissions to its users and roles. These policies are set in the external account and give permissions to users and roles in that account.

The key policy determines who *can* have access to the KMS key. The IAM policy determines who *does* have access to the KMS key. Neither the key policy nor the IAM policy alone is sufficient—you must change both.

To edit the key policy, you can use the Policy View in the AWS Management Console or use the CreateKey or PutKeyPolicy operations. For help setting the key policy when creating a KMS key, see Creating KMS keys that other accounts can use.

For help with editing IAM policies, see Using IAM policies with AWS KMS.

For an example that shows how the key policy and IAM policies work together to allow use of a KMS key in a different account, see Example 2: User assumes role with permission to use a KMS key in a different AWS account.

You can view the resulting cross-account AWS KMS operations on the KMS key in your AWS CloudTrail logs. Operations that use KMS keys in other accounts are logged in both the caller's account and the KMS key owner account.

Topics

> **Note**
>
> The examples in this topic show how to use a key policy and IAM policy together to provide and limit access to a KMS key. These generic examples are not intended to represent the permissions that any particular AWS service requires on a KMS key. For information about the permissions that an AWS service requires, see the encryption topic in the service documentation.

Step 1: Add a key policy statement in the local account

The key policy for a KMS key is the primary determinant of who can access the KMS key and which operations they can perform. The key policy is always in the account that owns the KMS key. Unlike IAM policies, key policies do not specify a resource. The resource is the KMS key that is associated with the key policy. When providing cross-account permission, the key policy for the KMS key must give the external account (or users and roles in the external account) permission to use the KMS key.

To give an external account permission to use the KMS key, add a statement to the key policy that specifies the external account. In the `Principal` element of the key policy, enter the Amazon Resource Name (ARN) of the external account.

When you specify an external account in a key policy, IAM administrators in the external account can use IAM policies to delegate those permissions to any users and roles in the external account. They can also decide which of the actions specified in the key policy the users and roles can perform.

Permissions given to the external account and its principals are effective only if the external account is enabled in the Region that hosts the KMS key and its key policy. For information about Regions that are not enabled by default ("opt-in Regions"), see Managing AWS Regions in the *AWS General Reference*.

For example, suppose you want to allow account `444455556666` to use a symmetric encryption KMS key in account `111122223333`. To do that, add a policy statement like the one in the following example to the key policy for the KMS key in account `111122223333`. This policy statement gives the external account, `444455556666`, permission to use the KMS key in cryptographic operations for symmetric encryption KMS keys.

Note

The following example represents a sample key policy for sharing a KMS key with another account. Replace the example `Sid`, `Principal`, and `Action` values with valid values for the intended use of your KMS key.

```
{
    "Sid": "Allow an external account to use this KMS key",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::444455556666:root"
        ]
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
    ],
    "Resource": "*"
}
```

Instead of giving permission to the external account, you can specify particular external users and roles in the key policy . However, those users and roles cannot use the KMS key until IAM administrators in the external account attach the proper IAM policies to their identities. The IAM policies can give permission to all or a subset of the external users and roles that are specified in the key policy. And they can allow all or a subset of the actions specified in the key policy.

Specifying identities in a key policy restricts the permissions that IAM administrators in the external account can provide. However, it makes policy management with two accounts more complex. For example, assume that you need to add a user or role. You must add that identity to the key policy in the account that owns the KMS key and create IAM policies in the identity's account.

To specify particular external users or roles in a key policy, in the `Principal` element, enter the Amazon Resource Name (ARN) of a user or role in the external account.

For example, the following example key policy statement allows `ExampleRole` in account `444455556666` to use a KMS key in account `111122223333`. This key policy statement gives the external account, `444455556666`, permission to use the KMS key in cryptographic operations for symmetric encryption KMS keys.

> **Note**
>
> The following example represents a sample key policy for sharing a KMS key with another account. Replace the example `Sid`, `Principal`, and `Action` values with valid values for the intended use of your KMS key.

```
{
    "Sid": "Allow an external account to use this KMS key",
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::444455556666:role/ExampleRole"
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
    ],
    "Resource": "*"
}
```

> **Note**
>
> Do not set the Principal to an asterisk (*) in any key policy statement that allows permissions unless you use conditions to limit the key policy. An asterisk gives every identity in every AWS account permission to use the KMS key, unless another policy statement explicitly denies it. Users in other AWS accounts can use your KMS key whenever they have corresponding permissions in their own account.

You also need to decide which permissions you want to give to the external account. For a list of permissions on KMS keys, see AWS KMS permissions.

You can give the external account permission to use the KMS key in cryptographic operations and use the KMS key with AWS services that are integrated with AWS KMS. To do that, use the **Key Users** section of the AWS Management Console. For details, see Creating KMS keys that other accounts can use.

To specify other permissions in key policies, edit the key policy document. For example, you might want to give users permission to decrypt but not encrypt, or permission to view the KMS key but not use it. To edit the key policy document, you can use the Policy View in the AWS Management Console or the CreateKey or PutKeyPolicy operations.

Step 2: Add IAM policies in the external account

The key policy in the account that owns the KMS key sets the valid range for permissions. But, users and roles in the external account cannot use the KMS key until you attach IAM policies that delegate those permissions, or use grants to manage access to the KMS key. The IAM policies are set in the external account.

If the key policy gives permission to the external account, you can attach IAM policies to any user or role in the account. But if the key policy gives permission to specified users or roles, the IAM policy can only give those permissions to all or a subset of the specified users and roles. If an IAM policy gives KMS key access to other external users or roles, it has no effect.

The key policy also limits the actions in the IAM policy. The IAM policy can delegate all or a subset of the actions specified in the key policy. If the IAM policy lists actions that are not specified in the key policy, those permissions are not effective.

The following example IAM policy allows the principal to use the KMS key in account 111122223333 for cryptographic operations. To give this permission to users and roles in account 444455556666, attach the policy to the users or roles in account 444455556666.

> **Note**
>
> The following example represents a sample IAM policy for sharing a KMS key with another account. Replace the example `Sid`, `Resource`, and `Action` values with valid values for the intended use of your KMS key.

```
{
    "Sid": "AllowUseOfKeyInAccount111122223333",
    "Effect": "Allow",
    "Action": [
      "kms:Encrypt",
      "kms:Decrypt",
      "kms:ReEncrypt*",
      "kms:GenerateDataKey*",
      "kms:DescribeKey"
    ],
    "Resource": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
}
```

Note the following details about this policy:

- Unlike key policies, IAM policy statements do not contain the `Principal` element. In IAM policies, the principal is the identity to which the policy is attached.
- The `Resource` element in the IAM policy identifies the KMS key that the principal can use. To specify a KMS key, add its key ARN to the `Resource` element.
- You can specify more than one KMS key in the `Resource` element. But if you don't specify particular KMS keys in the `Resource` element, you might inadvertently give access to more KMS keys than you intend.

- To allow the external user to use the KMS key with AWS services that integrate with AWS KMS, you might need to add permissions to the key policy or the IAM policy. For details, see Allowing use of external KMS keys with AWS services.

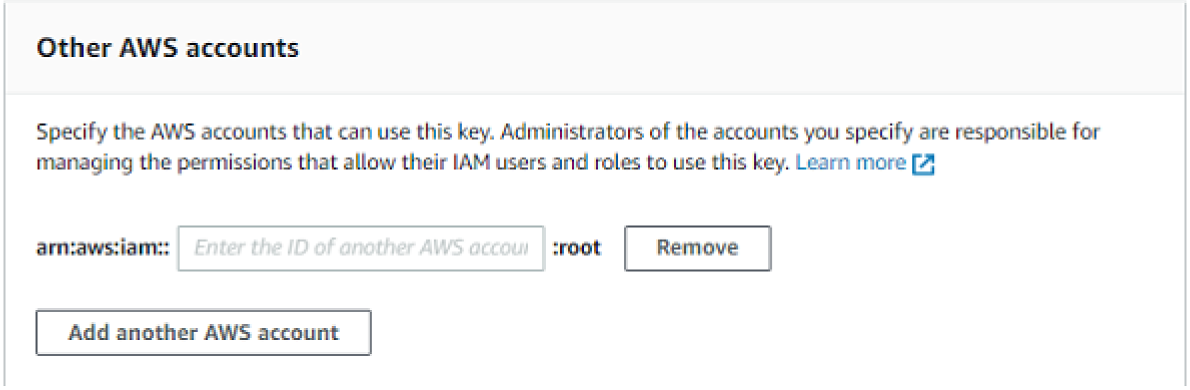
For more information about working with IAM policies, see IAM policies.

Creating KMS keys that other accounts can use

When you use the CreateKey operation to create a KMS key, you can use its `Policy` parameter to specify a key policy that gives an external account, or external users and roles, permission to use the KMS key. You must also add IAM policies in the external account that delegate these permissions to the account's users and roles, even when users and roles are specified in the key policy. You can change the key policy at any time by using the PutKeyPolicy operation.

When you create a KMS key in the AWS Management Console, you also create its key policy. When you select identities in the **Key Administrators** and **Key Users** sections, AWS KMS adds policy statements for those identities to the KMS key's key policy.

The **Key Users** section also lets you add external accounts as key users.



Other AWS accounts

Specify the AWS accounts that can use this key. Administrators of the accounts you specify are responsible for managing the permissions that allow their IAM users and roles to use this key. Learn more

arn:aws:iam:: Enter the ID of another AWS accou :root

When you enter the account ID of an external account, AWS KMS adds two statements to the key policy. This action only affects the key policy. Users and roles in the external account cannot use the KMS key until you attach IAM policies to give them some or all of these permissions.

The first key policy statement gives the external account permission to use the KMS key in cryptographic operations.

Note

The following examples represent a sample key policy for sharing a KMS key with another account. Replace the example `Sid`, `Principal`, and `Action` values with valid values for the intended use of your KMS key.

```
{
    "Sid": "Allow use of the key",
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::444455556666:root"
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
    ],
    "Resource": "*"
}
```

The second key policy statement allows the external account to create, view, and revoke grants on the KMS key, but only when the request comes from an AWS service that is integrated with AWS KMS. These permissions allow other AWS services that encrypt user data to use the KMS key.

These permissions are designed for KMS keys that encrypt user data in AWS services, such as Amazon WorkMail. These services typically use grants to get the permissions they need to use the KMS key on the user's behalf. For details, see Allowing use of external KMS keys with AWS services.

```
{
    "Sid": "Allow attachment of persistent resources",
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::444455556666:root"
    },
    "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
    ],
    "Resource": "*",
    "Condition": {
        "Bool": {
            "kms:GrantIsForAWSResource": "true"
        }
    }
}
```

If these permissions don't meet your needs, you can edit them in the console policy view or by using the PutKeyPolicy operation. You can specify particular external users and role instead of giving permission to the external account. You can change the actions that the policy specifies. And you can use global and AWS KMS policy conditions to refine the permissions.

Allowing use of external KMS keys with AWS services

You can give a user in a different account permission to use your KMS key with a service that is integrated with AWS KMS. For example, a user in an external account can use your KMS key to encrypt the objects in an Amazon S3 bucket or to encrypt the secrets they store in AWS Secrets Manager.

The key policy must give the external user or the external user's account permission to use the KMS key. In addition, you need to attach IAM policies to the identity that gives the user permission to use the AWS service. The service might also require that users have additional permissions in the key policy or IAM policy. For a list of permissions that the AWS service requires on a customer managed key, see the Data Protection topic in the Security chapter of the user guide or developer guide for the service.

Using KMS keys in other accounts

If you have permission to use a KMS key in a different AWS account, you can use the KMS key in the AWS Management Console, AWS SDKs, AWS CLI, and AWS Tools for PowerShell.

To identify a KMS key in a different account in a shell command or API request, use the following key identifiers.

- For cryptographic operations, DescribeKey, and GetPublicKey, use the key ARN or alias ARN of the KMS key.
- For CreateGrant, GetKeyRotationStatus, ListGrants, and RevokeGrant, use the key ARN of the KMS key.

If you enter only a key ID or alias name, AWS assumes the KMS key is in your account.

The AWS KMS console does not display KMS keys in other accounts, even if you have permission to use them. Also, the lists of KMS keys displayed in the consoles of other AWS services do not include KMS keys in other accounts.

To specify a KMS key in a different account in the console of an AWS service, you must enter the key ARN or alias ARN of the KMS key. The required key identifier varies with the service, and might differ between the service console and its API operations. For details, see the service documentation.

Using service-linked roles for AWS KMS

AWS Key Management Service uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to AWS KMS. Service-linked roles are defined by AWS KMS and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up AWS KMS easier because you don’t have to manually add the necessary permissions. AWS KMS defines the permissions of its service-linked roles, and unless defined otherwise, only AWS KMS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting the related resources. This protects your AWS KMS resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for AWS KMS custom key stores

AWS KMS uses a service-linked role named **AWSServiceRoleForKeyManagementServiceCustomKeyStores** to support custom key stores. This service-linked role gives AWS KMS permission to view your AWS CloudHSM clusters and create the network infrastructure to support a connection between your custom key store and its AWS CloudHSM cluster. AWS KMS creates this role only when you create a custom key store. You cannot create this service-linked role directly.

The **AWSServiceRoleForKeyManagementServiceCustomKeyStores** service-linked role trusts `cks.kms.amazonaws.com` to assume the role. As a result, only AWS KMS can assume this service-linked role.

The permissions in the role are limited to the actions that AWS KMS performs to connect a custom key store to an AWS CloudHSM cluster. It does not give AWS KMS any additional permissions. For example, AWS KMS does not have permission to create, manage, or delete your AWS CloudHSM clusters, HSMs, or backups.

For more information about the **AWSServiceRoleForKeyManagementServiceCustomKeyStores** role, including a list of permissions and instructions for how to view the role, edit the role description, delete the role, and have AWS KMS recreate it for you, see Authorizing AWS KMS to manage AWS CloudHSM and Amazon EC2 resources.

Service-linked role permissions for AWS KMS multi-Region keys

AWS KMS uses a service-linked role named **AWSServiceRoleForKeyManagementServiceMultiRegionKeys** to support multi-Region keys. This service-linked role gives AWS KMS permission to synchronize any changes to the key material of a multi-Region primary key to its replica keys. AWS KMS creates this role only when you create a multi-Region primary key. You cannot create this service-linked role directly.

The **AWSServiceRoleForKeyManagementServiceMultiRegionKeys** service-linked role trusts `mrk.kms.amazonaws.com` to assume the role. As a result, only AWS KMS can assume this service-linked role. The permissions in the role are limited to the actions that AWS KMS performs to keep the key material in related multi-Region keys synchronized. It does not give AWS KMS any additional permissions.

For more information about the **AWSServiceRoleForKeyManagementServiceMultiRegionKeys** role, including a list of permissions and instructions for how to view the role, edit the role description, delete the role, and have AWS KMS recreate it for you, see Authorizing AWS KMS to synchronize multi-Region keys.

AWS KMS updates to AWS managed policies

View details about updates to AWS managed policies for AWS KMS since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the AWS KMS Document history page.

Change	Description	Date
AWSKeyManagementServiceCustomKeyStoresServiceRolePolicy – Update to existing policy	AWS KMS added the `ec2:DescribeVpcs` , `ec2:DescribeNetworkAcls` , and `ec2:DescribeNetworkInterfaces` permissions to monitor changes in the VPC that contains your AWS CloudHSM cluster so that AWS KMS can provide clear error messages in the case of failures.	November 10, 2023
AWS KMS started tracking changes	AWS KMS started tracking changes for its AWS managed policies.	November 10, 2023

Using hybrid post-quantum TLS with AWS KMS

AWS Key Management Service (AWS KMS) supports a hybrid post-quantum key exchange option for the Transport Layer Security (TLS) network encryption protocol. You can use this TLS option when you connect to AWS KMS API endpoints. We're offering this feature before post-quantum algorithms are standardized so you can begin testing the effect of these key exchange protocols on AWS KMS calls. These optional hybrid post-quantum key exchange features are at least as secure as the TLS encryption we use today and are likely to provide additional long-term security benefits. However, they affect latency and throughput compared to the classic key exchange protocols in use today.

The data that you send to AWS Key Management Service (AWS KMS) is protected in transit by the encryption provided by a Transport Layer Security (TLS) connection. The classic cipher suites that AWS KMS supports for TLS sessions make brute force attacks on the key exchange mechanisms infeasible with current technology. However, if large-scale quantum computing becomes practical in the future, the classic cipher suites used in TLS key exchange mechanisms will be susceptible to these attacks. If you're developing applications that rely on the long-term confidentiality of data passed over a TLS connection, you should consider a plan to migrate to post-quantum cryptography before large-scale quantum computers become available for use. AWS is working to prepare for this future, and we want you to be well-prepared, too.

To protect data encrypted today against potential future attacks, AWS is participating with the cryptographic community in the development of quantum-resistant or *post-quantum* algorithms. We've implemented *hybrid* post-quantum key exchange cipher suites in AWS KMS that combine classic and post-quantum elements to ensure that your TLS connection is at least as strong as it would be with classic cipher suites.

These hybrid cipher suites are available for use on your production workloads in most AWS Regions. However, because the performance characteristics and bandwidth requirements of hybrid cipher suites are different from those of classic key exchange mechanisms, we recommend that you test them on your AWS KMS API calls under different conditions.

Feedback

As always, we welcome your feedback and participation in our open-source repositories. We'd especially like to hear how your infrastructure interacts with this new variant of TLS traffic.

- To provide feedback on this topic, use the **Feedback** link in the upper right corner of this page.
- We're developing these hybrid cipher suites in open source in the s2n-tls repository on GitHub. To provide feedback on the usability of the cipher suites, or share novel test conditions or results, create an issue in the s2n-tls repository.
- We're writing code samples for using hybrid post-quantum TLS with AWS KMS in the aws-kms-pq-tls-example GitHub repository. To ask questions or share ideas about configuring your HTTP client or AWS KMS client to use the hybrid cipher suites, create an issue in the aws-kms-pq-tls-example repository.

Supported AWS Regions

Post-quantum TLS for AWS KMS is available in all AWS Regions that AWS KMS supports except for China (Beijing) and China (Ningxia).

> **Note**
>
> AWS KMS does not support hybrid post-quantum TLS for FIPS endpoints in AWS GovCloud (US).

For a list of AWS KMS endpoints for each AWS Region, see AWS Key Management Service endpoints and quotas in the *Amazon Web Services General Reference*. For information about FIPS endpoints, see FIPS endpoints in the *Amazon Web Services General Reference*.

About hybrid post-quantum key exchange in TLS

AWS KMS supports hybrid post-quantum key exchange cipher suites. You can use the AWS SDK for Java 2.x and AWS Common Runtime on Linux systems to configure an HTTP client that uses these cipher suites. Then, whenever you connect to an AWS KMS endpoint with your HTTP client, the hybrid cipher suites are used.

This HTTP client uses s2n-tls, which is an open source implementation of the TLS protocol. The hybrid cipher suites that s2n-tls uses are implemented only for key exchange, not for direct data encryption. During *key exchange*, the client and server calculate the key they will use to encrypt and decrypt the data on the wire.

The algorithms that s2n-tls uses are a *hybrid* that combines Elliptic Curve Diffie-Hellman (ECDH), a classic key exchange algorithm used today in TLS, with Kyber, a public-key encryption and key-establishment algorithm that the National Institute for Standards and Technology (NIST) has designated as its first standard post-quantum key-agreement algorithm. This hybrid uses each of the algorithms independently to generate a key. Then it combines the two keys cryptographically. With s2n-tls, you can configure an HTTP client to prefer post-quantum TLS, which places ECDH with Kyber first in the preference list. Classic key exchange algorithms are included in the preference list to ensure compatibility, but they are lower in the preference order.

If ongoing research reveals that the Kyber algorithm lacks the anticipated post-quantum strength, the hybrid key is still at least as strong as the single ECDH key currently in use. Until research on post-quantum algorithms is complete, we recommend using hybrid algorithms, rather than using post-quantum algorithms alone.

Using hybrid post-quantum TLS with AWS KMS

You can use hybrid post-quantum TLS for your calls to AWS KMS. When setting up your HTTP client test environment, be aware of the following information:

Encryption in Transit

The hybrid cipher suites in s2n-tls are used only for encryption in transit. They protect your data while it is traveling from your client to the AWS KMS endpoint. AWS KMS does not use these cipher suites to encrypt data under AWS KMS keys.

Instead, when AWS KMS encrypts your data under KMS keys, it uses symmetric cryptography with 256-bit keys and the Advanced Encryption Standard in Galois Counter Mode (AES-GCM) algorithm, which is already quantum resistant. Theoretical future, large-scale quantum computing attacks on ciphertexts created under 256-bit AES-GCM keys reduce the effective security of the key to 128 bits. This security level is sufficient to make brute force attacks on AWS KMS ciphertexts infeasible.

Supported Systems

Use of the hybrid cipher suites in s2n-tls is currently supported only on Linux systems. In addition, these cipher suites are supported only in SDKs that support the AWS Common Runtime, such as the AWS SDK for Java 2.x. For an example, see How to configure hybrid post-quantum TLS.

AWS KMS Endpoints

When using the hybrid cipher suites, use the standard AWS KMS endpoint. The hybrid cipher suites in s2n-tls are not compatible with the FIPS 140-2 validated endpoints for AWS KMS.

When you configure a HTTP client to prefer post-quantum TLS connections with s2n-tls, the post-quantum ciphers are first in the cipher preference list. However, the preference list includes the classic, non-hybrid ciphers lower in the preference order for compatibility. When you configure an HTTP client to prefer post-quantum TLS with an AWS KMS FIPS 140-2 validated endpoint, s2n-tls negotiates a classic, non-hybrid key exchange cipher.

For a list of AWS KMS endpoints for each AWS Region, see AWS Key Management Service endpoints and quotas in the *Amazon Web Services General Reference*. For information about FIPS endpoints, see FIPS endpoints in the *Amazon Web Services General Reference*.

Expected Performance

Our early benchmark testing shows that the hybrid cipher suites in s2n-tls are slower than classic TLS cipher suites. The effect varies based on the network profile, CPU speed, the number of cores, and your call rate. For performance test results, see How to tune TLS for hybrid post-quantum cryptography with Kyber.

How to configure hybrid post-quantum TLS

In this procedure, add a Maven dependency for the AWS Common Runtime HTTP Client. Next, configure an HTTP client that prefers post-quantum TLS. Then, create an AWS KMS client that uses the HTTP client.

To see a complete working examples of configuring and using hybrid post-quantum TLS with AWS KMS, see the aws-kms-pq-tls-example repository.

Note

The AWS Common Runtime HTTP Client, which has been available as a preview, became generally available in February 2023. In that release, the `tlsCipherPreference` class and the `tlsCipherPreference()` method parameter are replaced by the `postQuantumTlsEnabled()` method parameter. If you were using this example during the preview, you need to update your code.

1. Add the AWS Common Runtime client to your Maven dependencies. We recommend using the latest available version.

   For example, this statement adds version `2.20.0` of the AWS Common Runtime client to your Maven dependencies.

   ```
   <dependency>
       <groupId>software.amazon.awssdk</groupId>
       <artifactId>aws-crt-client</artifactId>
       <version>2.20.0</version>
   </dependency>
   ```

2. To enable the hybrid post-quantum cipher suites, add the AWS SDK for Java 2.x to your project and initialize it. Then enable the hybrid post-quantum cipher suites on your HTTP client as shown in the following example.

   This code uses the `postQuantumTlsEnabled()` method parameter to configure an AWS common runtime HTTP client that prefers the recommended hybrid post-quantum cipher suite, ECDH with Kyber. Then it uses the configured HTTP client to build an instance of the AWS KMS asynchronous client, `KmsAsyncClient`. After this code completes, all AWS KMS API requests on the `KmsAsyncClient` instance use hybrid post-quantum TLS.

   ```
   // Configure HTTP client
   SdkAsyncHttpClient awsCrtHttpClient = AwsCrtAsyncHttpClient.builder()
             .postQuantumTlsEnabled(true)
             .build();

   // Create the AWS KMS async client
   ```

```
KmsAsyncClient kmsAsync = KmsAsyncClient.builder()
          .httpClient(awsCrtHttpClient)
          .build();
```

3. Test your AWS KMS calls with hybrid post-quantum TLS.

   When you call AWS KMS API operations on the configured AWS KMS client, your calls are transmitted to the AWS KMS endpoint using hybrid post-quantum TLS. To test your configuration, call an AWS KMS API, such as `ListKeys`.

   ```
   ListKeysReponse keys = kmsAsync.listKeys().get();
   ```

Testing hybrid post-quantum TLS with AWS KMS

Consider running the following tests with hybrid cipher suites on your applications that call AWS KMS.

- Run load tests and benchmarks. The hybrid cipher suites perform differently than traditional key exchange algorithms. You might need to adjust your connection timeouts to allow for the longer handshake times. If you’re running inside an AWS Lambda function, extend the execution timeout setting.
- Try connecting from different locations. Depending on the network path your request takes, you might discover that intermediate hosts, proxies, or firewalls with deep packet inspection (DPI) block the request. This might result from using the new cipher suites in the ClientHello part of the TLS handshake, or from the larger key exchange messages. If you have trouble resolving these issues, work with your security team or IT administrators to update the relevant configuration and unblock the new TLS cipher suites.

Learn more about post-quantum TLS in AWS KMS

For more information about using hybrid post-quantum TLS in AWS KMS, see the following resources.

- To learn about post-quantum cryptography at AWS, including links to blog posts and research papers, see Post-Quantum Cryptography.
- For information about s2n-tls, see Introducing s2n-tls, a New Open Source TLS Implementation and Using s2n-tls.

- For information about the AWS Common Runtime HTTP Client, see Configuring the AWS CRT-based HTTP client in the *AWS SDK for Java 2.x Developer Guide*.
- For information about the post-quantum cryptography project at the National Institute for Standards and Technology (NIST), see Post-Quantum Cryptography.
- For information about NIST post-quantum cryptography standardization, see Post-Quantum Cryptography Standardization.

Determining access to AWS KMS keys

To determine the full extent of who or what currently has access to an AWS KMS key, you must examine the key policy of the KMS key, all grants that apply to the KMS key, and potentially all AWS Identity and Access Management (IAM) policies. You might do this to determine the scope of potential usage of a KMS key, or to help you meet compliance or auditing requirements. The following topics can help you generate a complete list of the AWS principals (identities) that currently have access to a KMS key.

Topics

Examining the key policy

Key policies are the primary way to control access to KMS keys. Every KMS key has exactly one key policy.

When a key policy consists of or includes the default key policy, the key policy allows IAM administrators in the account to use IAM policies to control access to the KMS key. Also, if the key policy gives another AWS account permission to use the KMS key, the IAM administrators in the external account can use IAM policies to delegate those permissions. To determine the complete list of principals that can access the KMS key, examine the IAM policies.

To view the key policy of an AWS KMS customer managed key or AWS managed key in your account, use the AWS Management Console or the GetKeyPolicy operation in the AWS KMS API.

To view the key policy, you must have `kms:GetKeyPolicy` permissions for the KMS key. For instructions for viewing the key policy for a KMS key, see the section called “Viewing a key policy”.

Examine the key policy document and take note of all principals specified in each policy statement's `Principal` element. In a policy statement with an `Allow` effect, the IAM users, IAM roles, and AWS accounts in the `Principal` element have access to this KMS key.

> **Note**
>
> Do not set the Principal to an asterisk (*) in any key policy statement that allows permissions unless you use conditions to limit the key policy. An asterisk gives every identity in every AWS account permission to use the KMS key, unless another policy statement explicitly denies it. Users in other AWS accounts can use your KMS key whenever they have corresponding permissions in their own account.

The following examples use the policy statements found in the default key policy to demonstrate how to do this.

Example Policy statement 1

```
{
  "Sid": "Enable IAM User Permissions",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::111122223333:root"},
  "Action": "kms:*",
  "Resource": "*"
}
```

In policy statement 1, `arn:aws:iam::111122223333:root` is an AWS account principal that refers to the AWS account 111122223333. (It is not the account root user.) By default, a policy statement like this one is included in the key policy document when you create a new KMS key with the AWS Management Console, or create a new KMS key programmatically but do not provide a key policy.

A key policy document with a statement that allows access to the AWS account enables IAM policies in the account to allow access to the KMS key. This means that users and roles in the account might have access to the KMS key even if they are not explicitly listed as principals in the key policy document. Take care to examine all IAM policies in all AWS accounts listed as principals to determine whether they allow access to this KMS key.

Example Policy statement 2

```
{
  "Sid": "Allow access for Key Administrators",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::111122223333:role/KMSKeyAdmins"},
  "Action": [
    "kms:Describe*",
    "kms:Put*",
    "kms:Create*",
    "kms:Update*",
    "kms:Enable*",
    "kms:Revoke*",
    "kms:List*",
    "kms:Disable*",
    "kms:Get*",
    "kms:Delete*",
    "kms:ScheduleKeyDeletion",
    "kms:CancelKeyDeletion"
  ],
  "Resource": "*"
}
```

In policy statement 2, `arn:aws:iam::111122223333:role/KMSKeyAdmins` refers to the IAM role named KMSKeyAdmins in AWS account 111122223333. Users who are authorized to assume this role are allowed to perform the actions listed in the policy statement, which are the administrative actions for managing a KMS key.

Example Policy statement 3

```
{
  "Sid": "Allow use of the key",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::111122223333:role/EncryptionApp"},
  "Action": [
    "kms:DescribeKey",
    "kms:GenerateDataKey*",
    "kms:Encrypt",
    "kms:ReEncrypt*",
    "kms:Decrypt"
  ],
  "Resource": "*"
}
```

In policy statement 3, `arn:aws:iam::111122223333:role/EncryptionApp` refers to the IAM role named EncryptionApp in AWS account 111122223333. Principals who are authorized to assume this role are allowed to perform the actions listed in the policy statement, which include the cryptographic operations for a symmetric encryption KMS key.

Example Policy statement 4

```
{
  "Sid": "Allow attachment of persistent resources",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::111122223333:role/EncryptionApp"},
  "Action": [
    "kms:ListGrants",
    "kms:CreateGrant",
    "kms:RevokeGrant"
  ],
  "Resource": "*",
  "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
}
```

In policy statement 4, `arn:aws:iam::111122223333:role/EncryptionApp` refers to the IAM role named EncryptionApp in AWS account 111122223333. Principals who are authorized assume this role are allowed to perform the actions listed in the policy statement. These actions, when combined with the actions allowed in **Example policy statement 3**, are those necessary to delegate use of the KMS key to most AWS services that integrate with AWS KMS, specifically the services that use grants. The kms:GrantIsForAWSResource value in the `Condition` element ensures that the delegation is allowed only when the delegate is an AWS service that integrates with AWS KMS and uses grants for authorization.

To learn all the different ways you can specify a principal in a key policy document, see Specifying a Principal in the *IAM User Guide*.

To learn more about AWS KMS key policies, see Key policies in AWS KMS.

Examining IAM policies

In addition to the key policy and grants, you can also use IAM policies to allow access to a KMS key. For more information about how IAM policies and key policies work together, see Troubleshooting key access.

To determine which principals currently have access to a KMS key through IAM policies, you can use the browser-based IAM Policy Simulator tool, or you can make requests to the IAM API.

Ways to examine IAM policies

- Examining IAM policies with the IAM policy simulator
- Examining IAM policies with the IAM API

Examining IAM policies with the IAM policy simulator

The IAM Policy Simulator can help you learn which principals have access to a KMS key through an IAM policy.

To use the IAM policy simulator to determine access to a KMS key

1. Sign in to the AWS Management Console and then open the IAM Policy Simulator at https://policysim.aws.amazon.com/.
2. In the **Users, Groups, and Roles** pane, choose the user, group, or role whose policies you want to simulate.
3. (Optional) Clear the check box next to any policies that you want to omit from the simulation. To simulate all policies, leave all policies selected.
4. In the **Policy Simulator** pane, do the following:

    a. For **Select service**, choose **Key Management Service**.

    b. To simulate specific AWS KMS actions, for **Select actions**, choose the actions to simulate. To simulate all AWS KMS actions, choose **Select All**.
5. (Optional) The Policy Simulator simulates access to all KMS keys by default. To simulate access to a specific KMS key, choose **Simulation Settings**and then type the Amazon Resource Name (ARN) of the KMS key to simulate.
6. Choose **Run Simulation**.

You can view the results of the simulation in the **Results** section. Repeat steps 2 through 6 for every user, group, and role in the AWS account.

Examining IAM policies with the IAM API

You can use the IAM API to examine IAM policies programmatically. The following steps provide a general overview of how to do this:

1. For each AWS account listed as a principal in the key policy (that is, each AWS account principal specified in this format: `"Principal": {"AWS": "arn:aws:iam::111122223333:root"}`), use the ListUsers and ListRoles operations in the IAM API to get all users and roles in the account.
2. For each user and role in the list, use the SimulatePrincipalPolicy operation in the IAM API, passing in the following parameters:
   - For `PolicySourceArn`, specify the Amazon Resource Name (ARN) of a user or role from your list. You can specify only one `PolicySourceArn` for each `SimulatePrincipalPolicy` request, so you must call this operation multiple times, once for each user and role in your list.
   - For the `ActionNames` list, specify every AWS KMS API action to simulate. To simulate all AWS KMS API actions, use `kms:*`. To test individual AWS KMS API actions, precede each API action with "`kms:`", for example "`kms:ListKeys`". For a complete list of AWS KMS API actions, see Actions in the *AWS Key Management Service API Reference*.
   - (Optional) To determine whether the users or roles have access to specific KMS keys, use the `ResourceArns` parameter to specify a list of the Amazon Resource Names (ARNs) of the KMS keys. To determine whether the users or roles have access to any KMS key, omit the `ResourceArns` parameter.

IAM responds to each `SimulatePrincipalPolicy` request with an evaluation decision: `allowed`, `explicitDeny`, or `implicitDeny`. For each response that contains an evaluation decision of `allowed`, the response includes the name of the specific AWS KMS API operation that is allowed. It also includes the ARN of the KMS key that was used in the evaluation, if any.

Examining grants

Grants are advanced mechanisms for specifying permissions that you or an AWS service integrated with AWS KMS can use to specify how and when a KMS key can be used. Grants are attached to a KMS key, and each grant contains the principal who receives permission to use the KMS key and a list of operations that are allowed. Grants are an alternative to the key policy, and are useful for specific use cases. For more information, see Grants in AWS KMS.

To get a list of grants for a KMS key, use the AWS KMS ListGrants operation. You can examine the grants for a KMS key to determine who or what currently has access to use the KMS key via those grants. For example, the following is a JSON representation of a grant that was obtained from the list-grants command in the AWS CLI.

```
{"Grants": [{
  "Operations": ["Decrypt"],
  "KeyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
  "Name": "0d8aa621-43ef-4657-b29c-3752c41dc132",
  "RetiringPrincipal": "arn:aws:iam::123456789012:root",
  "GranteePrincipal": "arn:aws:sts::111122223333:assumed-role/aws:ec2-infrastructure/
i-5d476fab",
  "GrantId": "dc716f53c93acacf291b1540de3e5a232b76256c83b2ecb22cdefa26576a2d3e",
  "IssuingAccount": "arn:aws:iam::111122223333:root",
  "CreationDate": 1.444151834E9,
  "Constraints": {"EncryptionContextSubset": {"aws:ebs:id": "vol-5cccfb4e"}}
}]}
```

To find out who or what has access to use the KMS key, look for the `"GranteePrincipal"` element. In the preceding example, the grantee principal is an assumed role user that is associated with the EC2 instance i-5d476fab. The EC2 infrastructure uses this role to attach the encrypted EBS volume vol-5cccfb4e to the instance. In this case, the EC2 infrastructure role has permission to use the KMS key because you previously created an encrypted EBS volume that is protected by this KMS key. You then attached the volume to an EC2 instance.

The following is another example of a JSON representation of a grant that was obtained from the list-grants command in the AWS CLI. In the following example, the grantee principal is another AWS account.

```
{"Grants": [{
  "Operations": ["Encrypt"],
  "KeyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
  "Name": "",
  "GranteePrincipal": "arn:aws:iam::444455556666:root",
  "GrantId": "f271e8328717f8bde5d03f4981f06a6b3fc18bcae2da12ac38bd9186e7925d11",
  "IssuingAccount": "arn:aws:iam::111122223333:root",
  "CreationDate": 1.444151269E9
}]}
```

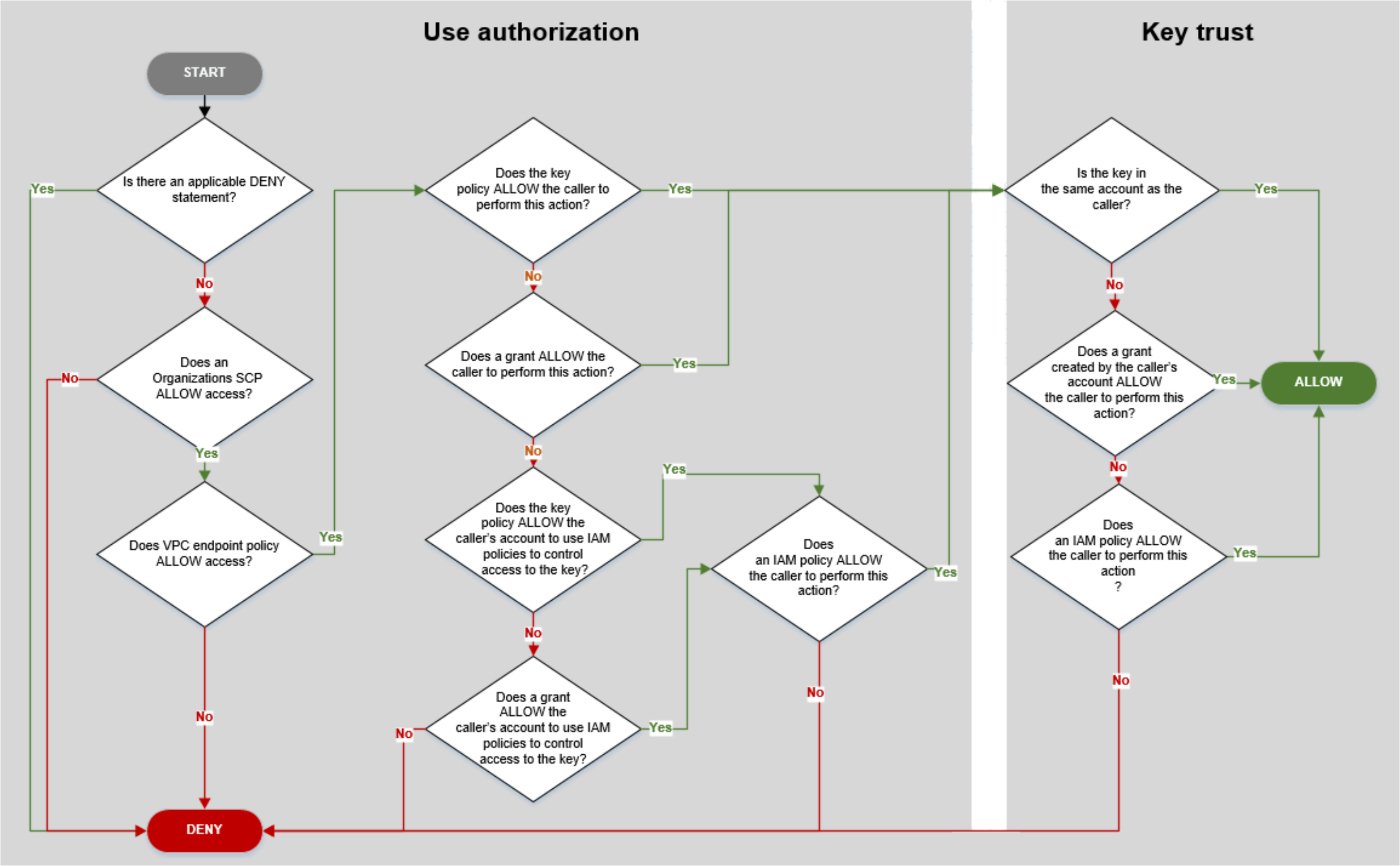
Troubleshooting key access

When authorizing access to a KMS key, AWS KMS evaluates the following:

- The key policy that is attached to the KMS key. The key policy is always defined in the AWS account and Region that owns the KMS key.

- All IAM policies that are attached to the user or role making the request. IAM policies that govern a principal's use of a KMS key are always defined in the principal's AWS account.
- All grants that apply to the KMS key.
- Other types of policies that might apply to the request to use the KMS key, such as AWS Organizations service control policies and VPC endpoint policies. These policies are optional and allow all actions by default, but you can use them to restrict permissions otherwise given to principals.

AWS KMS evaluates these policy mechanisms together to determine whether access to the KMS key is allowed or denied. To do this, AWS KMS uses a process similar to the one depicted in the following flowchart. The following flowchart provides a visual representation of the policy evaluation process.


Use authorization
Key trust
START
Is there an applicable DENY statement?
Does an Organizations SCP ALLOW access?
Does VPC endpoint policy ALLOW access?
Does the key policy ALLOW the caller to perform this action?
Does a grant ALLOW the caller to perform this action?
Does the key policy ALLOW the caller's account to use IAM policies to control access to the key?
Does a grant ALLOW the caller's account to use IAM policies to control access to the key?
Does an IAM policy ALLOW the caller to perform this action?
Is the key in the same account as the caller?
Does a grant created by the caller's account ALLOW the caller to perform this action?
Does an IAM policy ALLOW the caller to perform this action ?
Yes
No
ALLOW
DENY

This flowchart is divided into two parts. The parts appear to be sequential, but they are typically evaluated at the same time.

- *Use authorization* determines whether you are permitted to use a KMS key based on its key policy, IAM policies, grants, and other applicable policies.
- *Key trust* determines whether you should trust a KMS key that you are permitted to use. In general, you trust the resources in your AWS account. But, you can also feel confident about using KMS keys in a different AWS account if a grant or IAM policy in your account allows you to use the KMS key.

You can use this flowchart to discover why a caller was allowed or denied permission to use a KMS key. You can also use it to evaluate your policies and grants. For example, the flowchart shows that a caller can be denied access by an explicit `DENY` statement, or by the absence of an explicit `ALLOW` statement, in the key policy, IAM policy, or grant.

The flowchart can explain some common permission scenarios.

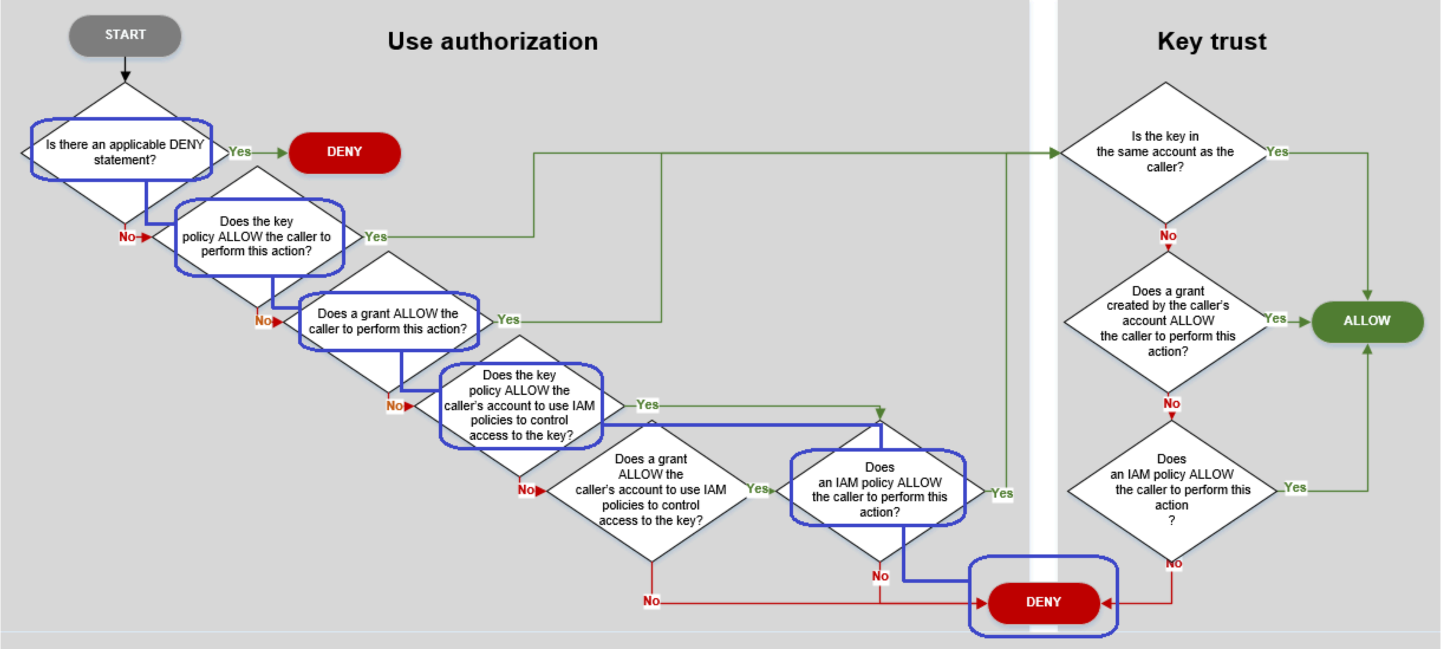
Permission Examples

- Example 1: User is denied access to a KMS key in their AWS account
- Example 2: User assumes role with permission to use a KMS key in a different AWS account

Example 1: User is denied access to a KMS key in their AWS account

Alice is an IAM user in the 111122223333 AWS account. She was denied access to a KMS key in same AWS account. Why can't Alice use the KMS key?

In this case, Alice is denied access to the KMS key because there is no key policy, IAM policy, or grant that gives her the required permissions. The key policy of the KMS key allows the AWS account to use IAM policies to control access to the KMS key, but no IAM policy gives Alice permission to use the KMS key.


START
Use authorization
Key trust
Is there an applicable DENY statement?
Yes
DENY
No
Does the key policy ALLOW the caller to perform this action?
Yes
No
Does a grant ALLOW the caller to perform this action?
Yes
No
Does the key policy ALLOW the caller's account to use IAM policies to control access to the key?
Yes
No
Does a grant ALLOW the caller's account to use IAM policies to control access to the key?
Yes
No
Does an IAM policy ALLOW the caller to perform this action?
Yes
No
Is the key in the same account as the caller?
Yes
No
Does a grant created by the caller's account ALLOW the caller to perform this action?
Yes
No
ALLOW
Does an IAM policy ALLOW the caller to perform this action ?
Yes
No
DENY

Consider the relevant policies for this example.

- The KMS key that Alice wants to use has the default key policy. This policy allows the AWS account that owns the KMS key to use IAM policies to control access to the KMS key. This key policy satisfies the *Does the key policy ALLOW the callers account to use IAM policies to control access to the key?* condition in the flowchart.

```
{
  "Version" : "2012-10-17",
  "Id" : "key-test-1",
  "Statement" : [ {
    "Sid" : "Delegate to IAM policies",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::111122223333:root"
    },
    "Action" : "kms:*",
    "Resource" : "*"
  } ]
}
```

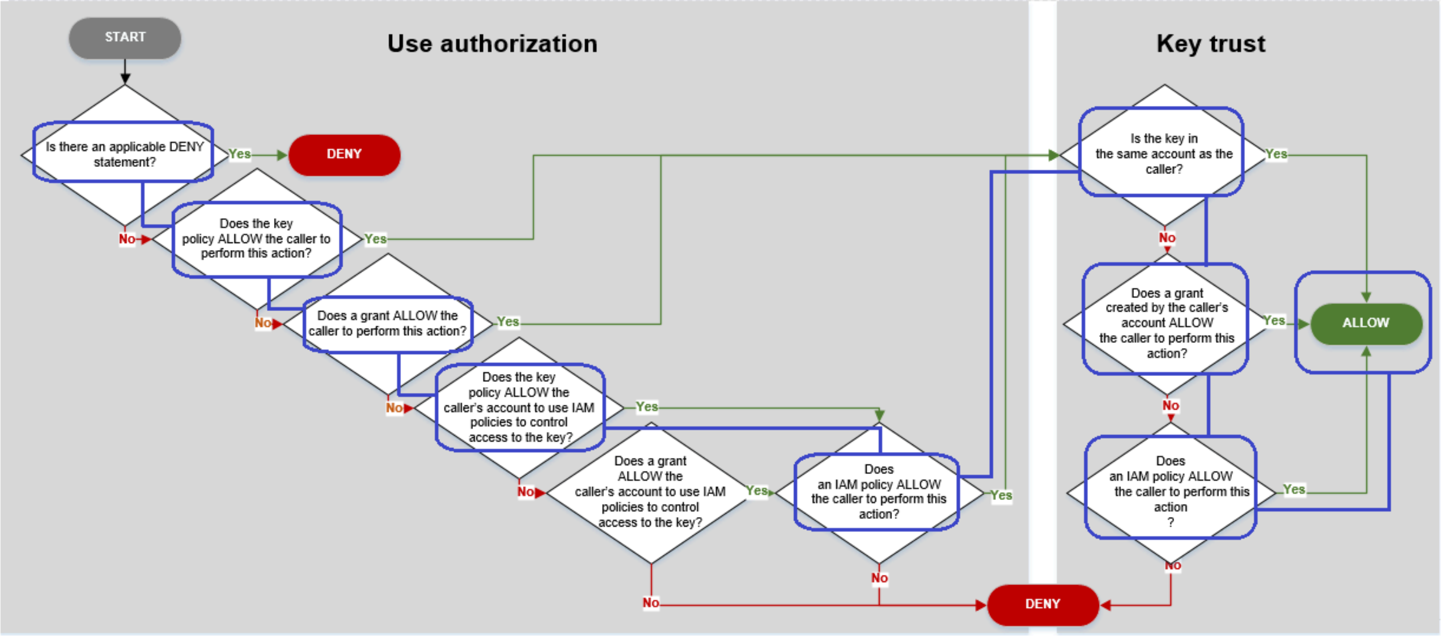
- However, no key policy, IAM policy, or grant gives Alice permission to use the KMS key. Therefore, Alice is denied permission to use the KMS key.

Example 2: User assumes role with permission to use a KMS key in a different AWS account

Bob is a user in account 1 (111122223333). He is allowed to use a KMS key in account 2 (444455556666) in cryptographic operations. How is this possible?

> **Tip**
>
> When evaluating cross-account permissions, remember that the key policy is specified in the KMS key's account. The IAM policy is specified in the caller's account, even when the caller is in a different account. For details about providing cross-account access to KMS keys, see Allowing users in other accounts to use a KMS key.

- The key policy for the KMS key in account 2 allows account 2 to use IAM policies to control access to the KMS key.
- The key policy for the KMS key in account 2 allows account 1 to use the KMS key in cryptographic operations. However, account 1 must use IAM policies to give its principals access to the KMS key.
- An IAM policy in account 1 allows the `Engineering` role to use the KMS key in account 2 for cryptographic operations.
- Bob, a user in account 1, has permission to assume the `Engineering` role.
- Bob can trust this KMS key, because even though it is not in his account, an IAM policy in his account gives him explicit permission to use this KMS key.


START
Use authorization
Key trust
Is there an applicable DENY statement?
Yes
DENY
No
Does the key policy ALLOW the caller to perform this action?
Yes
No
Does a grant ALLOW the caller to perform this action?
Yes
No
Does the key policy ALLOW the caller's account to use IAM policies to control access to the key?
Yes
No
Does a grant ALLOW the caller's account to use IAM policies to control access to the key?
Yes
No
Does an IAM policy ALLOW the caller to perform this action?
Yes
No
DENY
Is the key in the same account as the caller?
Yes
No
Does a grant created by the caller's account ALLOW the caller to perform this action?
Yes
No
ALLOW
Does an IAM policy ALLOW the caller to perform this action ?
Yes
No

Consider the policies that let Bob, a user in account 1, use the KMS key in account 2.

- The key policy for the KMS key allows account 2 (444455556666, the account that owns the KMS key) to use IAM policies to control access to the KMS key. This key policy also allows account 1 (111122223333) to use the KMS key in cryptographic operations (specified in the `Action` element of the policy statement). However, no one in account 1 can use the KMS key in account 2 until account 1 defines IAM policies that give the principals access to the KMS key.

  In the flowchart, this key policy in account 2 satisfies the *Does the key policy ALLOW the caller's account to use IAM policies to control access to the key?* condition.

```
{
    "Id": "key-policy-acct-2",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Permission to use IAM policies",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::444455556666:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow account 1 to use this KMS key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncryptFrom",
                "kms:ReEncryptTo",
                "kms:GenerateDataKey",
                "kms:GenerateDataKeyWithoutPlaintext",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        }
    ]
}
```

- An IAM policy in the caller's AWS account (account 1, 111122223333) gives the principal permission to perform cryptographic operations using the KMS key in account 2 (444455556666). The `Action` element delegates to the principal the same permissions that the key policy in account 2 gave to account 1. To give these permission to the `Engineering` role in account 1, this inline policy is embedded in the `Engineering` role.

  Cross-account IAM policies like this one are effective only when the key policy for the KMS key in account 2 gives account 1 permission to use the KMS key. Also, account 1 can only give its principals permission to perform the actions that the key policy gave to the account.

  In the flowchart, this satisfies the *Does an IAM policy allow the caller to perform this action?* condition.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncryptFrom",
                "kms:ReEncryptTo",
                "kms:GenerateDataKey",
                "kms:GenerateDataKeyWithoutPlaintext",
                "kms:DescribeKey"
            ],
            "Resource": [
                "arn:aws:kms:us-
west-2:444455556666:key/1234abcd-12ab-34cd-56ef-1234567890ab"
            ]
        }
    ]
}
```

- The last required element is the definition of the `Engineering` role in account 1. The `AssumeRolePolicyDocument` in the role allows Bob to assume the `Engineering` role.

```
{
    "Role": {
        "Arn": "arn:aws:iam::111122223333:role/Engineering",
        "CreateDate": "2019-05-16T00:09:25Z",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": {
                "Principal": {
                    "AWS": "arn:aws:iam::111122223333:user/bob"
                },
                "Effect": "Allow",
                "Action": "sts:AssumeRole"
            }
        },
        "Path": "/",
        "RoleName": "Engineering",
        "RoleId": "AROA4KJY2TU23Y7NK62MV"
    }
}
```

AWS KMS permissions

This table is designed to help you understand AWS KMS permissions so you can control access to your AWS KMS resources. Definitions of the column headings appear below the table.

You can also learn about AWS KMS permissions in the Actions, resources, and condition keys for AWS Key Management Service topic of the *Service Authorization Reference*. However, that topic doesn't list all of the condition keys that you can use to refine each permission.

Note

You might have to scroll horizontally or vertically to see all of the data in the table.

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
CancelKeyDeletion `kms:CancelKeyDeletion`	Key policy	No	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService
ConnectCustomKeyStore `kms:ConnectCustomKeyStore`	IAM policy	No	*	kms:CallerAccount

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
CreateAlias `kms:CreateAlias` To use this operation, the caller needs `kms:CreateAlias` permission on two resources: • The alias (in an IAM policy) • The KMS key (in a key policy) For details, see Controlling access to aliases.	IAM policy (for the alias)	No	Alias	None (when controlling access to the alias)
	Key policy (for the KMS key)	No	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService
CreateCustomKeyStore `kms:CreateCustomKeyStore`	IAM policy	No	*	kms:CallerAccount

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
CreateGrant `kms:CreateGrant`	Key policy	Yes	KMS key	*Encryption context conditions:* kms:EncryptionContext:*context-key* kms:EncryptionContextKeys *Grant conditions:* kms:GrantConstraintType kms:GranteePrincipal kms:GrantIsForAWSResource kms:GrantOperations kms:RetiringPrincipal *Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
				aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService
CreateKey `kms:CreateKey`	IAM policy	No	*	kms:BypassPolicyLockoutSafetyCheck kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ViaService aws:RequestTag/*tag-key* (AWS global condition key) aws:ResourceTag/*tag-key* (AWS global condition key) aws:TagKeys (AWS global condition key)

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
Decrypt `kms:Decrypt`	Key policy	Yes	KMS key	*Conditions for cryptographic operations* kms:EncryptionAlgorithm kms:RequestAlias *Encryption context conditions:* kms:EncryptionContext:*context-key* kms:EncryptionContextKeys *Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key)

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
				kms:ViaService
DeleteAlias `kms:DeleteAlias` To use this operation, the caller needs `kms:DeleteAlias` permission on two resources: - The alias (in an IAM policy) - The KMS key (in a key policy) For details, see Controlling access to aliases.	IAM policy (for the alias)	No	Alias	None (when controlling access to the alias)
	Key policy (for the KMS key)	No	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService
DeleteCustomKeyStore `kms:DeleteCustomKeyStore`	IAM policy	No	*	kms:CallerAccount

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
DeleteImportedKeyMaterial `kms:DeleteImportedKeyMaterial`	Key policy	No	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
DedriveSharedSecret `kms:DeriveSharedSecret`	Key policy	Yes	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService *Conditions for cryptographic operations:* kms:KeyAgreementAlgorithm
DescribeCustomKeyStores `kms:DescribeCustomKeyStores`	IAM policy	No	*	kms:CallerAccount

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
DescribeKey `kms:DescribeKey`	Key policy	Yes	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService *Other conditions:* kms:RequestAlias

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
DisableKey `kms:DisableKey`	Key policy	No	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
DisableKeyRotation `kms:DisableKeyRotation`	Key policy	No	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService
DisconnectCustomKeyStore `kms:DisconnectCustomKeyStore`	IAM policy	No	*	kms:CallerAccount

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
EnableKey `kms:EnableKey`	Key policy	No	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
EnableKeyRotation `kms:EnableKeyRotation`	Key policy	No	KMS key (symmetric only)	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService *Automatic key rotation conditions:* kms:RotationPeriodInDays

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
Encrypt `kms:Encrypt`	Key policy	Yes	KMS key	*Conditions for cryptographic operations* kms:EncryptionAlgorithm kms:RequestAlias *Encryption context conditions:* kms:EncryptionContext:*context-key* kms:EncryptionContextKeys *Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key)

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
				kms:ViaService

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
GenerateDataKey `kms:GenerateDataKey`	Key policy	Yes	KMS key (symmetric only)	*Conditions for cryptographic operations* kms:EncryptionAlgorithm kms:RequestAlias *Encryption context conditions:* kms:EncryptionContext:*context-key* kms:EncryptionContextKeys *Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key)

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
				kms:ViaService

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
GenerateDataKeyPair `kms:GenerateDataKeyPair`	Key policy	Yes	KMS key (symmetric only) Generates an asymmetric data key pair that is protected by a symmetric encryption KMS key.	*Conditions for data key pairs:* kms:DataKeyPairSpec *Conditions for cryptographic operations* kms:EncryptionAlgorithm kms:RequestAlias *Encryption context conditions:* kms:EncryptionContext:*context-key* kms:EncryptionContextKeys *Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
				kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
GenerateDataKeyPairWithoutPlaintext `kms:GenerateDataKeyPairWithoutPlaintext`	Key policy	Yes	KMS key (symmetric only) Generates an asymmetric data key pair that is protected by a symmetric encryption KMS key.	*Conditions for data key pairs:* kms:DataKeyPairSpec *Conditions for cryptographic operations* kms:EncryptionAlgorithm kms:RequestAlias *Encryption context conditions:* kms:EncryptionContext:*context-key* kms:EncryptionContextKeys *Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType

Actions and permissions	Policy type	Cross-acc ount use	Resources (for IAM policies)	AWS KMS condition keys
				kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
GenerateDataKeyWithoutPlaintext `kms:GenerateDataKeyWithoutPlaintext`	Key policy	Yes	KMS key (symmetric only)	*Conditions for cryptographic operations* kms:EncryptionAlgorithm kms:RequestAlias *Encryption context conditions:* kms:EncryptionContext:*context-key* kms:EncryptionContextKeys *Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key)

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
				kms:ViaService
GenerateMac `kms:GenerateMac`	Key policy	Yes	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService *Conditions for cryptographic operations:* kms:MacAlgorithm kms:RequestAlias
GenerateRandom `kms:GenerateRandom`	IAM policy	N/A	*	None

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
GetKeyPolicy `kms:GetKeyPolicy`	Key policy	No	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
GetKeyRotationStatus `kms:GetKeyRotationStatus`	Key policy	Yes	KMS key (symmetric only)	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
GetParametersForImport `kms:GetParametersForImport`	Key policy	No	KMS key	kms:WrappingAlgorithm kms:WrappingKeySpec *Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
GetPublicKey `kms:GetPublicKey`	Key policy	Yes	KMS key (asymmetric only)	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService *Other conditions:* kms:RequestAlias

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
ImportKeyMaterial `kms:ImportKeyMaterial`	Key policy	No	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService *Other conditions:* kms:ExpirationModel kms:ValidTo
ListAliases `kms:ListAliases`	IAM policy	No	*	None

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
ListGrants `kms:ListGrants`	Key policy	Yes	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService *Other conditions:* kms:GrantIsForAWSResource

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
ListKeyPolicies `kms:ListKeyPolicies`	Key policy	No	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
ListKeyRotations `kms:ListKeyRotations`	Key policy	No	KMS key (symmetric only)	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService
ListKeys `kms:ListKeys`	IAM policy	No	*	None

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
ListResourceTags `kms:ListResourceTags`	Key policy	No	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService
ListRetirableGrants `kms:ListRetirableGrants`	IAM policy	The specified principal must be in the local account, but the operation returns grants in all accounts.	*	None

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
PutKeyPolicy `kms:PutKeyPolicy`	Key policy	No	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService *Other conditions:* kms:BypassPolicyLockoutSafetyCheck

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys

ReEncrypt
`kms:ReEncryptFrom`
`kms:ReEncryptTo`
To use this operation, the caller needs permission on two KMS keys:
- `kms:ReEncryptFrom` on the KMS key used to decrypt
- `kms:ReEncryptTo` on the KMS key used to encrypt
	Key policy	Yes	KMS key	*Conditions for cryptographic operations*
				kms:EncryptionAlgorithm
				kms:RequestAlias
				Encryption context conditions:
				kms:EncryptionContext:*context-key*
				kms:EncryptionContextKeys
				Conditions for KMS key operations:
				kms:CallerAccount
				kms:KeySpec
				kms:KeyUsage
				kms:KeyOrigin
				kms:MultiRegion
				kms:MultiRegionKeyType
				kms:ResourceAliases
				aws:ResourceTag/*tag-key* (AWS global condition key)

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
				kms:ViaService *Other conditions:* kms:ReEncryptOnSameKey
ReplicateKey `kms:ReplicateKey` To use this operation, the caller needs the following permissions: - `kms:ReplicateKey` on the multi-Region primary key - `kms:CreateKey` in an IAM policy in the replica Region	Key policy	No	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService *Other conditions:* kms:ReplicaRegion

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
RetireGrant `kms:RetireGrant` Permission to retire a grant is determined primarily by the grant. A policy alone cannot allow access to this operation . For more information, see Retiring and revoking grants.	IAM policy (This permission is not effective in a key policy.)	Yes	KMS key	*Encryption context conditions:* kms:EncryptionContext:*context-key* kms:EncryptionContextKeys *Grant conditions:* kms:GrantConstraintType *Conditions for KMS key operations:* kms:CallerAccount kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
RevokeGrant `kms:RevokeGrant`	Key policy	Yes	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService *Other conditions:* kms:GrantIsForAWSResource

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
RotateKeyOnDemand `kms:RotateKeyOnDemand`	Key policy	No	KMS key (symmetric only)	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
ScheduleKeyDeletion `kms:ScheduleKeyDeletion`	Key policy	No	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
Sign `kms:Sign`	Key policy	Yes	KMS key (asymmetric only)	*Conditions for signing and verification:* kms:MessageType kms:RequestAlias kms:SigningAlgorithm *Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
TagResource `kms:TagResource`	Key policy	No	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService *Conditions for tagging:* aws:RequestTag/*tag-key* (AWS global condition key) aws:TagKeys (AWS global condition key)

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
UntagResource `kms:UntagResource`	Key policy	No	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService *Conditions for tagging:* aws:RequestTag/*tag-key* (AWS global condition key) aws:TagKeys (AWS global condition key)

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
UpdateAlias `kms:UpdateAlias` To use this operation, the caller needs `kms:UpdateAlias` permission on three resources: - The alias - The currently associated KMS key - The newly associated KMS key For details, see Controlling access to aliases.	IAM policy (for the alias)	No	Alias	None (when controlling access to the alias)
	Key policy (for the KMS keys)	No	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService
UpdateCustomKeyStore `kms:UpdateCustomKeyStore`	IAM policy	No	*	kms:CallerAccount

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
UpdateKeyDescription `kms:UpdateKeyDescription`	Key policy	No	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
UpdatePrimaryRegion `kms:UpdatePrimaryRegion` To use this operation, the caller needs `kms:UpdatePrimaryRegion` permission on both the multi-Region primary key that will become a replica key and the multi-Region replica key that will become the primary key.	Key policy	No	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService *Other conditions* kms:PrimaryRegion

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
Verify `kms:Verify`	Key policy	Yes	KMS key (asymmetric only)	*Conditions for signing and verification:* kms:MessageType kms:RequestAlias kms:SigningAlgorithm *Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService

Actions and permissions	Policy type	Cross-account use	Resources (for IAM policies)	AWS KMS condition keys
VerifyMac `kms:VerifyMac`	Key policy	Yes	KMS key	*Conditions for KMS key operations:* kms:CallerAccount kms:KeySpec kms:KeyUsage kms:KeyOrigin kms:MultiRegion kms:MultiRegionKeyType kms:ResourceAliases aws:ResourceTag/*tag-key* (AWS global condition key) kms:ViaService *Conditions for cryptographic operations:* kms:MacAlgorithm kms:RequestAlias

Column descriptions

The columns in this table provide the following information:

- **Actions and permissions** lists each AWS KMS API operation and the permission that allows the operation. You specify the operation in `Action` element of a policy statement.
- **Policy type** indicates whether the permission can be used in a key policy or IAM policy.

Key policy means that you can specify the permission in the key policy. When the key policy contains the policy statement that enables IAM policies, you can specify the permission in an IAM policy.

IAM policy means that you can specify the permission only in an IAM policy.

- **Cross-account use** shows the operations that authorized users can perform on resources in a different AWS account.

  A value of *Yes* means that principals can perform the operation on resources in a different AWS account.

  A value of *No* means that principals can perform the operation only on resources in their own AWS account.

  If you give a principal in a different account a permission that can't be used on a cross-account resource, the permission is not effective. For example, if you give a principal in a different account kms:TagResource permission to a KMS key in your account, their attempts to tag the KMS key in your account will fail.

- **Resources** lists the AWS KMS resources to which the permissions apply. AWS KMS supports two resource types: a KMS key and an alias. In a key policy, the value of the `Resource` element is always `*`, which indicates the KMS key to which the key policy is attached.

  Use the following values to represent an AWS KMS resource in an IAM policy.

  KMS key

  When the resource is a KMS key, use its key ARN. For help, see the section called "Finding the key ID and key ARN".

  ```
  arn:AWS_partition_name:kms:AWS_Region:AWS_account_ID:key/key_ID
  ```

  For example:

  arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

  Alias

  When the resource is an alias, use its alias ARN. For help, see the section called "Finding the alias name and alias ARN".

  ```
  arn:AWS_partition_name:kms:AWS_region:AWS_account_ID:alias/alias_name
  ```

For example:

arn:aws:kms:us-west-2:111122223333:alias/ExampleAlias

*** (asterisk)**

When the permission doesn't apply to a particular resource (KMS key or alias), use an asterisk (*).

In an IAM policy for an AWS KMS permission, an asterisk in the `Resource` element indicates all AWS KMS resources (KMS keys and aliases). You can also use an asterisk in the `Resource` element when the AWS KMS permission doesn't apply to any particular KMS keys or aliases. For example, when allowing or denying `kms:CreateKey` or `kms:ListKeys` permission, you can set the `Resource` element to `*` or to an account-specific variation, such as `arn:`*`AWS_partition_name`*`:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:*`.

- **AWS KMS condition keys** lists the AWS KMS condition keys that you can use to control access to the operation. You specify conditions in a policy's `Condition` element. For more information, see AWS KMS condition keys. This column also includes AWS global condition keys that are supported by AWS KMS, but not by all AWS services.

Testing your permissions

To use AWS KMS, you must have credentials that AWS can use to authenticate your API requests. The credentials must include the permission to access KMS keys and aliases. The permissions are determined by key policies, IAM policies, grants, and cross-account access controls. In addition to controlling access to KMS keys, you can control access to your CloudHSM, and to your custom key stores.

You can specify the `DryRun` API parameter to verify that you have the necessary permissions to use AWS KMS keys. You can also use `DryRun` to verify that the request parameters in a AWS KMS API call are correctly specified.

Topics

What is the DryRun parameter?

`DryRun` is an optional API parameter that you specify to verify that AWS KMS API calls will succeed. Use `DryRun` to test your API call, before actually making the call to AWS KMS. You can verify the following.

- That you have the necessary permissions to use AWS KMS keys.
- That you have specified the parameters in the call correctly.

AWS KMS supports using the `DryRun` parameter in certain API actions:

- CreateGrant
- Decrypt
- DeriveSharedSecret
- Encrypt
- GenerateDataKey
- GenerateDataKeyPair
- GenerateDataKeyPairWithoutPlaintext
- GenerateDataKeyWithoutPlaintext
- GenerateMac
- ReEncrypt
- RetireGrant
- RevokeGrant
- Sign
- Verify
- VerifyMac

Using the `DryRun` parameter will incur charges and will be billed as a standard API request. For more information about AWS KMS pricing, see AWS Key Management Service Pricing.

All API requests using the `DryRun` parameter apply to the request quota of the API and can result in a throttling exception if you exceed an API request quota. For example, calling Decrypt with `DryRun` or without `DryRun` counts against the same cryptographic operations quota. See Throttling AWS KMS requests to learn more.

Every call to an AWS KMS API operation is captured as an event and recorded in an AWS CloudTrail log. The output of any operations that specify the DryRun parameter appear in your CloudTrail log. For more information, see Logging AWS KMS API calls with AWS CloudTrail.

Specifying DryRun with the API

To use DryRun, specify the –dry-run parameter in AWS CLI commands and AWS KMS API calls that support the parameter. When you do, AWS KMS will verify whether your call will succeed. AWS KMS calls that use DryRun will always fail and return a message with information about reason why the call failed. The message can include the following exceptions:

- DryRunOperationException - The request would succeed if DryRun wasn't specified.
- ValidationException - The request failed from specifying an incorrect API parameter.
- AccessDeniedException - You do not have permissions to perform the specified API action on the KMS resource.

For example, the following command uses the CreateGrant operation and creates a grant that allows users who are authorized to assume the keyUserRole role to call the Decrypt operation on a specified symmetric KMS key. The DryRun parameter is specified.

```
$  aws kms create-grant \
    --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
    --grantee-principal arn:aws:iam::111122223333:role/keyUserRole \
    --operations Decrypt \
    --dry-run
```

Special-purpose keys

AWS Key Management Service (AWS KMS) supports several different types of keys for different uses.

When you create an AWS KMS key, by default, you get a symmetric encryption KMS key. In AWS KMS, a *symmetric encryption KMS key* represents a 256-bit AES-GCM key that is used for encryption and decryption, except in China Regions, where it represents a symmetric a 128-bit symmetric key that uses SM4 encryption. Symmetric key material never leaves AWS KMS unencrypted. Unless your task explicitly requires asymmetric encryption or HMAC keys, symmetric encryption KMS keys, which never leave AWS KMS unencrypted, are a good choice. Also, AWS services that are integrated with AWS KMS use only symmetric encryption KMS keys to encrypt your data. These services do not support encryption with asymmetric KMS keys.

You can use a symmetric encryption KMS key in AWS KMS to encrypt, decrypt, and re-encrypt data, generate data keys and data key pairs, and generate random byte strings. You can import your own key material into a symmetric encryption KMS key and create symmetric encryption KMS keys in custom key stores. For a table comparing the operations that you can perform on symmetric and asymmetric KMS keys, see Key type reference.

AWS KMS also supports the following special-purpose KMS key types:

- Asymmetric RSA keys for encrypting and decrypting or signing and verification
- Asymmetric ECC keys for signing and verification or deriving shared secrets
- Asymmetric SM2 keys (China Regions only) for encrypting and decrypting, signing and verification, or deriving shared secrets
- HMAC keys to generate and verify hash-based message authentication codes
- Multi-Region keys (symmetric and asymmetric) that work like copies of the same key in different AWS Regions
- Keys with imported key material that you provide
- Keys in a custom key store that is backed by a AWS CloudHSM cluster or an external key manager outside of AWS.

Choosing a KMS key type

AWS KMS supports several types of KMS keys: symmetric encryption keys, symmetric HMAC keys, asymmetric encryption keys, and asymmetric signing keys.

KMS keys differ because they contain different cryptographic key material.

- Symmetric encryption KMS key: Represents a single 256-bit AES-GCM encryption key, except in China Regions, where it represents a 128-bit SM4 encryption key. Symmetric key material never leaves AWS KMS unencrypted. To use your symmetric encryption KMS key, you must call AWS KMS.

  Symmetric encryption keys, which are the default KMS keys, are ideal for most uses. If you need a KMS key to protect your data in an AWS service, use a symmetric encryption key unless you are instructed to use another type of key.
- Asymmetric KMS key: Represents a mathematically related public key and private key pair that you can use for encryption and decryption, signing and verification, or deriving shared secrets (you must choose one key usage type). The private key never leaves AWS KMS unencrypted. You can use the public key within AWS KMS by calling the AWS KMS API operations, or download the public key and use it outside of AWS KMS.
- HMAC KMS key (symmetric): Represents a symmetric key of varying length that is used to generate and verify hash-based message authentication codes. The key material in an HMAC KMS key never leaves AWS KMS unencrypted. To use your HMAC KMS key, you must call AWS KMS.

The type of KMS key that you create depends largely on how you plan to use the KMS key, your security requirements, and your authorization requirements. When creating your KMS key, remember that the cryptographic configuration of the KMS key, including its key spec and key usage, are established when you create the KMS key and cannot be changed.

Use the following guidance to determine which type of KMS key you need based on your use case.

Encrypt and decrypt data

Use a symmetric KMS key for most use cases that require encrypting and decrypting data. The symmetric encryption algorithm that AWS KMS uses is fast, efficient, and assures the confidentiality and authenticity of data. It supports authenticated encryption with additional authenticated data (AAD), defined as an encryption context. This type of KMS key requires both the sender and recipient of encrypted data to have valid AWS credentials to call AWS KMS.

If your use case requires encryption outside of AWS by users who cannot call AWS KMS, asymmetric KMS keys are a good choice. You can distribute the public key of the asymmetric KMS key to allow these users to encrypt data. And your applications that need to decrypt that data can use the private key of the asymmetric KMS key within AWS KMS.

Sign messages and verify signatures

To sign messages and verify signatures, you must use an asymmetric KMS key. You can use a KMS key with a key spec that represents an RSA key pair, an elliptic curve (ECC) key pair, or an SM2 key pair (China Regions only). The key spec you choose is determined by the signing algorithm that you want to use. The ECDSA signing algorithms that ECC key pairs support are recommended over the RSA signing algorithms. However, you might need to use a particular key spec and signing algorithm to support users who verify signatures outside of AWS.

Encrypt with asymmetric key pairs

To encrypt data with an asymmetric key pair, you must use an asymmetric KMS key with an RSA key spec or an SM2 key spec (China Regions only). To encrypt data in AWS KMS with the public key of a KMS key pair, use the Encrypt operation. You can also download the public key and share it with the parties that need to encrypt data outside of AWS KMS.

When you download the public key of an asymmetric KMS key, you can use it outside of AWS KMS. But it is no longer subject to the security controls that protect the KMS key in AWS KMS. For example, you cannot use AWS KMS key policies or grants to control use of the public key. Nor can you control whether the key is used only for encryption and decryption using the encryption algorithms that AWS KMS supports. For more details, see Special Considerations for Downloading Public Keys.

To decrypt data that was encrypted with the public key outside of AWS KMS, call the Decrypt operation. The `Decrypt` operation fails if the data was encrypted under a public key from a KMS key with a key usage of `SIGN_VERIFY`. It will also fail if it was encrypted by using an algorithm that AWS KMS does not support for the key spec you selected. For more information on key specs and supported algorithms, see Asymmetric key specs.

To avoid these errors, anyone using a public key outside of AWS KMS must store the key configuration. The AWS KMS console and the GetPublicKey response provide the information that you must include when you share the public key.

Derive shared secrets

To derive shared secrets, use a KMS key with NIST-recommended elliptic curve or SM2 (China Regions only) key material. AWS KMS uses the Elliptic Curve Cryptography Cofactor Diffie-Hellman Primitive (ECDH) to establish a key agreement between two peers by deriving a shared secret from their elliptic curve public-private key pairs. You can use the raw shared secret that the DeriveSharedSecret operation returns to derive a symmetric key that can encrypt and decrypt data that is sent between two parties, or generate and verify HMACs. AWS KMS recommends that you follow NIST recommendations for key derivation when using the raw shared secret to derive a symmetric key.

Generate and verify HMAC codes

To generate and verify hash-based message authentication codes, use an HMAC KMS key. When you create an HMAC key in AWS KMS, AWS KMS creates and protects your key material and ensures that you use the correct MAC algorithms for your key. HMAC codes can also be used as pseudo-random numbers, and in certain scenarios for symmetric signing and tokenizing.

HMAC KMS keys are symmetric keys. When creating an HMAC KMS key in the AWS KMS console, choose the `Symmetric` key type.

Use with AWS services

To create a KMS key for use with an AWS service that is integrated with AWS KMS, consult the documentation for the service. AWS services that encrypt your data require a symmetric encryption KMS key.

In addition to these considerations, cryptographic operations on KMS keys with different key specs have different prices and different request quotas. For information about AWS KMS pricing, see AWS Key Management Service Pricing. For information about request quotas, see Request quotas.

Selecting the key usage

The key usage of a KMS key determines whether the KMS key is used for encryption and decryption, or signing and verifying signatures, or generating and verifying HMAC tags. Each KMS key has only one key usage. Using a KMS key for more than one type of operation makes the product of all operations more vulnerable to attack.

Each KMS key can have only one key usage. As shown in the following table, symmetric encryption KMS keys can be used only for encryption and decryption. HMAC KMS keys can be used only for generating and verifying HMAC codes. You need to make a key usage decision for asymmetric KMS keys. Asymmetric KMS keys with RSA key pairs can be used to encrypt or decrypt data or sign and verify messages (but not both). Asymmetric KMS keys with NIST-recommended elliptic curve key pairs can be used to sign and verify messages or derive shared secrets (but not both). Asymmetric KMS keys with ECC_SECG_P256K1 key pairs can be used only to sign and verify messages. Asymmetric KMS keys with SM2 (China Regions only) key pairs can be used to either encrypt and decrypt data, sign and verify messages, or derive shared secrets (you must choose one key usage type).

Valid key usage for KMS key types

KMS key type	Encrypt and decrypt ENCRYPT_DECRYPT	Sign and verify SIGN_VERIFY	Generate and verify MAC GENERATE_VERIFY_MAC	Derive shared secrets KEY_AGREEMENT
Symmetric encryption KMS keys	✓	⊗	⊗	⊗
HMAC KMS keys (symmetric)	⊗	⊗	✓	⊗
Asymmetric KMS keys with RSA key pairs	✓	✓	⊗	⊗
Asymmetric KMS keys with ECC key pairs	⊗	✓	⊗	✓ You must use an asymmetric KMS key with NIST-recommended elliptic curve key material to derive shared secrets.

KMS key type	Encrypt and decrypt ENCRYPT_DECRYPT	Sign and verify SIGN_VERIFY	Generate and verify MAC GENERATE_VERIFY_MAC	Derive shared secrets KEY_AGREEMENT
Asymmetric KMS keys with SM2 key pairs (China Regions only)	✓	✓	⊗	✓

In the AWS KMS console, you first choose the key type (symmetric or asymmetric) and then the key usage. The key type you choose determines which key usage options are displayed. The key usage you choose determines which key specs, if any, are displayed.

To choose a key usage in the AWS KMS console:

- For symmetric encryption KMS keys (default), choose **Encrypt and decrypt**.
- For HMAC KMS keys, choose **Generate and verify MAC**.
- For asymmetric KMS keys with NIST-recommended elliptic curve (ECC) key material, choose **Sign and verify** or **Key agreement**.
- For asymmetric KMS keys with ECC_SECG_P256K1 key material, choose **Sign and verify**.
- For asymmetric KMS keys with RSA key material, choose **Encrypt and decrypt** or **Sign and verify**.
- For asymmetric KMS keys with SM2 key material, choose **Encrypt and decrypt**, **Sign and verify**, or **Key agreement**. The SM2 key spec is available only in China Regions.

To allow principals to create KMS keys only for a particular key usage, use the kms:KeyUsage condition key. You can also use the `kms:KeyUsage` condition key to allow principals to call API operations for a KMS key based on its key usage. For example, you can allow permission to disable a KMS key only if its key usage is SIGN_VERIFY.

Selecting the key spec

When you create an asymmetric KMS key or an HMAC KMS key, you select its key spec. The *key spec*, which is a property of every AWS KMS key, represents the cryptographic configuration of your KMS key. You choose the key spec when you create the KMS key, and you cannot change it. If you've selected the wrong key spec, delete the KMS key, and create a new one.

> **Note**
>
> The key spec for a KMS key was known as a "customer master key spec." The `CustomerMasterKeySpec` parameter of the CreateKey operation is deprecated. Instead, use the `KeySpec` parameter. The response of the `CreateKey` and DescribeKey operations includes a `KeySpec` and `CustomerMasterKeySpec` member with the same value.

The key spec determines whether the KMS key is symmetric or asymmetric, the type of key material in the KMS key, and the encryption algorithms, signing algorithms, or message authentication code (MAC) algorithms that AWS KMS supports for the KMS key. The key spec that you choose is typically determined by your use case and regulatory requirements. However, cryptographic operations on KMS keys with different key specs are priced differently and are subject to different quotas. For pricing details, see AWS Key Management Service Pricing. For information about request quotas, see Request quotas.

To determine the key specs that principals in your account are permitted to use for KMS keys, use the kms:KeySpec condition key.

AWS KMS supports the following key specs for KMS keys:

Symmetric encryption key spec (default)

- SYMMETRIC_DEFAULT

HMAC key specs

- HMAC_224
- HMAC_256
- HMAC_384
- HMAC_512

RSA key specs (encryption and decryption -or- signing and verification)

- RSA_2048
- RSA_3072
- RSA_4096

Elliptic curve key specs

- Asymmetric NIST-recommended elliptic curve key pairs (signing and verification -or- deriving shared secrets)
    - ECC_NIST_P256 (secp256r1)
    - ECC_NIST_P384 (secp384r1)
    - ECC_NIST_P521 (secp521r1)
- Other asymmetric elliptic curve key pairs (signing and verification)
    - ECC_SECG_P256K1 (secp256k1), commonly used for cryptocurrency.

SM2 key spec (encryption and decryption -or- signing and verification -or- deriving shared secrets)

- SM2 (China Regions only)

Asymmetric keys in AWS KMS

AWS KMS supports asymmetric KMS keys that represent a mathematically related RSA, elliptic curve (ECC), or SM2 (China Regions only) public and private key pair. These key pairs are generated in AWS KMS hardware security modules certified under the FIPS 140-2 Cryptographic Module Validation Program, except in the China (Beijing) and China (Ningxia) Regions. The private key never leaves the AWS KMS HSMs unencrypted. You can download the public key for distribution and use outside of AWS. You can create asymmetric KMS keys for encryption and decryption, signing and verification, or deriving shared secrets (you must choose one key usage type).

You can create and manage the asymmetric KMS keys in your AWS account, including setting the key policies, IAM policies, and grants that control access to the keys, enabling and disabling the KMS keys, creating tags and aliases, and deleting the KMS keys. You can audit all operations that use or manage your asymmetric KMS keys within AWS in AWS CloudTrail logs.

AWS KMS also provides asymmetric data key pairs that are designed to be used for client-side cryptography outside of AWS KMS. The private key in an asymmetric data key pair is protected by a symmetric encryption KMS key in AWS KMS.

This topic explains how asymmetric KMS keys work, how they differ from other KMS keys and how to decide which type of KMS key you need to protect your data. It also explains how asymmetric data key pairs work and how to use them outside of AWS KMS.

Regions

Asymmetric KMS keys and asymmetric data key pairs are supported in all AWS Regions that AWS KMS supports.

Learn more

- To create asymmetric KMS keys, see Creating asymmetric KMS keys. To create symmetric encryption KMS keys, see Creating keys.
- To create multi-Region asymmetric KMS keys, see Creating multi-Region keys.
- To find out whether a KMS key is symmetric or asymmetric, see Identifying asymmetric KMS keys.
- For a table that compares the AWS KMS API operations that apply to each type of KMS key, see the section called "Key type reference".
- To control access to the key specs, key usage, encryption algorithms, and signing algorithms that principals in your account can use for KMS keys and data keys, see the section called "AWS KMS condition keys".
- To learn about the request quotas that apply to different types of KMS keys, see the section called "Request quotas".
- To learn how to sign messages and verify signatures with asymmetric KMS keys, see Digital signing with the new asymmetric keys feature of AWS KMS in the *AWS Security Blog*.

Topics

Asymmetric KMS keys

You can create an asymmetric KMS key in AWS KMS. An *asymmetric KMS key* represents a mathematically related public key and private key pair. You can give the public key to anyone, even if they're not trusted, but the private key must be kept secret.

In an asymmetric KMS key, the private key is created in AWS KMS and never leaves AWS KMS unencrypted. To use the private key, you must call AWS KMS. You can use the public key within

AWS KMS by calling the AWS KMS API operations. Or, you can download the public key and use it outside of AWS KMS.

If your use case requires encryption outside of AWS by users who cannot call AWS KMS, asymmetric KMS keys are a good choice. However, if you are creating a KMS key to encrypt the data that you store or manage in an AWS service, use a symmetric encryption KMS key. AWS services that are integrated with AWS KMS use only symmetric encryption KMS keys to encrypt your data. These services do not support encryption with asymmetric KMS keys.

AWS KMS supports three types of asymmetric KMS keys.

- **RSA KMS keys**: A KMS key with an RSA key pair for encryption and decryption or signing and verification (but not both). AWS KMS supports several key lengths for different security requirements.
- **Elliptic Curve (ECC) KMS keys**: A KMS key with an elliptic curve key pair for signing and verification or deriving shared secrets (but not both). AWS KMS supports several commonly-used curves.
- **SM2 KMS keys (China Regions only)**: A KMS key with an SM2 key pair for encryption and decryption, signing and verification, or deriving shared secrets (you must choose one key usage type).

For help choosing your asymmetric key configuration, see Choosing a KMS key type. For technical details about the encryption and signing algorithms that AWS KMS supports for RSA KMS keys, see RSA key specs. For technical details about the signing algorithms that AWS KMS supports for ECC KMS keys, see Elliptic curve key specs. For technical details about the encryption and signing algorithms that AWS KMS supports for SM2 KMS keys (China Regions only), see SM2 key spec.

For a table comparing the operations that you can perform on symmetric and asymmetric KMS keys, see Comparing Symmetric and Asymmetric KMS keys. For help determining whether a KMS key is symmetric or asymmetric, see Identifying asymmetric KMS keys.

Regions

Asymmetric KMS keys and asymmetric data key pairs are supported in all AWS Regions that AWS KMS supports.

Creating asymmetric KMS keys

You can create asymmetric KMS keys in the AWS KMS console, by using the CreateKey API, or by using an AWS CloudFormation template. An asymmetric KMS key represents a public and private key pair that can be used for encryption, signing, or deriving shared secrets. The private key remains within AWS KMS. To download the public key for use outside of AWS KMS, see Downloading public keys.

When creating a KMS key to encrypt data that you store or manage in an AWS service, use a symmetric encryption KMS key. AWS services that integrate with AWS KMS do not support asymmetric KMS keys. For help deciding whether to create a symmetric or asymmetric KMS key, see Choosing a KMS key type.

For information about the permissions required to create KMS keys, see Permissions for creating KMS keys.

Topics

Creating asymmetric KMS keys (console)

You can use the AWS Management Console to create asymmetric AWS KMS keys (KMS keys). Each asymmetric KMS key represents a public and private key pair.

> ⚠ **Important**
>
> Do not include confidential or sensitive information in the alias, description, or tags. These fields may appear in plain text in CloudTrail logs and other output.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Choose **Create key**.
5. To create an asymmetric KMS key, in **Key type,** choose **Asymmetric**.

For information about how to create an symmetric encryption KMS key in the AWS KMS console, see Creating symmetric encryption KMS keys (console).

6. To create an asymmetric KMS key for public key encryption, in **Key usage**, choose **Encrypt and decrypt**. Or, to create an asymmetric KMS key for signing messages and verifying signatures, in **Key usage**, choose **Sign and verify**. To create an asymmetric KMS key for deriving shared secrets, in **Key usage**, choose **Key agreement**.

   For help choosing a key usage value, see Selecting the key usage.

7. Select a specification (**Key spec**) for your asymmetric KMS key.

   Often the key spec that you select is determined by regulatory, security, or business requirements. It might also be influenced by the size of messages that you need to encrypt or sign. In general, longer encryption keys are more resistant to brute-force attacks.

   For help choosing a key spec, see Selecting the key spec.

8. Choose **Next**.

9. Type an alias for the KMS key. The alias name cannot begin with **`aws/`**. The **`aws/`** prefix is reserved by Amazon Web Services to represent AWS managed keys in your account.

   An *alias* is a friendly name that you can use to identify the KMS key in the console and in some AWS KMS APIs. We recommend that you choose an alias that indicates the type of data you plan to protect or the application you plan to use with the KMS key.

   Aliases are required when you create a KMS key in the AWS Management Console. You cannot specify an alias when you use the CreateKey operation, but you can use the console or the CreateAlias operation to create an alias for an existing KMS key. For details, see Using aliases.

10. (Optional) Type a description for the KMS key.

    Enter a description that explains the type of data you plan to protect or the application you plan to use with the KMS key.

    You can add a description now or update it any time unless the key state is `Pending Deletion` or `Pending Replica Deletion`. To add, change, or delete the description of an existing customer managed key, edit the description in the AWS Management Console or use the UpdateKeyDescription operation.

11. (Optional) Type a tag key and an optional tag value. To add more than one tag to the KMS key, choose **Add tag**.

When you add tags to your AWS resources, AWS generates a cost allocation report with usage and costs aggregated by tags. Tags can also be used to control access to a KMS key. For information about tagging KMS keys, see Tagging keys and ABAC for AWS KMS.

12. Choose **Next**.

13. Select the IAM users and roles that can administer the KMS key.

> **Note**
>
> This key policy gives the AWS account full control of this KMS key. It allows account administrators to use IAM policies to give other principals permission to manage the KMS key. For details, see the section called “Default key policy”.
> IAM best practices discourage the use of IAM users with long-term credentials. Whenever possible, use IAM roles, which provide temporary credentials. For details, see Security best practices in IAM in the *IAM User Guide*.

14. (Optional) To prevent the selected IAM users and roles from deleting this KMS key, in the **Key deletion** section at the bottom of the page, clear the **Allow key administrators to delete this key** check box.

15. Choose **Next**.

16. Select the IAM users and roles that can use the KMS key for cryptographic operations.

> **Note**
>
> This key policy gives the AWS account full control of this KMS key. It allows account administrators to use IAM policies to give other principals permission to use the KMS key in cryptographic operations. For details, see the section called “Default key policy”.
> IAM best practices discourage the use of IAM users with long-term credentials. Whenever possible, use IAM roles, which provide temporary credentials. For details, see Security best practices in IAM in the *IAM User Guide*.

17. (Optional) You can allow other AWS accounts to use this KMS key for cryptographic operations. To do so, in the **Other AWS accounts** section at the bottom of the page, choose **Add another AWS account** and enter the AWS account identification number of an external account. To add multiple external accounts, repeat this step.

> **Note**
>
> To allow principals in the external accounts to use the KMS key, administrators of the external account must create IAM policies that provide these permissions. For more information, see Allowing users in other accounts to use a KMS key.

18. Choose **Next**.
19. Review the key settings that you chose. You can still go back and change all settings.
20. Choose **Finish** to create the KMS key.

Creating asymmetric KMS keys (AWS KMS API)

You can use the CreateKey operation to create an asymmetric AWS KMS key. These examples use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

When you create an asymmetric KMS key, you must specify the `KeySpec` parameter, which determines the type of keys you create. Also, you must specify a `KeyUsage` value of ENCRYPT_DECRYPT, SIGN_VERIFY, or KEY_AGREEMENT. You cannot change these properties after the KMS key is created.

The `CreateKey` operation doesn't let you specify an alias, but you can use the CreateAlias operation to create an alias for your new KMS key.

> **Important**
>
> Do not include confidential or sensitive information in the `Description` or `Tags` fields. These fields may appear in plain text in CloudTrail logs and other output.

The following example uses the `CreateKey` operation to create an asymmetric KMS key of 4096-bit RSA keys designed for public key encryption.

```
$ aws kms create-key --key-spec RSA_4096 --key-usage ENCRYPT_DECRYPT
{
    "KeyMetadata": {
        "KeyState": "Enabled",
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "KeyManager": "CUSTOMER",
        "Description": "",
        "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "CreationDate": 1569973196.214,
        "MultiRegion": false,
        "KeySpec": "RSA_4096",
        "CustomerMasterKeySpec": "RSA_4096",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "EncryptionAlgorithms": [
            "RSAES_OAEP_SHA_1",
            "RSAES_OAEP_SHA_256"
        ],
        "AWSAccountId": "111122223333",
        "Origin": "AWS_KMS",
        "Enabled": true
    }
}
```

The following example command creates an asymmetric KMS key that represents a pair of ECDSA keys used for signing and verification. You cannot create an elliptic curve key pair for encryption and decryption.

```
$ aws kms create-key --key-spec ECC_NIST_P521 --key-usage SIGN_VERIFY
{
    "KeyMetadata": {
        "KeyState": "Enabled",
        "KeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
        "CreationDate": 1570824817.837,
        "Origin": "AWS_KMS",
        "SigningAlgorithms": [
            "ECDSA_SHA_512"
        ],
        "Arn": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321",
        "AWSAccountId": "111122223333",
        "KeySpec": "ECC_NIST_P521",
        "CustomerMasterKeySpec": "ECC_NIST_P521",
        "KeyManager": "CUSTOMER",
        "Description": "",
        "Enabled": true,
        "MultiRegion": false,
        "KeyUsage": "SIGN_VERIFY"
    }
}
```

The following example command creates an asymmetric KMS key that represents a pair of ECDH keys used for deriving shared secrets. You cannot create an elliptic curve key pair for encryption and decryption.

```
$ aws kms create-key --key-spec ECC_NIST_P256 --key-usage KEY_AGREEMENT
{
    "KeyMetadata": {
        "AWSAccountId": "111122223333",
        "KeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
        "Arn": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321",
        "CreationDate": "2023-12-27T19:10:15.063000+00:00",
        "Enabled": true,
        "Description": "",
        "KeyUsage": "KEY_AGREEMENT",
        "KeyState": "Enabled",
        "Origin": "AWS_KMS",
        "KeyManager": "CUSTOMER",
        "CustomerMasterKeySpec": "ECC_NIST_P256",
        "KeySpec": "ECC_NIST_P256",
        "KeyAgreementAlgorithms": [
            "ECDH"
        ],
        "MultiRegion": false
    }
}
```

Downloading public keys

You can view, copy, and download the public key from an asymmetric KMS key pair by using the AWS Management Console or the AWS KMS API. You must have `kms:GetPublicKey` permission on the asymmetric KMS key.

Each asymmetric KMS key pair consists of a private key that never leaves AWS KMS unencrypted and a public key that you can download and share.

You might share a public key to let others encrypt data outside of AWS KMS that you can decrypt only with your private key. Or, to allow others to verify a digital signature outside of AWS KMS that you have generated with your private key. Or, to share your public key with a peer to derive a shared secret.

When you use the public key in your asymmetric KMS key within AWS KMS, you benefit from the authentication, authorization, and logging that are part of every AWS KMS operation. You also reduce of risk of encrypting data that cannot be decrypted. These features are not effective outside of AWS KMS. For details, see Special considerations for downloading public keys.

Tip

Looking for data keys or SSH keys? This topic explains how to manage asymmetric keys in AWS Key Management Service, where the private key is not exportable. For exportable data key pairs where the private key is protected by a symmetric encryption KMS key, see GenerateDataKeyPair. For help with downloading the public key associated with an Amazon EC2 instance, see *Retrieving the public key* in the Amazon EC2 User Guide and Amazon EC2 User Guide.

Topics

- Special considerations for downloading public keys
- Downloading a public key (console)
- Downloading a public key (AWS KMS API)

Special considerations for downloading public keys

To protect your KMS keys, AWS KMS provides access controls, authenticated encryption, and detailed logs of every operation. AWS KMS also allows you to prevent the use of KMS keys, temporarily or permanently. Finally, AWS KMS operations are designed to minimize of risk of encrypting data that cannot be decrypted. These features are not available when you use downloaded public keys outside of AWS KMS.

Authorization

Key policies and IAM policies that control access to the KMS key within AWS KMS have no effect on operations performed outside of AWS. Any user who can get the public key can use it outside of AWS KMS even if they don't have permission to encrypt data or verify signatures with the KMS key.

Key usage restrictions

Key usage restrictions are not effective outside of AWS KMS. If you call the Encrypt operation with a KMS key that has a `KeyUsage` of `SIGN_VERIFY`, the AWS KMS operation fails. But if you encrypt data outside of AWS KMS with a public key from a KMS key with a `KeyUsage` of `SIGN_VERIFY` or `KEY_AGREEMENT`, the data cannot be decrypted.

Algorithm restrictions

Restrictions on the encryption and signing algorithms that AWS KMS supports are not effective outside of AWS KMS. If you encrypt data with the public key from a KMS key outside of AWS KMS, and use an encryption algorithm that AWS KMS does not support, the data cannot be decrypted.

Disabling and deleting KMS keys

Actions that you can take to prevent the use of KMS key in a cryptographic operation within AWS KMS do not prevent anyone from using the public key outside of AWS KMS. For example, disabling a KMS key, scheduling deletion of a KMS key, deleting a KMS key, or deleting the key material from a KMS key have no effect on a public key outside of AWS KMS. If you delete an asymmetric KMS key or delete or lose its key material, data that you encrypt with a public key outside of AWS KMS is unrecoverable.

Logging

AWS CloudTrail logs that record every AWS KMS operation, including the request, response, date, time, and authorized user, do not record the use of the public key outside of AWS KMS.

Offline verification with SM2 key pairs (China Regions only)

To verify a signature outside of AWS KMS with an SM2 public key, you must specify the distinguishing ID. By default, AWS KMS uses `1234567812345678` as the distinguishing ID. For more information, see Offline verification with SM2 key pairs (China Regions only).

Downloading a public key (console)

You can use the AWS Management Console to view, copy, and download the public key from an asymmetric KMS key in your AWS account. To download the public key from an asymmetric KMS key in different AWS account, use the AWS KMS API.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.

2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Choose the alias or key ID of an asymmetric KMS key.
5. Choose the **Cryptographic configuration** tab. Record the values of the **Key spec**, **Key usage**, and **Encryption algorithms** or **Signing Algorithms** fields. You'll need to use these values to use the public key outside of AWS KMS. Be sure to share this information when you share the public key.
6. Choose the **Public key** tab.
7. To copy the public key to your clipboard, choose **Copy**. To download the public key to a file, choose **Download**.

Downloading a public key (AWS KMS API)

The GetPublicKey operation returns the public key in an asymmetric KMS key. It also returns critical information that you need to use the public key correctly outside of AWS KMS, including the key usage and encryption algorithms. Be sure to save these values and share them whenever you share the public key.

The examples in this section use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

To specify a KMS key, use its key ID, key ARN, alias name, or alias ARN. When using an alias name, prefix it with **alias/**. To specify a KMS key in a different AWS account, you must use its key ARN or alias ARN.

Before running this command, replace the example alias name with a valid identifier for the KMS key. To run this command, you must have `kms:GetPublicKey` permissions on the KMS key.

```
$ aws kms get-public-key --key-id alias/example_RSA_3072

{
    "KeySpec": "RSA_3072",
    "KeyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "EncryptionAlgorithms": [
        "RSAES_OAEP_SHA_1",
        "RSAES_OAEP_SHA_256"
    ],
    "PublicKey": "MIIBojANBgkqhkiG..."
}
```

Identifying asymmetric KMS keys

To determine if a particular KMS key is an asymmetric KMS key, find the *key type* or key spec. You can use the AWS KMS console or AWS KMS API.

Some of these methods also show you other aspects of the cryptographic configuration of a KMS key, including the key usage and the encryption or signing algorithms that the KMS key supports. You can view the cryptographic configuration of an existing KMS key, but you cannot change it.

For general information about viewing KMS keys, including sorting, filtering, and choosing columns for your console display, see Viewing KMS keys in the console.

Topics

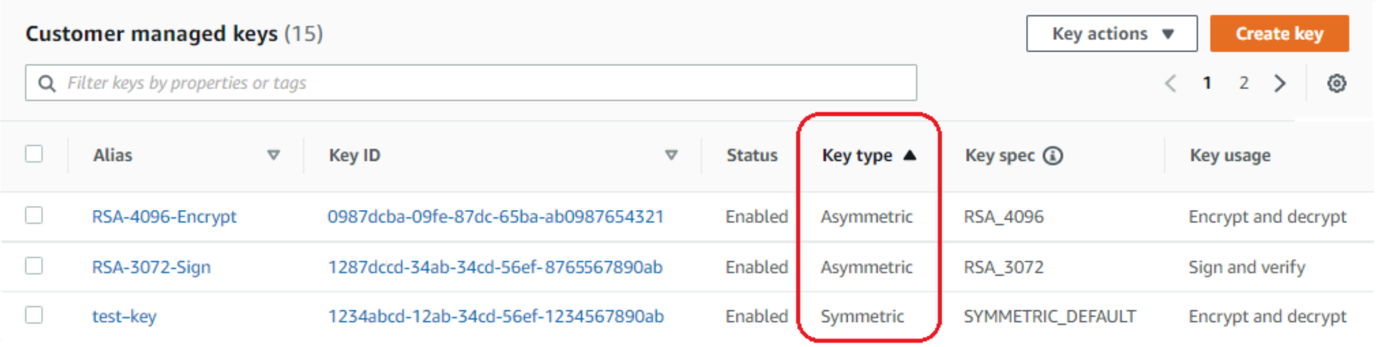
Finding the key type in the KMS key table

In the AWS KMS console, the **Key type** column shows whether each KMS key is symmetric or asymmetric. You can add a **Key type** column to the KMS key table on the **Customer managed keys** or **AWS managed keys** pages in the console.

To identify symmetric and asymmetric KMS keys in your KMS key table, use the following procedure.

1. Open the AWS KMS console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. To view the keys in your account that you create and manage, in the navigation pane choose **Customer managed keys**. To view the keys in your account that AWS creates and manages for you, in the navigation pane, choose **AWS managed keys**.
4. The **Key type** columns shows whether each KMS key is symmetric or asymmetric. You can also sort and filter by the **Key type** value.

If the **Key type** column does not appear in your KMS key table, choose the gear icon in the upper right corner of the page, choose **Key type**, and then choose **Confirm**. You can also add the **Key spec** and **Key usage** columns.



□	Alias	Key ID	Status	Key type ▲	Key spec	Key usage
□	RSA-4096-Encrypt	0987dcba-09fe-87dc-65ba-ab0987654321	Enabled	Asymmetric	RSA_4096	Encrypt and decrypt
□	RSA-3072-Sign	1287dccd-34ab-34cd-56ef-8765567890ab	Enabled	Asymmetric	RSA_3072	Sign and verify
□	test-key	1234abcd-12ab-34cd-56ef-1234567890ab	Enabled	Symmetric	SYMMETRIC_DEFAULT	Encrypt and decrypt

Finding the key type on the details page

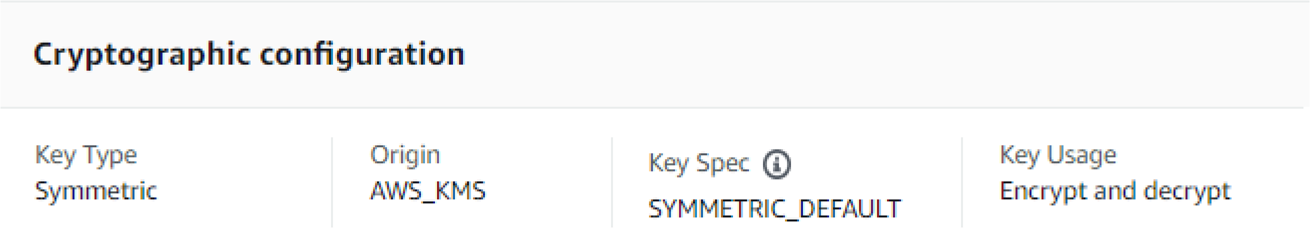
In the AWS KMS console, the details page for each KMS key includes a **Cryptographic Configuration** tab that displays the key type (symmetric or asymmetric) and other cryptographic details about the KMS key.

To identify symmetric and asymmetric KMS keys on the details page for a KMS key, use the following procedure.

1. Open the AWS KMS console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. To view the keys in your account that you create and manage, in the navigation pane choose **Customer managed keys**. To view the keys in your account that AWS creates and manages for you, in the navigation pane, choose **AWS managed keys**.
4. Choose the alias or key ID of a KMS key.
5. Choose the **Cryptographic configuration** tab. The tabs are below the **General configuration** section.

   The **Cryptographic configuration** tab displays the **Key Type**, which indicates whether it is symmetric or asymmetric. It also displays other details about the KMS key, including the **Key Usage**, which tells whether a KMS key can be used for encryption and decryption or signing and verification. For asymmetric KMS keys, it displays the encryption algorithms or signing algorithms that the KMS key supports.

For example, the following is an example **Cryptographic configuration** tab for a symmetric encryption KMS key.



Cryptographic configuration

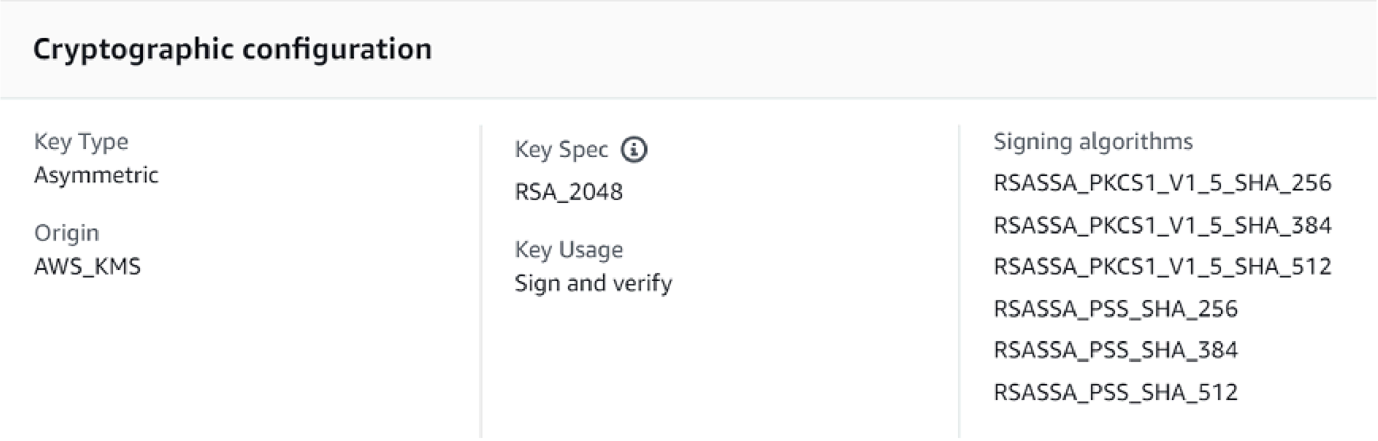
Key Type
Symmetric

Origin
AWS_KMS

Key Spec
SYMMETRIC_DEFAULT

Key Usage
Encrypt and decrypt

The following is an example **Cryptographic configuration** tab for an asymmetric RSA KMS key that's used for signing and verification.



Cryptographic configuration

Key Type
Asymmetric

Origin
AWS_KMS

Key Spec
RSA_2048

Key Usage
Sign and verify

Signing algorithms
RSASSA_PKCS1_V1_5_SHA_256
RSASSA_PKCS1_V1_5_SHA_384
RSASSA_PKCS1_V1_5_SHA_512
RSASSA_PSS_SHA_256
RSASSA_PSS_SHA_384
RSASSA_PSS_SHA_512

Finding the key spec using the AWS KMS API

To determine whether a KMS key is symmetric or asymmetric, use the DescribeKey operation. The `KeySpec` field in the response contains the key spec of the KMS key. For a symmetric encryption KMS key, the value of `KeySpec` is `SYMMETRIC_DEFAULT`. Other values indicate an asymmetric KMS key or an HMAC KMS key.

> **Note**
>
> The `CustomerMasterKeySpec` member is deprecated. Instead, use `KeySpec`. To prevent breaking changes, the `DescribeKey` response includes `KeySpec` and `CustomerMasterKeySpec` members with the same value.

For example, `DescribeKey` returns the following response for a symmetric encryption KMS key. The `KeySpec` value is `SYMMETRIC_DEFAULT`.

```
{
  "KeyMetadata": {
    "AWSAccountId": "111122223333",
    "KeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
    "Arn": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321",
    "CreationDate": 1496966810.831,
    "Enabled": true,
    "Description": "",
    "KeyState": "Enabled",
    "Origin": "AWS_KMS",
    "KeyManager": "CUSTOMER",
    "MultiRegion": false,
    "KeySpec": "SYMMETRIC_DEFAULT",
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "EncryptionAlgorithms": [
        "SYMMETRIC_DEFAULT"
    ]
  }
}
```

The `DescribeKey` response for an asymmetric RSA KMS key used in signing and verification looks similar to this example. The `KeySpec` value is RSA_2048 and the `KeyUsage` is `SIGN_VERIFY`. The `SigningAlgorithms` element lists the valid signing algorithms for the KMS key.

```
{
  "KeyMetadata": {
    "AWSAccountId": "111122223333",
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "CreationDate": 1571767572.317,
    "CustomerMasterKeySpec": "RSA_2048",
    "Enabled": false,
    "Description": "",
    "KeyState": "Disabled",
    "Origin": "AWS_KMS",
    "MultiRegion": false,
    "KeyManager": "CUSTOMER",
    "KeySpec": "RSA_2048",
    "KeyUsage": "SIGN_VERIFY",
    "SigningAlgorithms": [
        "RSASSA_PKCS1_V1_5_SHA_256",
        "RSASSA_PKCS1_V1_5_SHA_384",
        "RSASSA_PKCS1_V1_5_SHA_512",
        "RSASSA_PSS_SHA_256",
        "RSASSA_PSS_SHA_384",
        "RSASSA_PSS_SHA_512"
    ]
  }
}
```

The `DescribeKey` response for an asymmetric ECC KMS key used to derive shared secrets looks similar to this example. The `KeySpec` value is ECC_NIST_P256 and the `KeyUsage` is `SIGN_VERIFY`. The `KeyAgreementAlgorithms` element lists the valid key agreement algorithms for the KMS key.

```
{
    "KeyMetadata": {
        "AWSAccountId": "111122223333",
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "CreationDate": "2023-12-27T19:10:15.063000+00:00",
        "Enabled": true,
        "Description": "",
        "KeyUsage": "KEY_AGREEMENT",
        "KeyState": "Enabled",
        "Origin": "AWS_KMS",
        "KeyManager": "CUSTOMER",
        "CustomerMasterKeySpec": "ECC_NIST_P256",
        "KeySpec": "ECC_NIST_P256",
        "KeyAgreementAlgorithms": [
            "ECDH"
        ],
        "MultiRegion": false
    }
}
```

Asymmetric key specs

The following topics provide technical information about the key specs that AWS KMS supports for asymmetric KMS keys. Information about the SYMMETRIC_DEFAULT key spec for symmetric encryption keys is included for comparison.

Topics

RSA key specs

When you use an RSA key spec, AWS KMS creates an asymmetric KMS key with an RSA key pair. The private key never leaves AWS KMS unencrypted. You can use the public key within AWS KMS, or download the public key for use outside of AWS KMS.

> ⚠ **Warning**
>
> When you encrypt data outside of AWS KMS, be sure that you can decrypt your ciphertext. If you use the public key from a KMS key that has been deleted from AWS KMS, the public key from a KMS key configured for signing and verification, or an encryption algorithm that is not supported by the KMS key, the data is unrecoverable.

In AWS KMS, you can use asymmetric KMS keys with RSA key pairs for encryption and decryption, or signing and verification, but not both. This property, known as *key usage*, is determined separately from the key spec, but you should make that decision before you select a key spec.

AWS KMS supports the following RSA key specs for encryption and decryption or signing and verification:

- RSA_2048
- RSA_3072
- RSA_4096

RSA key specs differ by the length of the RSA key in bits. The RSA key spec that you choose might be determined by your security standards or the requirements of your task. In general, use the largest key that is practical and affordable for your task. Cryptographic operations on KMS keys with different RSA key specs are priced differently. For information about AWS KMS pricing, see AWS Key Management Service Pricing. For information about request quotas, see Request quotas.

RSA key specs for encryption and decryption

When an RSA asymmetric KMS key is used for encryption and decryption, you encrypt with the public key and decrypt with the private key. When you call the `Encrypt` operation in AWS KMS for an RSA KMS key, AWS KMS uses the public key in the RSA key pair and the encryption algorithm you specify to encrypt your data. To decrypt the ciphertext, call the `Decrypt` operation and specify the same KMS key and encryption algorithm. AWS KMS then uses the private key in the RSA key pair to decrypt your data.

You can also download the public key and use it to encrypt data outside of AWS KMS. Be sure to use an encryption algorithm that AWS KMS supports for RSA KMS keys. To decrypt the ciphertext, call the `Decrypt` function with the same KMS key and encryption algorithm.

AWS KMS supports two encryption algorithms for KMS keys with RSA key specs. These algorithms, which are defined in PKCS #1 v2.2, differ in the hash function they use internally. In AWS KMS, the RSAES_OAEP algorithms always use the same hash function for both hashing purposes and for the mask generation function (MGF1). You are required to specify an encryption algorithm when you call the Encrypt and Decrypt operations. You can choose a different algorithm for each request.

Supported encryption algorithms for RSA key specs

Encryption algorithm	Algorithm description
RSAES_OAEP_SHA_1	PKCS #1 v2.2, Section 7.1. RSA encryption with OAEP Padding using SHA-1 for both the hash and in the MGF1 mask generation function along with an empty label.
RSAES_OAEP_SHA_256	PKCS #1, Section 7.1. RSA encryption with OAEP Padding using SHA-256 for both the hash and in the MGF1 mask generation function along with an empty label.

You cannot configure a KMS key to use a particular encryption algorithm. However, you can use the kms:EncryptionAlgorithm policy condition to specify the encryption algorithms that principals are allowed to use with the KMS key.

To get the encryption algorithms for a KMS key, view the cryptographic configuration of the KMS key in the AWS KMS console or use the DescribeKey operation. AWS KMS also provides the key spec and encryption algorithms when you download your public key, either in the AWS KMS console or by using the GetPublicKey operation.

You might choose an RSA key spec based on the length of the plaintext data that you can encrypt in each request. The following table shows the maximum size, in bytes, of the plaintext that you can encrypt in a single call to the Encrypt operation. The values differ with the key spec and encryption algorithm. To compare, you can use a symmetric encryption KMS key to encrypt up to 4096 bytes at one time.

To compute the maximum plaintext length in bytes for these algorithms, use the following formula: (*key_size_in_bits* / 8) - (2 * *hash_length_in_bits*/8) - 2. For example, for RSA_2048 with SHA-256, the maximum plaintext size in bytes is (2048/8) - (2 * 256/8) -2 = 190.

Maximum plaintext size (in bytes) in an Encrypt operation

	Encryption algorithm	
Key spec	**RSAES_OAEP_SHA_1**	**RSAES_OAEP_SHA_256**
RSA_2048	214	190
RSA_3072	342	318
RSA_4096	470	446

RSA key specs for signing and verification

When an RSA asymmetric KMS key is used for signing and verification, you generate the signature for a message with the private key and verify the signature with the public key.

When you call the `Sign` operation in AWS KMS for an asymmetric KMS key, AWS KMS uses the private key in the RSA key pair, the message, and the signing algorithm you specify, to generate a signature. To verify the signature, call the Verify operation. Specify the signature, plus the same

KMS key, message, and signing algorithm. AWS KMS then uses the public key in the RSA key pair to verify the signature. You can also download the public key and use it to verify the signature outside of AWS KMS.

AWS KMS supports the following signing algorithms for all KMS keys with an RSA key spec. You are required to specify a signing algorithm when you call the Sign and Verify operations. You can choose a different algorithm for each request. When signing with RSA key pairs, RSASSA-PSS algorithms are preferred. We include RSASSA-PKCS1-v1_5 algorithms for compatibility with existing applications.

Supported signing algorithms for RSA key specs

Signing algorithm	Algorithm description
RSASSA_PSS_SHA_256	PKCS #1 v2.2, Section 8.1, RSA signature with PSS padding using SHA-256 for both the message digest and the MGF1 mask generatio n function along with a 256-bit salt
RSASSA_PSS_SHA_384	PKCS #1 v2.2, Section 8.1, RSA signature with PSS padding using SHA-384 for both the message digest and the MGF1 mask generatio n function along with a 384-bit salt
RSASSA_PSS_SHA_512	PKCS #1 v2.2, Section 8.1, RSA signature with PSS padding using SHA-512 for both the message digest and the MGF1 mask generatio n function along with a 512-bit salt
RSASSA_PKCS1_V1_5_SHA_256	PKCS #1 v2.2, Section 8.2, RSA signature with PKCS #1v1.5 Padding and SHA-256
RSASSA_PKCS1_V1_5_SHA_384	PKCS #1 v2.2, Section 8.2, RSA signature with PKCS #1v1.5 Padding and SHA-384
RSASSA_PKCS1_V1_5_SHA_512	PKCS #1 v2.2, Section 8.2, RSA signature with PKCS #1v1.5 Padding and SHA-512

You cannot configure a KMS key to use particular signing algorithms. However, you can use the kms:SigningAlgorithm policy condition to specify the signing algorithms that principals are allowed to use with the KMS key.

To get the signing algorithms for a KMS key, view the cryptographic configuration of the KMS key in the AWS KMS console or by using the DescribeKey operation. AWS KMS also provides the key spec and signing algorithms when you download your public key, either in the AWS KMS console or by using the GetPublicKey operation.

Elliptic curve key specs

When you use an elliptic curve (ECC) key spec, AWS KMS creates an asymmetric KMS key with an ECC key pair for signing and verification or deriving shared secrets (but not both). The private key that generates signatures or derives shared secrets never leaves AWS KMS unencrypted. You can use the public key to verify signatures within AWS KMS, or download the public key for use outside of AWS KMS.

AWS KMS supports the following ECC key specs for asymmetric KMS keys.

- Asymmetric NIST-recommended elliptic curve key pairs (signing and verification -or- deriving shared secrets)
  - ECC_NIST_P256 (secp256r1)
  - ECC_NIST_P384 (secp384r1)
  - ECC_NIST_P521 (secp521r1)
- Other asymmetric elliptic curve key pairs (signing and verification)
  - ECC_SECG_P256K1 (secp256k1), commonly used for cryptocurrencies.

The ECC key spec that you choose might be determined by your security standards or the requirements of your task. In general, use the curve with the most points that is practical and affordable for your task.

If you're creating an asymmetric KMS key to derive shared secrets, use one of the NIST-recommended elliptic curve key specs. The only supported key agreement algorithm for deriving shared secrets is the Elliptic Curve Cryptography Cofactor Diffie-Hellman Primitive (ECDH).

If you're creating an asymmetric KMS key to use with cryptocurrencies, use the ECC_SECG_P256K1 key spec. You can also use this key spec for other purposes, but it is required for Bitcoin, and other cryptocurrencies.

KMS keys with different ECC key specs are priced differently and are subject to different request quotas. For information about AWS KMS pricing, see AWS Key Management Service Pricing. For information about request quotas, see Request quotas.

The following table shows the signing algorithms that AWS KMS supports for each of the ECC key specs. You cannot configure a KMS key to use particular signing algorithms. However, you can use the kms:SigningAlgorithm policy condition to specify the signing algorithms that principals are allowed to use with the KMS key.

Supported signing algorithms for ECC key specs

Key spec	Signing algorithm	Algorithm description
ECC_NIST_P256	ECDSA_SHA_256	NIST FIPS 186-4, Section 6.4, ECDSA signature using the curve specified by the key and SHA-256 for the message digest.
ECC_NIST_P384	ECDSA_SHA_384	NIST FIPS 186-4, Section 6.4, ECDSA signature using the curve specified by the key and SHA-384 for the message digest.
ECC_NIST_P521	ECDSA_SHA_512	NIST FIPS 186-4, Section 6.4, ECDSA signature using the curve specified by the key and SHA-512 for the message digest.
ECC_SECG_P256K1	ECDSA_SHA_256	NIST FIPS 186-4, Section 6.4, ECDSA signature using the curve specified by the key and SHA-256 for the message digest.

Deriving shared secrets offline

You can download the public key of your ECC key pair for use in offline operations, that is, operations outside of AWS KMS.

The following OpenSSL walkthrough demonstrates one method of deriving a shared secret outside of AWS KMS using the public key of an ECC KMS key pair and a private key created with OpenSSL.

1. Create an ECC key pair in OpenSSL and prepare it for use with AWS KMS.

```
// Create an ECC key pair in OpenSSL and save the private key in
 openssl_ecc_key_priv.pem
export OPENSSL_CURVE_NAME="P-256"
export KMS_CURVE_NAME="ECC_NIST_P256"

export OPENSSL_KEY1_PRIV_PEM="openssl_ecc_key1_priv.pem"
openssl ecparam -name ${OPENSSL_CURVE_NAME} -genkey -out ${OPENSSL_KEY1_PRIV_PEM}


// Derive the public key from the private key
export OPENSSL_KEY1_PUB_PEM="openssl_ecc_key1_pub.pem"
openssl ec -in ${OPENSSL_KEY1_PRIV_PEM} -pubout -outform pem \
    -out ${OPENSSL_KEY1_PUB_PEM}

// View the PEM file containing the public key and extract the public key as a
// Base64 encoded string into OPENSSL_KEY1_PUB_BASE64 for use with AWS KMS
export OPENSSL_KEY1_PUB_BASE64=`cat ${OPENSSL_KEY1_PUB_PEM} | \
    tee /dev/stderr | grep -v "PUBLIC KEY" | tr -d "\n"`
```

2. Create an ECC key agreement key pair in AWS KMS and prepare it for use with OpenSSL.

```
// Create a KMS key on the same curve as the key pair from step 1
// with a key usage of KEY_AGREEMENT
// Save its ARN in KMS_KEY1_ARN.
export KMS_KEY1_ARN=`aws kms create-key --key-spec ${KMS_CURVE_NAME} \
    --key-usage KEY_AGREEMENT | tee /dev/stderr | jq -r .KeyMetadata.Arn`

// Download the public key and save the Base64-encoded version in KMS_KEY1_PUB_BASE64

export KMS_KEY1_PUB_BASE64=`aws kms get-public-key --key-id ${KMS_KEY1_ARN} | \
    tee /dev/stderr | jq -r .PublicKey`

// Create a PEM file for the public KMS key for use with OpenSSL
```

```
export KMS_KEY1_PUB_PEM="aws_kms_ecdh_key1_pub.pem"
echo "-----BEGIN PUBLIC KEY-----" > ${KMS_KEY1_PUB_PEM}
echo ${KMS_KEY1_PUB_BASE64} | fold -w 64 >> ${KMS_KEY1_PUB_PEM}
echo "-----END PUBLIC KEY-----" >> ${KMS_KEY1_PUB_PEM}
```

3. Derive shared secret in OpenSSL using the private key in OpenSSL and the public KMS key.

```
export OPENSSL_SHARED_SECRET1_BIN="openssl_shared_secret1.bin"
openssl pkeyutl -derive -inkey ${OPENSSL_KEY1_PRIV_PEM} \
    -peerkey ${KMS_KEY1_PUB_PEM} -out ${OPENSSL_SHARED_SECRET1_BIN}
```

SM2 key spec (China Regions only)

The SM2 key spec is an elliptic curve key spec defined within the GM/T series of specifications published by China's Office of State Commercial Cryptography Administration (OSCCA). The SM2 key spec is available only in China Regions. When you use the SM2 key spec, AWS KMS creates an asymmetric KMS key with an SM2 key pair. You can use your SM2 key pair within AWS KMS, or download the public key for use outside of AWS KMS.

Each KMS key can have only one key usage. You can use an SM2 KMS key for signing and verification, encryption and decryption, *or* deriving shared secrets. You must specify the key usage when you create the KMS key, and you cannot change it after the key is created.

If you're creating an asymmetric KMS key to derive shared secrets, use the SM2 key spec. The only supported key agreement algorithm for deriving shared secrets is the Elliptic Curve Cryptography Cofactor Diffie-Hellman Primitive (ECDH).

AWS KMS supports the following SM2 encryption and signing algorithms:

- **SM2PKE** encryption algorithm

  SM2PKE is an elliptic curve based encryption algorithm defined by OSCCA in GM/T 0003.4-2012.
- **SM2DSA** signing algorithm

  SM2DSA is an elliptic curve based signing algorithm defined by OSCCA in GM/T 0003.2-2012. SM2DSA requires a distinguishing ID that is hashed with the SM3 hashing algorithm and then combined with the message, or message digest, that you passed to AWS KMS. This concatenated value is then hashed and signed by AWS KMS.

Offline operations with SM2 (China Regions only)

You can download the public key of your SM2 key pair for use in offline operations, that is, operations outside of AWS KMS. However, when using your SM2 public key offline, you may need to manually perform extra conversions and calculations. SM2DSA operations may require you to provide a distinguishing ID or calculate a message digest. SM2PKE encrypt operations may require you to convert the raw ciphertext output to a format AWS KMS can accept.

To help you with these operations, the `SM2OfflineOperationHelper` class for Java has methods that perform the tasks for you. You can use this helper class as a model for other cryptographic providers.

> ⚠ **Important**
>
> The `SM2OfflineOperationHelper` reference code is designed to be compatible with Bouncy Castle version 1.68. For help with other versions, contact bouncycastle.org.

Offline verification with SM2 key pairs (China Regions only)

To verify a signature outside of AWS KMS with an SM2 public key, you must specify the distinguishing ID. When you pass a raw message, `MessageType:RAW`, to the Sign API, AWS KMS uses the default distinguishing ID, `1234567812345678`, defined by OSCCA in GM/T 0009-2012. You cannot specify your own distinguishing ID within AWS KMS.

However, if you are generating a message digest outside of AWS, you can specify your own distinguishing ID, then pass the message digest, `MessageType:DIGEST`, to AWS KMS to sign. To do this, change the `DEFAULT_DISTINGUISHING_ID` value in the `SM2OfflineOperationHelper` class. The distinguishing ID you specify can be any string up to 8,192 characters long. After AWS KMS signs the message digest, you need either the message digest or the message and the distinguishing ID used to compute the digest to verify it offline.

`SM2OfflineOperationHelper` class

Within AWS KMS, the raw ciphertext conversions and SM2DSA message digest calculations occur automatically. Not all cryptographic providers implement SM2 in the same way. Some libraries, like OpenSSL versions 1.1.1 and later, perform these actions automatically. AWS KMS confirmed this behavior in testing with OpenSSL version 3.0. Use the following `SM2OfflineOperationHelper` class with libraries, like Bouncy Castle, that require you to perform these conversions and calculations manually.

The SM2OfflineOperationHelper class provides methods for the following offline operations:

- **Message digest calculation**

  To generate a message digest offline that you can use for offline verification, or that you can pass to AWS KMS to sign, use the calculateSM2Digest method. The calculateSM2Digest method generates a message digest with the SM3 hashing algorithm. The GetPublicKey API returns your public key in binary format. You must parse the binary key into a Java PublicKey. Provide the parsed public key with the message. The method automatically combines your message with the default distinguishing ID, 1234567812345678, but you can set your own distinguishing ID by changing the DEFAULT_DISTINGUISHING_ID value.

- **Verify**

  To verify a signature offline, use the offlineSM2DSAVerify method. The offlineSM2DSAVerify method uses the message digest calculated from the specified distinguishing ID, and original message you provide to verify the digital signature. The GetPublicKey API returns your public key in binary format. You must parse the binary key into a Java PublicKey. Provide the parsed public key with the original message and the signature you want to verify. For more details, see Offline verification with SM2 key pairs.

- **Encrypt**

  To encrypt plaintext offline, use the offlineSM2PKEEncrypt method. This method ensures the ciphertext is in a format AWS KMS can decrypt. The offlineSM2PKEEncrypt method encrypts the plaintext, and then converts the raw ciphertext produced by SM2PKE to the ASN.1 format. The GetPublicKey API returns your public key in binary format. You must parse the binary key into a Java PublicKey. Provide the parsed public key with the plaintext that you want to encrypt.

  If you're unsure whether you need to perform the conversion, use the following OpenSSL operation to test the format of your ciphertext. If the operation fails, you need to convert the ciphertext to the ASN.1 format.

  ```
  openssl asn1parse -inform DER -in ciphertext.der
  ```

By default, the SM2OfflineOperationHelper class uses the default distinguishing ID, 1234567812345678, when generating message digests for SM2DSA operations.

```
package com.amazon.kms.utils;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import java.io.IOException;
import java.math.BigInteger;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.security.InvalidKeyException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.PrivateKey;
import java.security.PublicKey;

import org.bouncycastle.crypto.CryptoException;
import org.bouncycastle.jce.interfaces.ECPublicKey;

import java.util.Arrays;

import org.bouncycastle.asn1.ASN1EncodableVector;
import org.bouncycastle.asn1.ASN1Integer;
import org.bouncycastle.asn1.DEROctetString;
import org.bouncycastle.asn1.DERSequence;
import org.bouncycastle.asn1.gm.GMNamedCurves;
import org.bouncycastle.asn1.x9.X9ECParameters;
import org.bouncycastle.crypto.CipherParameters;
import org.bouncycastle.crypto.params.ParametersWithID;
import org.bouncycastle.crypto.params.ParametersWithRandom;
import org.bouncycastle.crypto.signers.SM2Signer;
import org.bouncycastle.jcajce.provider.asymmetric.util.ECUtil;

public class SM2OfflineOperationHelper {
    // You can change the DEFAULT_DISTINGUISHING_ID value to set your own
 distinguishing ID,
    // the DEFAULT_DISTINGUISHING_ID can be any string up to 8,192 characters long.
    private static final byte[] DEFAULT_DISTINGUISHING_ID =
 "1234567812345678".getBytes(StandardCharsets.UTF_8);
    private static final X9ECParameters SM2_X9EC_PARAMETERS =
 GMNamedCurves.getByName("sm2p256v1");
```

```
   // ***calculateSM2Digest***
   // Calculate message digest
   public static byte[] calculateSM2Digest(final PublicKey publicKey, final byte[] 
message) throws
           NoSuchProviderException, NoSuchAlgorithmException {
       final ECPublicKey ecPublicKey = (ECPublicKey) publicKey;

       // Generate SM3 hash of default distinguishing ID, 1234567812345678
       final int entlenA = DEFAULT_DISTINGUISHING_ID.length * 8;
       final byte [] entla = new byte[] { (byte) (entlenA & 0xFF00), (byte) (entlenA & 
0x00FF) };
       final byte [] a = SM2_X9EC_PARAMETERS.getCurve().getA().getEncoded();
       final byte [] b = SM2_X9EC_PARAMETERS.getCurve().getB().getEncoded();
       final byte [] xg = SM2_X9EC_PARAMETERS.getG().getXCoord().getEncoded();
       final byte [] yg = SM2_X9EC_PARAMETERS.getG().getYCoord().getEncoded();
       final byte[] xa = ecPublicKey.getQ().getXCoord().getEncoded();
       final byte[] ya = ecPublicKey.getQ().getYCoord().getEncoded();
       final byte[] za = MessageDigest.getInstance("SM3", "BC")
               .digest(ByteBuffer.allocate(entla.length + 
DEFAULT_DISTINGUISHING_ID.length + a.length + b.length + xg.length + yg.length +
                       xa.length + 
ya.length).put(entla).put(DEFAULT_DISTINGUISHING_ID).put(a).put(b).put(xg).put(yg).put(xa).put
                       .array());

       // Combine hashed distinguishing ID with original message to generate final 
digest
       return MessageDigest.getInstance("SM3", "BC")
               .digest(ByteBuffer.allocate(za.length + 
message.length).put(za).put(message)
                       .array());
   }

   // ***offlineSM2DSAVerify***
   // Verify digital signature with SM2 public key
   public static boolean offlineSM2DSAVerify(final PublicKey publicKey, final byte [] 
message,
           final byte [] signature) throws InvalidKeyException {
       final SM2Signer signer = new SM2Signer();
       CipherParameters cipherParameters = 
ECUtil.generatePublicKeyParameter(publicKey);
       cipherParameters = new ParametersWithID(cipherParameters, 
DEFAULT_DISTINGUISHING_ID);
       signer.init(false, cipherParameters);
       signer.update(message, 0, message.length);
       return signer.verifySignature(signature);
   }

   // ***offlineSM2PKEEncrypt***
   // Encrypt data with SM2 public key
   public static byte[] offlineSM2PKEEncrypt(final PublicKey publicKey, final byte []
plaintext) throws
           NoSuchPaddingException, NoSuchAlgorithmException, NoSuchProviderException,
InvalidKeyException,
           BadPaddingException, IllegalBlockSizeException, IOException {
       final Cipher sm2Cipher = Cipher.getInstance("SM2", "BC");
       sm2Cipher.init(Cipher.ENCRYPT_MODE, publicKey);

       // By default, Bouncy Castle returns raw ciphertext in the c1c2c3 format
       final byte [] cipherText = sm2Cipher.doFinal(plaintext);

       // Convert the raw ciphertext to the ASN.1 format before passing it to AWS KMS
       final ASN1EncodableVector asn1EncodableVector = new ASN1EncodableVector();
       final int coordinateLength = (SM2_X9EC_PARAMETERS.getCurve().getFieldSize() +
7) / 8 * 2 + 1;
       final int sm3HashLength = 32;
       final int xCoordinateInCipherText = 33;
       final int yCoordinateInCipherText = 65;
       byte[] coords = new byte[coordinateLength];
       byte[] sm3Hash = new byte[sm3HashLength];
       byte[] remainingCipherText = new byte[cipherText.length - coordinateLength -
sm3HashLength];

       // Split components out of the ciphertext
       System.arraycopy(cipherText, 0, coords, 0, coordinateLength);
       System.arraycopy(cipherText, cipherText.length - sm3HashLength, sm3Hash, 0,
sm3HashLength);
       System.arraycopy(cipherText, coordinateLength, remainingCipherText,
0,cipherText.length - coordinateLength - sm3HashLength);

       // Build standard SM2PKE ASN.1 ciphertext vector
       asn1EncodableVector.add(new ASN1Integer(new BigInteger(1,
Arrays.copyOfRange(coords, 1, xCoordinateInCipherText))));
       asn1EncodableVector.add(new ASN1Integer(new BigInteger(1,
Arrays.copyOfRange(coords, xCoordinateInCipherText, yCoordinateInCipherText))));
       asn1EncodableVector.add(new DEROctetString(sm3Hash));
       asn1EncodableVector.add(new DEROctetString(remainingCipherText));

       return new DERSequence(asn1EncodableVector).getEncoded("DER");
```

```
    }
}
```

SYMMETRIC_DEFAULT key spec

The default key spec, SYMMETRIC_DEFAULT, is the key spec for symmetric encryption KMS keys. When you select the **Symmetric** key type and the **Encrypt and decrypt** key usage in the AWS KMS console, it selects the `SYMMETRIC_DEFAULT` key spec. In the CreateKey operation, if you don't specify a `KeySpec` value, SYMMETRIC_DEFAULT is selected. If you don't have a reason to use a different key spec, SYMMETRIC_DEFAULT is a good choice.

SYMMETRIC_DEFAULT currently represents AES-256-GCM, a symmetric algorithm based on Advanced Encryption Standard (AES) in Galois Counter Mode (GCM) with 256-bit keys, an industry standard for secure encryption. The ciphertext that this algorithm generates supports additional authenticated data (AAD), such as an encryption context, and GCM provides an additional integrity check on the ciphertext. For technical details, see AWS Key Management Service Cryptographic Details.

Data encrypted under AES-256-GCM is protected now and in the future. Cryptographers consider this algorithm to be *quantum resistant*. Theoretical future, large-scale quantum computing attacks on ciphertexts created under 256-bit AES-GCM keys reduce the effective security of the key to 128 bits. But, this security level is sufficient to make brute force attacks on AWS KMS ciphertexts infeasible.

The only exception in China Regions, where SYMMETRIC_DEFAULT represents a 128-bit symmetric key that uses SM4 encryption. You can only create a 128-bit SM4 key within China Regions. You cannot create a 256-bit AES-GCM KMS key in China Regions.

You can use a symmetric encryption KMS key in AWS KMS to encrypt, decrypt, and re-encrypt data, and to protect generated data keys and data key pairs. AWS services that are integrated with AWS KMS use symmetric encryption KMS keys to encrypt your data at rest. You can import your own key material into a symmetric encryption KMS key and create symmetric encryption KMS keys in custom key stores. For a table comparing the operations that you can perform on symmetric and asymmetric KMS keys, see Comparing Symmetric and Asymmetric KMS keys.

For technical details about AWS KMS and symmetric encryption keys, see AWS Key Management Service Cryptographic Details.

HMAC keys in AWS KMS

Hash-Based Message Authentication Code (HMAC) KMS keys are symmetric keys that you use to generate and verify HMACs within AWS KMS. The unique key material associated with each HMAC KMS key provides the secret key that HMAC algorithms require. You can use an HMAC KMS key with the `GenerateMac` and `VerifyMac` operations to verify the integrity and authenticity of data within AWS KMS.

HMAC algorithms combine a cryptographic hash function and a shared secret key. They take a message and a secret key, such as the key material in an HMAC KMS key, and return a unique, fixed-size code or *tag*. If even one character of the message changes, or if the secret key is not identical, the resulting tag is entirely different. By requiring a secret key, HMAC also provides authenticity; it is impossible to generate an identical HMAC tag without the secret key. HMACs are sometimes called *symmetric signatures*, because they work like digital signatures, but use a single key for both signing and verification.

HMAC KMS keys and the HMAC algorithms that AWS KMS uses conform to industry standards defined in RFC 2104. The AWS KMS GenerateMac operation generates standard HMAC tags. HMAC KMS keys are generated in AWS KMS hardware security modules that are certified under the FIPS 140-2 Cryptographic Module Validation Program (except in China (Beijing) and China (Ningxia) Regions) and never leave AWS KMS unencrypted. To use an HMAC KMS key, you must call AWS KMS.

You can use HMAC KMS keys to determine the authenticity of a message, such as a JSON Web Token (JWT), tokenized credit card information, or a submitted password. They can also be used as secure Key Derivation Functions (KDFs), especially in applications that require deterministic keys.

HMAC KMS keys provide an advantage over HMACs from application software because the key material is generated and used entirely within AWS KMS, subject to the access controls that you set on the key.

> **Tip**
>
> Best practices recommend that you limit the time during which any signing mechanism, including an HMAC, is effective. This deters an attack where the actor uses a signed message to establish validity repeatedly or long after the message is superseded. HMAC tags do not include a timestamp, but you can include a timestamp in the token or message to help you detect when its time to refresh the HMAC.

Authorized users can create, manage, and use the HMAC KMS keys in your AWS account. This includes enabling and disabling keys, setting and changing aliases and tags, and scheduling deletion of HMAC KMS keys. You can also control access to HMAC KMS keys using key policies, IAM policies, and grants. You can audit all operations that use or manage your HMAC KMS keys within AWS in AWS CloudTrail logs. You can create HMAC KMS keys with imported key material. You can also create HMAC multi-Region KMS keys that behave like copies of the same HMAC KMS key in multiple AWS Regions.

HMAC KMS keys support only the `GenerateMac` and `VerifyMac` cryptographic operations. You cannot use HMAC KMS keys to encrypt data or sign messages, or use any other type of KMS key in HMAC operations. When you use the `GenerateMac` operation, you supply a message of up to 4,096 bytes, an HMAC KMS key, and the MAC algorithm that is compatible with the HMAC key spec, and `GenerateMac` computes the HMAC tag. To verify an HMAC tag, you must supply the HMAC tag, and the same message, HMAC KMS key, and MAC algorithm that `GenerateMac` used to compute the original HMAC tag. The `VerifyMac` operation computes the HMAC tag and verifies that it is identical to the supplied HMAC tag. If the input and computed HMAC tags are not identical, verification fails.

HMAC KMS keys *do not* support automatic key rotation and you cannot create an HMAC KMS key in a custom key store.

If you are creating a KMS key to encrypt data in an AWS service, use a symmetric encryption key. You cannot use an HMAC KMS key.

Regions

HMAC KMS keys are supported in all AWS Regions that AWS KMS supports.

Learn more

- For help with choosing a type of KMS key, see Choosing a KMS key type.
- For a table that compares the AWS KMS API operations supported by each type of KMS key, see Key type reference.
- For information about creating multi-Region HMAC KMS keys, see Multi-Region keys in AWS KMS.
- To examine the difference in the default key policy that the AWS KMS console sets for HMAC KMS keys, see the section called “Allows key users to use the KMS key with AWS services”.
- For information about pricing of HMAC KMS keys, see AWS Key Management Service pricing.

- For information about quotas that apply to HMAC KMS keys, see Resource quotas and Request quotas.
- For information about deleting HMAC KMS keys, see Deleting AWS KMS keys.
- To learn about using HMACs to create JSON web tokens, see How to protect HMACs inside AWS KMS in the *AWS Security Blog*.
- Listen to a podcast: Introducing HMACs for AWS Key Management Service on *The Official AWS Podcast*.

Topics

Key specs for HMAC KMS keys

AWS KMS supports symmetric HMAC keys in varying lengths. The key spec that you select can depend on your security, regulatory, or business requirements. The length of the key determines the MAC algorithm that is used in GenerateMac and VerifyMac operations. In general, longer keys are more secure. Use the longest key that is practical for your use case.

HMAC key spec	MAC algorithm
HMAC_224	HMAC_SHA_224
HMAC_256	HMAC_SHA_256
HMAC_384	HMAC_SHA_384
HMAC_512	HMAC_SHA_512

Creating HMAC KMS keys

You can create HMAC KMS keys in the AWS KMS console, by using the CreateKey API, or by using an AWS CloudFormation template.

AWS KMS supports multiple key specs for HMAC KMS keys. The key spec that you select might be determined by regulatory, security, or business requirements. In general, longer keys are more resistant to brute-force attacks.

> ⚠ **Important**
>
> Do not include confidential or sensitive information in the alias, description, or tags. These fields may appear in plain text in CloudTrail logs and other output.

If you are creating a KMS key to encrypt data in an AWS service, use a symmetric encryption KMS key. AWS services that integrate with AWS KMS do not support asymmetric KMS keys or HMAC KMS keys. For help with creating a symmetric encryption KMS key, see Creating keys.

Learn more

- To determine which kind of KMS key to create, see Choosing a KMS key type.
- You can use the procedures described in this topic to create a multi-Region *primary* HMAC KMS key. To replicate a multi-Region HMAC key, see the section called "Creating replica keys".
- For information about the permissions required to create KMS keys, see Permissions for creating KMS keys.
- For information about using an AWS CloudFormation template to create an HMAC KMS key, see AWS::KMS::Key in the *AWS CloudFormation User Guide*.

Topics

- Creating HMAC KMS keys (console)
- Creating HMAC KMS keys (AWS KMS API)

Creating HMAC KMS keys (console)

You can use the AWS Management Console to create HMAC KMS keys. HMAC KMS keys are symmetric keys with a key usage of **Generate and verify MAC**. You can also create multi-Region HMAC keys.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.

3. In the navigation pane, choose **Customer managed keys**.
4. Choose **Create key**.
5. For **Key type**, choose **Symmetric**.

   HMAC KMS keys are symmetric. You use the same key to generate and verify HMAC tags.
6. For **Key usage**, choose **Generate and verify MAC**.

   Generate and verify MAC is the only valid key usage for HMAC KMS keys.

   > **Note**
   >
   > **Key usage** is displayed for symmetric keys only when HMAC KMS keys are supported in your selected Region.
7. Select a specification (**Key spec**) for your HMAC KMS key.

   The key spec that you select can be determined by regulatory, security, or business requirements. In general, longer keys are more secure.
8. To create a multi-Region *primary* HMAC key, in **Advanced options**, choose **Multi-Region key**. The shared properties that you define for this KMS key, such as its key type and key usage, will be shared with its replica keys. For details, see Creating multi-Region keys.

   You cannot use this procedure to create a replica key. To create a multi-Region *replica* HMAC key, follow the instructions for creating a replica key.
9. Choose **Next**.
10. Enter an alias for the KMS key. The alias name cannot begin with **`aws/`**. The **`aws/`** prefix is reserved by Amazon Web Services to represent AWS managed keys in your account.

    We recommend that you use an alias that identifies the KMS key as an HMAC key, such as `HMAC/test-key`. This will make it easier for you to identify your HMAC keys in the AWS KMS console where you can sort and filter keys by tags and aliases, but not by key spec or key usage.

    Aliases are required when you create a KMS key in the AWS Management Console. You cannot specify an alias when you use the CreateKey operation, but you can use the console or the CreateAlias operation to create an alias for an existing KMS key. For details, see Using aliases.
11. (Optional) Enter a description for the KMS key.

Enter a description that explains the type of data you plan to protect or the application you plan to use with the KMS key.

You can add a description now or update it any time unless the key state is `Pending Deletion` or `Pending Replica Deletion`. To add, change, or delete the description of an existing customer managed key, edit the description in the AWS Management Console or use the UpdateKeyDescription operation.

12. (Optional) Enter a tag key and an optional tag value. To add more than one tag to the KMS key, choose **Add tag**.

    Consider adding a tag that identifies the key as an HMAC key, such as `Type=HMAC`. This will make it easier for you to identify your HMAC keys in the AWS KMS console where you can sort and filter keys by tags and aliases, but not by key spec or key usage.

    When you add tags to your AWS resources, AWS generates a cost allocation report with usage and costs aggregated by tags. Tags can also be used to control access to a KMS key. For information about tagging KMS keys, see Tagging keys and ABAC for AWS KMS.

13. Choose **Next**.
14. Select the IAM users and roles that can administer the KMS key.

    > **Note**
    >
    > This key policy gives the AWS account full control of this KMS key. It allows account administrators to use IAM policies to give other principals permission to manage the KMS key. For details, see the section called "Default key policy".
    > IAM best practices discourage the use of IAM users with long-term credentials. Whenever possible, use IAM roles, which provide temporary credentials. For details, see Security best practices in IAM in the *IAM User Guide*.

15. (Optional) To prevent the selected IAM users and roles from deleting this KMS key, in the **Key deletion** section at the bottom of the page, clear the **Allow key administrators to delete this key** check box.
16. Choose **Next**.
17. Select the IAM users and roles that can use the KMS key for cryptographic operations.

> **Note**
>
> This key policy gives the AWS account full control of this KMS key. It allows account administrators to use IAM policies to give other principals permission to use the KMS key in cryptographic operations. For details, see the section called "Default key policy".
>
> IAM best practices discourage the use of IAM users with long-term credentials. Whenever possible, use IAM roles, which provide temporary credentials. For details, see Security best practices in IAM in the *IAM User Guide*.

18. (Optional) You can allow other AWS accounts to use this KMS key for cryptographic operations. To do so, in the **Other AWS accounts** section at the bottom of the page, choose **Add another AWS account** and enter the AWS account identification number of an external account. To add multiple external accounts, repeat this step.

> **Note**
>
> To allow principals in the external accounts to use the KMS key, Administrators of the external account must create IAM policies that provide these permissions. For more information, see Allowing users in other accounts to use a KMS key.

19. Choose **Next**.
20. Review the key settings that you chose. You can still go back and change all settings.
21. Choose **Finish** to create the HMAC KMS key.

Creating HMAC KMS keys (AWS KMS API)

You can use the CreateKey operation to create an HMAC KMS key. These examples use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

When you create an HMAC KMS key, you must specify the `KeySpec` parameter, which determines the type of the KMS key. Also, you must specify a `KeyUsage` value of GENERATE_VERIFY_MAC, even though it's the only valid key usage value for HMAC keys. To create a multi-Region HMAC KMS key, add the `MultiRegion` parameter with a value of `true`. You cannot change these properties after the KMS key is created.

The `CreateKey` operation doesn't let you specify an alias, but you can use the CreateAlias operation to create an alias for your new KMS key. We recommend that you use an alias that identifies the KMS key as an HMAC key, such as `HMAC/test-key`. This will make it easier for you to identify your HMAC keys in the AWS KMS console where you can sort and filter keys by alias, but not by key spec or key usage.

If you try to create an HMAC KMS key in an AWS Region in which HMAC keys are not supported, the `CreateKey` operation returns an `UnsupportedOperationException`

The following example uses the `CreateKey` operation to create a 512-bit HMAC KMS key.

```
$ aws kms create-key --key-spec HMAC_512 --key-usage GENERATE_VERIFY_MAC
{
    "KeyMetadata": {
        "KeyState": "Enabled",
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "KeyManager": "CUSTOMER",
        "Description": "",
        "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "CreationDate": 1669973196.214,
        "MultiRegion": false,
        "KeySpec": "HMAC_512",
        "CustomerMasterKeySpec": "HMAC_512",
        "KeyUsage": "GENERATE_VERIFY_MAC",
        "MacAlgorithms": [
            "HMAC_SHA_512"
        ],
        "AWSAccountId": "111122223333",
        "Origin": "AWS_KMS",
        "Enabled": true
    }
}
```

Controlling access to HMAC KMS keys

To control access to an HMAC KMS key, you use a key policy, which is required for every KMS key. You can also use IAM policies and grants.

The default key policy for HMAC keys created in the AWS KMS console gives key users permission to call the `GenerateMac` and `VerifyMac` operations. However, it does not include the key policy statement designed for using grants with AWS services. If you create HMAC keys by using the `CreateKey` operation, you must specify these permissions in the key policy or an IAM policy.

You can use AWS global condition keys and AWS KMS condition keys to refine and limit permissions to HMAC keys. For example, you can use the kms:ResourceAliases condition key to control access to AWS KMS operations based on the aliases associated with an HMAC key. The following AWS KMS policy conditions are useful for policies on HMAC keys.

- Use a kms:MacAlgorithm condition key to limit the algorithms that the principals can request when they call the GenerateMac and VerifyMac operations. For example, you can allow principals to call the GenerateMac operations but only when the MAC algorithm in the request is HMAC_SHA_384.
- Use a kms:KeySpec condition key to allow or prevent principals from creating certain types of HMAC keys. For example, to allow principals to create only HMAC keys, you can allow the CreateKey operation, but use the kms:KeySpec condition to allow only keys with an HMAC_384 key spec.

  You can also use the kms:KeySpec condition key to control access to other operations on a KMS key based on the key spec of the key. For example, you can allow principals to schedule and cancel key deletion only on KMS keys with an HMAC_256 key spec.
- Use the kms:KeyUsage condition key to allow or prevent principals from creating any HMAC keys. For example, to allow principals to create only HMAC keys, you can allow the CreateKey operation, but use the kms:KeyUsage condition to allow only keys with a GENERATE_VERIFY_MAC key usage.

  You can also use the kms:KeyUsage condition key to control access to other operations on a KMS key based on the key usage of the key. For example, you can allow principals to enable and disable only on KMS keys with a GENERATE_VERIFY_MAC key usage.

You can also create grants for GenerateMac and VerifyMac operations, which are grant operations. However, you cannot use an encryption context grant constraint in a grant for an HMAC key. The HMAC tag format does not support encryption context values.

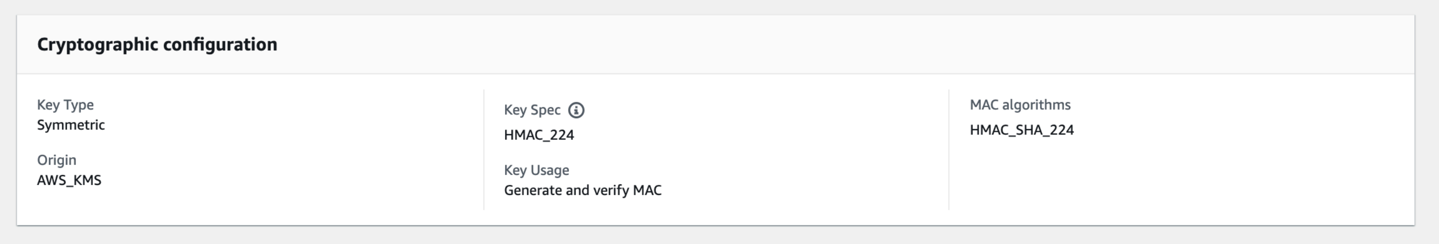
Viewing HMAC KMS keys

You can view HMAC KMS keys in the AWS KMS console or by using the DescribeKey API. You can monitor the use of your HMAC KMS keys in AWS CloudTrail logs and in Amazon CloudWatch. For basic instructions on viewing KMS keys, see Viewing keys.

You can distinguish HMAC KMS keys from other types of KMS keys by their key spec, which begins with HMAC, or their key usage, which is always **Generate and verify MAC** (GENERATE_VERIFY_MAC).

HMAC KMS keys are included in the table on the **Customer managed keys** page of the AWS KMS console. However, you cannot sort or filter KMS keys by key spec or key usage. To make it easier to find your HMAC keys, assign them a distinctive alias or tag. Then you can sort or filter by the alias or tag.

On the key details page for a HMAC KMS key, you can find its configuration details on the **Cryptographic configuration** tab.



Cryptographic configuration

Key Type
Symmetric

Origin
AWS_KMS

Key Spec
HMAC_224

Key Usage
Generate and verify MAC

MAC algorithms
HMAC_SHA_224

Multi-Region keys in AWS KMS

AWS KMS supports *multi-Region keys*, which are AWS KMS keys in different AWS Regions that can be used interchangeably – as though you had the same key in multiple Regions. Each set of *related* multi-Region keys has the same key material and key ID, so you can encrypt data in one AWS Region and decrypt it in a different AWS Region without re-encrypting or making a cross-Region call to AWS KMS.

Like all KMS keys, multi-Region keys never leave AWS KMS unencrypted. You can create symmetric or asymmetric multi-Region keys for encryption or signing, create HMAC multi-Region keys for generating and verifying HMAC tags, and create multi-Region keys with imported key material or key material that AWS KMS generates. You must manage each multi-Region key independently, including creating aliases and tags, setting their key policies and grants, and enabling and disabling them selectively. You can use multi-Region keys in all cryptographic operations that you can do with single-Region keys.

Multi-Region keys are a flexible and powerful solution for many common data security scenarios.

Disaster recovery

In a backup and recovery architecture, multi-Region keys let you process encrypted data without interruption even in the event of an AWS Region outage. Data maintained in backup

Regions can be decrypted in the backup Region, and data newly encrypted in the backup Region can be decrypted in the primary Region when that Region is restored.

Global data management

Businesses that operate globally need globally distributed data that is available consistently across AWS Regions. You can create multi-Region keys in all Regions where your data resides, then use the keys as though they were a single-Region key without the latency of a cross-Region call or the cost of re-encrypting data under a different key in each Region.

Distributed signing applications

Applications that require cross-Region signature capabilities can use multi-Region asymmetric signing keys to generate identical digital signatures consistently and repeatedly in different AWS Regions.

If you use certificate chaining with a single global trust store (for a single root certificate authority (CA), and Regional intermediate CAs signed by the root CA, you don't need multi-Region keys. However, if your system doesn't support intermediate CAs, such as application signing, you can use multi-Region keys to bring consistency to Regional certifications.

Active-active applications that span multiple Regions

Some workloads and applications can span multiple Regions in active-active architectures. For these applications, multi-Region keys can reduce complexity by providing the same key material for concurrent encrypt and decrypt operations on data that might be moving across Region boundaries.

You can use multi-Region keys with client-side encryption libraries, such as the AWS Encryption SDK, the DynamoDB Encryption Client, and Amazon S3 client-side encryption. For an example of using multi-Region keys with Amazon DynamoDB global tables and the DynamoDB Encryption Client, see Encrypt global data client-side with AWS KMS multi-Region keys in the AWS Security Blog.

AWS services that integrate with AWS KMS for encryption at rest or digital signatures currently treat multi-Region keys as though they were single-Region keys. They might re-wrap or re-encrypt data moved between Regions. For example, Amazon S3 cross-region replication decrypts and re-encrypts data under a KMS key in the destination Region, even when replicating objects protected by a multi-Region key.

Multi-Region keys are not global. You create a multi-Region primary key and then replicate it into Regions that you select within an AWS partition. Then you manage the multi-Region key in each Region independently. Neither AWS nor AWS KMS ever automatically creates or replicates multi-Region keys into any Region on your behalf. AWS managed keys, the KMS keys that AWS services create in your account for you, are always single-Region keys.

You cannot convert an existing single-Region key to a multi-Region key. This design ensures that all data protected with existing single-Region keys maintain the same data residency and data sovereignty properties.

For most data security needs, the Regional isolation and fault tolerance of Regional resources make standard AWS KMS single-Region keys a best-fit solution. However, when you need to encrypt or sign data in client-side applications across multiple Regions, multi-Region keys might be the solution.

Regions

Multi-Region keys are supported in all AWS Regions that AWS KMS supports except for China (Beijing) and China (Ningxia).

Pricing and quotas

Every key in a set of related multi-Region keys counts as one KMS key for pricing and quotas. AWS KMS quotas are calculated separately for each Region of an account. Use and management of the multi-Region keys in each Region count toward the quotas for that Region.

Supported KMS key types

You can create the following types of multi-Region KMS keys:

- Symmetric encryption KMS keys
- Asymmetric KMS keys
- HMAC KMS keys
- KMS keys with imported key material

You cannot create multi-Region keys in a custom key store.

Topics

Security considerations for multi-Region keys

Use an AWS KMS multi-Region key only when you need one. Multi-Region keys provide a flexible and scalable solution for workloads that move encrypted data between AWS Regions or need cross-Region access. Consider a multi-Region key if you must share, move, or back up protected data across Regions or need to create identical digital signatures of applications operating in different Regions.

However, the process of creating a multi-Region key moves your key material across AWS Region boundaries within AWS KMS. The ciphertext generated by a multi-Region key can potentially be decrypted by multiple related keys in multiple geographic locations. There are also significant benefits to Regionally-isolated services and resources. Each AWS Region is isolated and independent of the other Regions. Regions provide fault tolerance, stability, and resilience, and can also reduce latency. They enable you to create redundant resources that remain available and unaffected by an outage in another Region. In AWS KMS, they also ensure that every ciphertext can be decrypted by only one key.

Multi-Region keys also raise new security considerations:

- Controlling access and enforcing data security policy is more complex with multi-Region keys. You need to ensure that policy is audited consistently on key across multiple, isolated regions. And you need to use policy to enforce boundaries, instead of relying on separate keys.

  For example, you need to set policy conditions on data to prevent payroll teams in one Region from being able to read payroll data for a different Region. Also, you must use access control to prevent a scenario where a multi-Region key in one Region protects one tenant's data and a related multi-Region key in another Region protects a different tenant's data.

- Auditing keys across Regions is also more complex. With multi-Region keys, you need to examine and reconcile audit activities across multiple Regions to gain a complete understanding of key activities on protected data.
- Compliance with data residency mandates can be more complex. With isolated Regions, you can ensure data residency and data sovereignty compliance. KMS keys in a given Region can decrypt sensitive data only in that Region. Data encrypted in one Region can remain completely protected and inaccessible in any other Region.

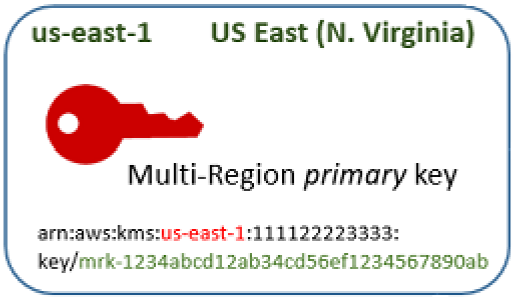
  To verify data residency and data sovereignty with multi-Region keys, you need to implement access policies and compile AWS CloudTrail events across multiple Regions.

To make it easier for you to manage access control on multi-Region keys, the permission to replicate a multi-Region key (kms:ReplicateKey) is separate from the standard permission to create keys (kms:CreateKey). Also, AWS KMS supports several policy conditions for multi-Region keys, including `kms:MultiRegion`, which allows or denies permission to create, use, or manage multi-Region keys and `kms:ReplicaRegion`, which restricts the Regions into which a multi-Region key can be replicated. For details, see Controlling access to multi-Region keys.

How multi-Region keys work

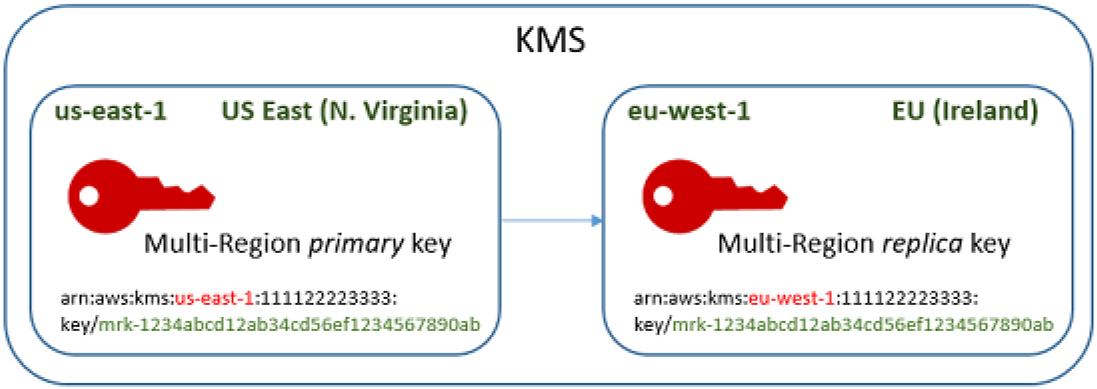
You begin by creating a symmetric or asymmetric multi-Region primary key in an AWS Region that AWS KMS supports, such as US East (N. Virginia). You decide whether a key is single-Region or multi-Region only when you create it; you can't change this property later. As with any KMS key, you set a key policy for the multi-Region key, and you can create grants, and add aliases and tags for categorization and authorization. (These are independent properties that aren't shared or synchronized with other keys.) You can use your multi-Region primary key in cryptographic operations for encryption or signing.

You can create a multi-Region primary key in the AWS KMS console or by using the CreateKey API with the `MultiRegion` parameter set to `true`. Notice that multi-Region keys have a distinctive key ID that begins with `mrk-`. You can use the `mrk-` prefix to identify MRKs programmatically.


us-east-1
US East (N. Virginia)
Multi-Region *primary* key
arn:aws:kms:us-east-1:111122223333:
key/mrk-1234abcd12ab34cd56ef1234567890ab

If you choose, you can replicate the multi-Region primary key into one or more different AWS Regions in the same AWS partition, such as Europe (Ireland). When you do, AWS KMS creates a replica key in the specified Region with the same key ID and other shared properties as the primary key. Then it securely transports the key material across the Region boundary and associates it with the new KMS key in the destination Region, all within AWS KMS. The result is two *related* multi-Region keys — a primary key and a replica key — that can be used interchangeably.

You can create a multi-Region replica key in the AWS KMS console or by using the ReplicateKey API.


KMS
us-east-1
US East (N. Virginia)
Multi-Region *primary* key
arn:aws:kms:us-east-1:111122223333:
key/mrk-1234abcd12ab34cd56ef1234567890ab
eu-west-1
EU (Ireland)
Multi-Region *replica* key
arn:aws:kms:eu-west-1:111122223333:
key/mrk-1234abcd12ab34cd56ef1234567890ab

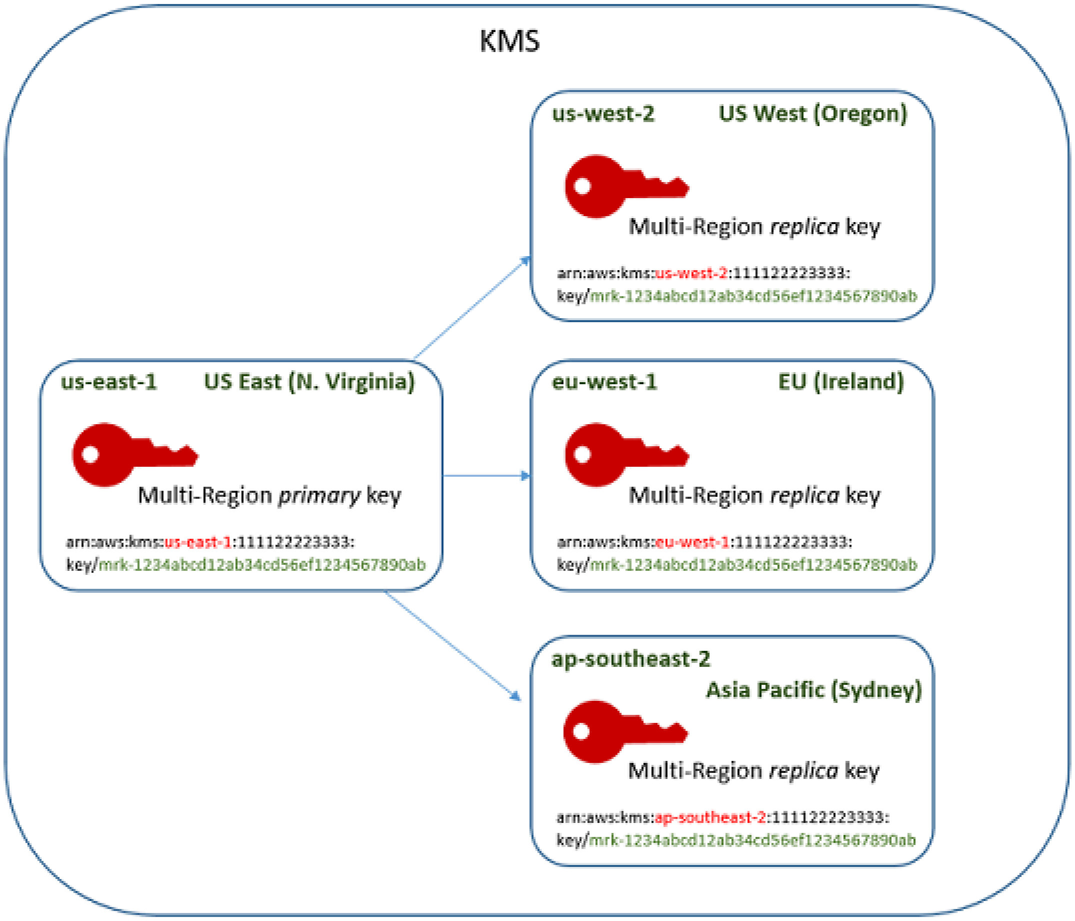
The resulting multi-Region replica key is a fully-functional KMS key with the same shared properties as the primary key. In all other respects, it is an independent KMS key with its own description, key policy, grants, aliases, and tags. Enabling or disabling a multi-Region key has no effect on related multi-Region keys. You can use the primary and replica keys independently in cryptographic operations or coordinate their use. For example, you can encrypt data with the primary key in the US East (N. Virginia) Region, move the data to the Europe (Ireland) Region and use the replica key to decrypt the data.

Related multi-Region keys have the same key ID. Their key ARNs (Amazon Resource Names) differ only in the Region field. For example, the multi-Region primary key and replica keys might have the following example key ARNs. The key ID – the last element in the key ARN – is identical. Both keys have the distinctive key ID of multi-Region keys, which begins with **mrk-**.

```
Primary key: arn:aws:kms:us-
east-1:111122223333:key/mrk-1234abcd12ab34cd56ef12345678990ab
Replica key: arn:aws:kms:eu-
west-1:111122223333:key/mrk-1234abcd12ab34cd56ef12345678990ab
```

Having the same key ID is required for interoperability. When encrypting, AWS KMS binds the key ID of the KMS key to the ciphertext so the ciphertext can be decrypted only with that KMS key or a KMS key with the same key ID. This feature also makes related multi-Region keys easy to recognize, and it makes it easier to use them interchangeably. For example, when using them in an application, you can refer to related multi-Region keys by their shared key ID. Then, if necessary, specify the Region or ARN to distinguish them.

As your data needs change, you can replicate the primary key to other AWS Regions in the same partition, such as US West (Oregon) and Asia Pacific (Sydney). The result is four *related* multi-Region keys with the same key material and key IDs, as shown in the following diagram. You manage the keys independently. You can use them independently or in a coordinated fashion. For example, you can encrypt data with the replica key in Asia Pacific (Sydney), move the data to US West (Oregon), and decrypt it with the replica key in US West (Oregon).


KMS
us-west-2
US West (Oregon)
Multi-Region replica key
arn:aws:kms:us-west-2:111122223333:
key/mrk-1234abcd12ab34cd56ef1234567890ab
us-east-1
US East (N. Virginia)
Multi-Region primary key
arn:aws:kms:us-east-1:111122223333:
key/mrk-1234abcd12ab34cd56ef1234567890ab
eu-west-1
EU (Ireland)
Multi-Region replica key
arn:aws:kms:eu-west-1:111122223333:
key/mrk-1234abcd12ab34cd56ef1234567890ab
ap-southeast-2
Asia Pacific (Sydney)
Multi-Region replica key
arn:aws:kms:ap-southeast-2:111122223333:
key/mrk-1234abcd12ab34cd56ef1234567890ab

Other considerations for multi-Region keys include the following.

Synchronizing shared properties — If a shared property of the multi-Region keys changes, AWS KMS automatically synchronizes the change from the primary key to all of its replica keys. You cannot request or force a synchronization of shared properties. AWS KMS detects and synchronizes all changes for you. However, you can audit synchronization by using the SynchronizeMultiRegionKey event in CloudTrail logs.

For example, if you enable automatic key rotation on a symmetric multi-Region primary key, AWS KMS copies that setting to all of its replica keys. When the key material is rotated, the rotation is synchronized among all of the related multi-Region keys, so they continue to have the same current key material, and access to all older versions of the key material. If you create a new replica key, it has the same current key material of all related multi-Region keys and access to all previous versions of the key material. For details, see Rotating multi-Region keys.

Changing the primary key — Every set of multi-Region keys must have exactly one primary key. The primary key is the only key that can be replicated. It's also the source of the shared properties of its replica keys. But you can change the primary key to a replica and promote one of the replica keys to primary. You might do this so you can delete a multi-Region primary key from a particular Region, or locate the primary key in a Region closer to project administrators. For details, see Updating the primary Region.

Deleting multi-Region keys — Like all KMS keys, you must schedule the deletion of multi-Region keys before AWS KMS deletes them. While the key is pending deletion, you cannot use it in any cryptographic operations. However, AWS KMS will not delete a multi-Region primary key until all of its replica keys are deleted. For details, see Deleting multi-Region keys.

Concepts

The following terms and concepts are used with multi-Region keys.

Multi-Region key

A *multi-Region key* is one of a set of KMS keys with the same key ID and key material (and other shared properties) in different AWS Regions. Each multi-Region key is a fully functioning KMS key that can be used entirely independently of its related multi-Region keys. Because all *related* multi-Region keys have the same key ID and key material, they are *interoperable*, that is, any related multi-Region key in any AWS Region can decrypt ciphertext encrypted by any other related multi-Region key.

You set the multi-Region property of a KMS key when you create it. You cannot change the multi-Region property on an existing key. You cannot convert a single-Region key to multi-Region key or a convert a multi-Region key to a single-Region key. To move existing workloads into multi-Region scenarios, you must re-encrypt your data or create new signatures with new multi-Region keys.

A multi-Region key can be symmetric or asymmetric and it can use AWS KMS key material or imported key material. You cannot create multi-Region keys in a custom key store.

In a set of related multi-Region keys, there is exactly one primary key at any time. You can create replica keys of that primary key in other AWS Regions. You can also update the primary region, which changes the primary key to a replica key and changes a specified replica key to the primary key. However, you can maintain only one primary key or replica key in each AWS Region. All of the Regions must be in the same AWS partition.

You can have multiple sets of related multi-Region keys in the same or different AWS Regions. Although related multi-Region keys are interoperable, unrelated multi-Region keys are not interoperable.

Primary key

A multi-Region *primary key* is a KMS key that can be replicated into other AWS Regions in the same partition. Each set of multi-Region keys has just one primary key.

A primary key differs from a replica key in the following ways:

- Only a primary key can be replicated.
- The primary key is the source for shared properties of its replica keys, including the key material and key ID.
- You can enable and disable automatic key rotation only on a primary key.
- You can schedule the deletion of a primary key at any time. But AWS KMS will not delete a primary key until all of its replica keys are deleted.

However, primary and replica keys don't differ in any cryptographic properties. You can use a primary key and its replica keys interchangeably.

You are not required to replicate a primary key. You can use it just as you would any KMS key and replicate it if and when it is useful. However, because multi-Region keys have different security properties than single-Region keys, we recommend that you create a multi-Region key only when you plan to replicate it.

Replica key

A multi-Region *replica key* is a KMS key that has the same key ID and key material as its primary key and related replica keys, but exists in a different AWS Region.

A replica key is a fully functional KMS key with it own key policy, grants, alias, tags, and other properties. It is not a copy of or pointer to the primary key or any other key. You can use a replica key even if its primary key and all related replica keys are disabled. You can also convert a replica key to a primary key and a primary key to a replica key. Once it is created, a replica key relies on its primary key only for key rotation and updating the primary Region.

Primary and replica keys don't differ in any cryptographic properties. You can use a primary key and its replica keys interchangeably. Data encrypted by a primary or replica key can be decrypted by the same key, or by any related primary or replica key.

Replicate

You can *replicate* a multi-Region primary key into a different AWS Region in the same partition. When you do, AWS KMS creates a multi-Region replica key in the specified Region with the same key ID and other shared properties as its primary key. Then it securely transports the key material across the Region boundary and associates it with the new replica key, all within AWS KMS.

Shared properties

Shared properties are properties of a multi-Region primary key that are shared with its replica keys. AWS KMS creates the replica keys with the same shared property values as those of the primary key. Then, it periodically synchronizes the shared property values of the primary key to its replica keys. You cannot set these properties on a replica key.

The following are the shared properties of multi-Region keys.

- Key ID — (The `Region` element of the key ARN differs.)
- Key material
- Key material origin
- Key spec and encryption algorithms
- Key usage
- Automatic key rotation — You can enable and disable automatic key rotation only on the primary key. New replica keys are created with all versions of the shared key material. For details, see Rotating multi-Region keys.
- On-demand rotation — You can perform on-demand rotation only on the primary key. New replica keys are created with all versions of the shared key material. For details, see Rotating multi-Region keys.

You can also think of the primary and replica designations of related multi-Region keys as shared properties. When you create new replica keys or update the primary key, AWS KMS synchronizes the change to all related multi-Region keys. When these changes are complete, all related multi-Region keys list their primary key and replica keys accurately.

All other properties of multi-Region keys are *independent properties*, including the description, key policy, grants, enabled and disabled key states, aliases, and tags. You can set the same values for these properties on all related multi-Region keys, but if you change the value of an independent property, AWS KMS does not synchronize it.

You can track the synchronization of the shared properties of your multi-Region keys. In your AWS CloudTrail log, look for the SynchronizeMultiRegionKey event.

Controlling access to multi-Region keys

You can use multi-Region keys in compliance, disaster recovery, and backup scenarios that would be more complex with single-Region keys. However, because the security properties of multi-Region keys are significantly different from those of single-Region keys, we recommend using caution when authorizing the creation, management, and use of multi-Region keys.

> **Note**
>
> Existing IAM policy statements with wildcard characters in the `Resource` field now apply to both single-Region and multi-Region keys. To restrict them to single-Region KMS keys or multi-Region keys, use the kms:MultiRegion condition key.

Use your authorization tools to prevent creation and use of multi-Region keys in any scenario where a single-Region will suffice. Allow principals to replicate a multi-Region key only into AWS Regions that require them. Give permission for multi-Region keys only to principals who need them and only for tasks that require them.

You can use key policies, IAM policies, and grants to allow IAM principals to manage and use multi-Region keys in your AWS account. Each multi-Region key is an independent resource with a unique key ARN and key policy. You need to establish and maintain a key policy for each key and make sure that new and existing IAM policies implement your authorization strategy.

Topics

Authorization basics for multi-Region keys

When designing key policies and IAM policies for multi-Region keys, consider the following principles.

- **Key policy** — Each multi-Region key is an independent KMS key resource with its own key policy. You can apply the same or a different key policy to each key in the set of related multi-Region keys. Key policies are *not* shared properties of multi-Region keys. AWS KMS does not copy or synchronize key policies among related multi-Region keys.

  When you create a replica key in the AWS KMS console, the console displays the current key policy of the primary key as a convenience. You can use this key policy, edit it, or delete and replace it. But even if you accept the primary key policy unchanged, AWS KMS doesn't synchronize the policies. For example, if you change the key policy of the primary key, the key policy of the replica key remains the same.
- **Default key policy** — When you create multi-Region keys by using the CreateKey and `ReplicateKey` operations, the default key policy is applied unless you specify a key policy in the request. This is the same default key policy that is applied to single-Region keys.
- **IAM policies** — As with all KMS keys, you can use IAM policies to control access to multi-Region keys only when the key policy allows it. IAM policies apply to all AWS Regions by default. However, you can use condition keys, such as aws:RequestedRegion, to limit permissions to a particular Region.

  To create primary and replica keys, principals must have `kms:CreateKey` permission in an IAM policy that applies to the Region where the key is created.
- **Grants** — AWS KMS grants are Regional. Each grant allows permissions to one KMS key. You can use grants to allow permissions to a multi-Region primary key or replica key. But you cannot use a single grant to allow permissions to multiple KMS keys, even if they are related multi-Region keys.
- **Key ARN** — Each multi-Region key has a unique key ARN. The key ARNs of related multi-Region keys have the same partition, account, and key ID, but different Regions.

  To apply an IAM policy statement to a particular multi-Region key, use its key ARN or a key ARN pattern that includes the Region. To apply an IAM policy statement to all related multi-Region keys, use a wildcard character (*) in the Region element of the ARN, as shown in the following example.

  ```
  {
    "Effect": "Allow",
    "Action": [
      "kms:Describe*",
      "kms:List*"
    ],
  ```

```
    "Resource": {
        "arn:aws:kms:*::111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab"
    }
}
```

To apply a policy statement to all multi-Region keys in your AWS account, you can use the kms:MultiRegion policy condition or a key ID pattern that includes the distinctive `mrk-` prefix.

- **Service-linked role** — Principals who create multi-Region primary keys must have iam:CreateServiceLinkedRole permission.

  To synchronize the shared properties of related multi-Region keys, AWS KMS assumes an IAM service-linked role. AWS KMS creates the service-linked role in the AWS account whenever you create a multi-Region primary key. (If the role exists, AWS KMS recreates it, which has no harmful effect.) The role is valid in all Regions. To allow AWS KMS to create (or recreate) the service-linked role, principals who create multi-Region primary keys must have iam:CreateServiceLinkedRole permission.

Authorizing multi-Region key administrators and users

Principals who create and manage multi-Region keys need the following permissions in the primary and replica Regions:

- `kms:CreateKey`
- `kms:ReplicateKey`
- `kms:UpdatePrimaryRegion`
- `iam:CreateServiceLinkedRole`

Creating a primary key

To create a multi-Region primary key, the principal needs kms:CreateKey and iam:CreateServiceLinkedRole permissions in an IAM policy that is effective in the primary key's Region. Principals who have these permissions can create single-Region and multi-Region keys unless you restrict their permissions.

The `iam:CreateServiceLinkedRole` permission allows AWS KMS to create the **AWSServiceRoleForKeyManagementServiceMultiRegionKeys** role to synchronize the shared properties of related multi-Region keys.

For example, this IAM policy allows a principal to create any type of KMS key.

```
{
  "Version": "2012-10-17",
  "Statement":{
      "Action": [
        "kms:CreateKey",
        "iam:CreateServiceLinkedRole"
      ],
      "Effect":"Allow",
      "Resource":"*"
  }
}
```

To allow or deny permission to create multi-Region primary keys, use the kms:MultiRegion condition key. Valid values are `true` (multi-Region key) or `false` (single-Region key). For example, the following IAM policy statement uses a `Deny` action with the `kms:MultiRegion` condition key to prevent principals from creating multi-Region keys.

```
{
  "Version": "2012-10-17",
  "Statement":{
      "Action":"kms:CreateKey",
      "Effect":"Deny",
      "Resource":"*",
      "Condition": {
          "Bool": "kms:MultiRegion": true
      }
  }
}
```

Replicating keys

To create a multi-Region replica key, the principal needs the following permissions:

- kms:ReplicateKey permission in the key policy of the primary key.
- kms:CreateKey permission in an IAM policy that is effective in the replica key Region.

Use caution when allowing these permissions. They allow principals to create KMS keys and the key policies that authorize their use. The `kms:ReplicateKey` permission also authorizes the transfer of key material across Region boundaries within AWS KMS.

To restrict the AWS Regions in which a multi-Region key can be replicated, use the kms:ReplicaRegion condition key. It limits only the `kms:ReplicateKey` permission. Otherwise, it has no effect. For example, the following key policy allows the principal to replicate this primary key, but only in the specified Regions.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/Administrator"
  },
  "Action": "kms:ReplicateKey",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ReplicaRegion": [
         "us-east-1",
         "eu-west-3",
         "ap-southeast-2"
      ]
    }
  }
}
```

Updating the primary Region

Authorized principals can convert a replica key to a primary key, which changes the former primary key into a replica. This action is known as updating the primary Region. To update the primary Region, the principal needs kms:UpdatePrimaryRegion permission in both Regions. You can provide these permissions in a key policy or IAM policy.

- `kms:UpdatePrimaryRegion` on the primary key. This permission must be effective in the primary key Region.
- `kms:UpdatePrimaryRegion` on the replica key. This permission must be effective in the replica key Region.

For example, the following key policy gives users who can assume the Administrator role permission to update the primary Region of the KMS key. This KMS key can be the primary key or a replica key in this operation.

```
{
  "Effect": "Allow",
  "Resource": "*",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/Administrator"
  },
  "Action": "kms:UpdatePrimaryRegion"
}
```

To restrict the AWS Regions that can host a primary key, use the kms:PrimaryRegion condition key. For example, the following IAM policy statement allows the principals to update the primary Region of the multi-Region keys in the AWS account, but only when the new primary Region is one of the specified Regions.

```
{
  "Effect": "Allow",
  "Action": "kms:UpdatePrimaryRegion",
  "Resource": {
      "arn:aws:kms:*:111122223333:key/*"
  },
  "Condition": {
    "StringEquals": {
      "kms:PrimaryRegion": [
         "us-west-2",
         "sa-east-1",
         "ap-southeast-1"
      ]
    }
  }
}
```

Using and managing multi-Region keys

By default, principals who have permission to use and manage KMS keys in an AWS account and Region also have permission to use and manage multi-Region keys. However, you can use the kms:MultiRegion condition key to allow only single-Region keys or only multi-Region keys. Or use the kms:MultiRegionKeyType condition key to allow only multi-Region primary keys or only replica keys. Both condition keys controls access to the CreateKey operation and to any operation that uses an existing KMS key, such as Encrypt or EnableKey.

The following example IAM policy statement uses the `kms:MultiRegion` condition key to prevent the principals from using or managing any multi-Region key.

```
{
  "Effect": "Deny",
  "Action": "kms:*",
  "Resource": "*",
  "Condition": {
    "Bool": "kms:MultiRegion": true
  }
}
```

This example IAM policy statement uses the `kms:MultiRegionKeyType` condition to allow principals to schedule and cancel key deletion, but only on multi-Region replica keys.

```
{
  "Effect": "Allow",
  "Action": [
    "kms:ScheduleKeyDeletion",
    "kms:CancelKeyDeletion"
  ],
  "Resource": {
      "arn:aws:kms:us-west-2:111122223333:key/*"
  },
  "Condition": {
    "StringEquals": "kms:MultiRegionKeyType": "REPLICA"
  }
}
```

Authorizing AWS KMS to synchronize multi-Region keys

To support multi-Region keys, AWS KMS uses an IAM service linked role. This role gives AWS KMS the permissions it needs to synchronize shared properties. You can view the SynchronizeMultiRegionKey CloudTrail event that records AWS KMS synchronizing shared properties in your AWS CloudTrail logs.

About the service-linked role for multi-Region keys

A service-linked role is an IAM role that gives one AWS service permission to call other AWS services on your behalf. It's designed to make it easier for you to use the features of multiple integrated AWS services without having to create and maintain complex IAM policies.

For multi-Region keys, AWS KMS creates the **AWSServiceRoleForKeyManagementServiceMultiRegionKeys** service-linked role with the

AWSKeyManagementServiceMultiRegionKeysServiceRolePolicy policy. This policy gives the role the `kms:SynchronizeMultiRegionKey` permission, which allows it to synchronize the shared properties of multi-Region keys.

Because the **AWSServiceRoleForKeyManagementServiceMultiRegionKeys** service-linked role trusts only `mrk.kms.amazonaws.com`, only AWS KMS can assume this service-linked role. This role is limited to the operations that AWS KMS needs to synchronize multi-Region shared properties. It does not give AWS KMS any additional permissions. For example, AWS KMS does not have permission to create, replicate, or delete any KMS keys.

For more information about how AWS services use service-linked roles, see Using Service-Linked Roles in the IAM User Guide.

Create the service-linked role

AWS KMS automatically creates the **AWSServiceRoleForKeyManagementServiceMultiRegionKeys** service-linked role in your AWS account when you create a multi-Region key, if the role does not already exist. You cannot create or re-create this service-linked role directly.

Edit the service-linked role description

You cannot edit the role name or the policy statements in the **AWSServiceRoleForKeyManagementServiceMultiRegionKeys** service-linked role, but you can edit the role description. For instructions, see Editing a Service-Linked Role in the *IAM User Guide*.

Delete the service-linked role

AWS KMS does not delete the **AWSServiceRoleForKeyManagementServiceMultiRegionKeys** service-linked role from your AWS account and you cannot delete it. However, AWS KMS does not assume the **AWSServiceRoleForKeyManagementServiceMultiRegionKeys** role or use any of its permissions unless you have multi-Region keys in your AWS account and Region.

Creating multi-Region keys

You can create multi-Region keys in the console or by using the AWS KMS API.

The multi-Region property that you set in this procedure is immutable. You cannot convert a single-Region key to multi-Region key or a convert a multi-Region key to a single-Region key.

Topics

- Creating multi-Region primary keys
- Creating multi-Region replica keys

Creating multi-Region primary keys

You can create a multi-Region primary key in the AWS KMS console or by using the AWS KMS API. You can create the primary key in any AWS Region where AWS KMS supports multi-Region keys.

To create a multi-Region primary key, the principal needs the same permissions that they need to create any KMS key, including the kms:CreateKey permission in an IAM policy. The principal also needs the iam:CreateServiceLinkedRole permission. You can use the kms:MultiRegionKeyType condition key to allow or deny permission to create multi-Region primary keys.

These instructions create a multi-Region primary key with key material that AWS KMS generates. To create a multi-Region primary key with imported key material, see Creating a primary key with imported key material.

Topics

- Creating a multi-Region primary key (console)
- Creating a multi-Region primary key (AWS KMS API)

Creating a multi-Region primary key (console)

To create a multi-Region primary key in the AWS KMS console, use the same process that you would use to create any KMS key.. You select a multi-Region key in **Advanced options**. For complete instructions, see Creating keys.

⚠ **Important**

Do not include confidential or sensitive information in the alias, description, or tags. These fields may appear in plain text in CloudTrail logs and other output.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.

4. Choose **Create key**.
5. Select a symmetric or asymmetric key type. Symmetric keys are the default.

   You can create multi-Region symmetric and asymmetric keys, including multi-Region HMAC KMS keys, which are symmetric.
6. Select your key usage. **Encrypt and decrypt** is the default.

   For help, see the section called "Creating keys", the section called "Creating asymmetric KMS keys", or the section called "Creating HMAC keys".
7. Expand **Advanced options**.
8. Under **Key material origin**, to have AWS KMS generate the key material that your primary and replica keys will share, choose **KMS**. If you are importing key material into the primary and replica keys, choose **External (Import key material)**.
9. Under **Multi-Region replication**, choose **Allow this key to be replicated into other Regions**.

   You can't change this setting after you create the KMS key.
10. Type an alias for the primary key.

    Aliases are not a shared property of multi-Region keys. You can give your multi-Region primary key and its replicas the same alias or different aliases. AWS KMS does not synchronize the aliases of multi-Region keys.

    > **Note**
    >
    > Adding, deleting, or updating an alias can allow or deny permission to the KMS key. For details, see ABAC for AWS KMS and Using aliases to control access to KMS keys.
11. (Optional) Type a description of the primary key.

    Descriptions are not a shared property of multi-Region keys. You can give your multi-Region primary key and its replicas the same description or different descriptions. AWS KMS does not synchronize the key descriptions of multi-Region keys.
12. (Optional) Type a tag key and an optional tag value. To assign more than one tag to the primary key, choose **Add tag**.

    Tags are not a shared property of multi-Region keys. You can give your multi-Region primary key and its replicas the same tags or different tags. AWS KMS does not synchronize the tags of multi-Region keys. You can change the tags on KMS keys at any time.

> **Note**
>
> Tagging or untagging a KMS key can allow or deny permission to the KMS key. For details, see ABAC for AWS KMS and Using tags to control access to KMS keys.

13. Select the IAM users and roles that can administer the primary key.

    > **Note**
    >
    > IAM policies can give other IAM users and roles permission to manage the KMS key. IAM best practices discourage the use of IAM users with long-term credentials. Whenever possible, use IAM roles, which provide temporary credentials. For details, see Security best practices in IAM in the *IAM User Guide*.

    This step starts the process of creating a key policy for the primary key. Key policies are not a shared property of multi-Region keys. You can give your multi-Region primary key and its replicas the same key policy or different key policies. AWS KMS does not synchronize the key policies of multi-Region keys. You can change the key policy of a KMS key at any time.

14. Complete the steps for creating the key policy, including selecting key users. After you review the key policy, choose **Finish** to create the KMS key.

Creating a multi-Region primary key (AWS KMS API)

To create a multi-Region primary key, use the CreateKey operation. Use the `MultiRegion` parameter with a value of `True`.

For example, the following command creates a multi-Region primary key in the caller's AWS Region (us-east-1). It accepts default values for all other properties, including the key policy. The default values for multi-Region primary keys are the same as the default values for all other KMS keys, including the default key policy. This procedure creates a symmetric encryption key, the default KMS key.

The response includes the `MultiRegion` element and the `MultiRegionConfiguration` element with typical sub-elements and values for a multi-Region primary key with no replica keys. The key ID of a multi-Region key always begins with `mrk-`.

Important

Do not include confidential or sensitive information in the `Description` or `Tags` fields. These fields may appear in plain text in CloudTrail logs and other output.

```
$ aws kms create-key --multi-region
{
    "KeyMetadata": {
        "Origin": "AWS_KMS",
        "KeyId": "mrk-1234abcd12ab34cd56ef1234567890ab",
        "Description": "",
        "KeyManager": "CUSTOMER",
        "Enabled": true,
        "KeySpec": "SYMMETRIC_DEFAULT",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "CreationDate": 1606329032.475,
        "Arn": "arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
        "AWSAccountId": "111122223333",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ],
        "MultiRegion": true,
        "MultiRegionConfiguration": {
            "MultiRegionKeyType": "PRIMARY",
            "PrimaryKey": {
                "Arn": "arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
                "Region": "us-east-1"
            },
            "ReplicaKeys": [ ]
      }
    }
}
```

Creating multi-Region replica keys

You can create a multi-Region replica key in the AWS KMS console, by using the ReplicateKey operation, or by using a AWS CloudFormation template. You cannot use the CreateKey operation to create a replica key.

You can use these procedures to replicate any multi-Region primary key, including a symmetric encryption KMS key, an asymmetric KMS key, or an HMAC KMS key.

When this operation completes, the new replica key has a transient key state of `Creating`. This key state changes to `Enabled` (or PendingImport) after a few seconds when the process of creating the new replica key is complete. While the key state is `Creating`, you can manage key, but you cannot yet use it in cryptographic operations. If you are creating and using the replica key programmatically, retry on `KMSInvalidStateException` or call DescribeKey to check its `KeyState` value before using it.

If you mistakenly delete a replica key, you can use this procedure to recreate it. If you replicate the same primary key in the same Region, the new replica key you create will have the same shared properties as the original replica key.

> ⚠ **Important**
>
> Do not include confidential or sensitive information in the alias, description, or tags. These fields may appear in plain text in CloudTrail logs and other output.

Learn more

- To create a multi-Region replica key with imported key material, see Creating a replica key with imported key material.
- To use a AWS CloudFormation template to create a replica key, see AWS::KMS::ReplicaKey in the *AWS CloudFormation User Guide*.

Topics

Replica Regions

You typically choose to replicate a multi-Region key into an AWS Region based on your business model and regulatory requirements. For example, you might replicate a key into Regions where you keep your resources. Or, to comply with a disaster recovery requirement, you might replicate a key into geographically distant Regions.

The following are the AWS KMS requirements for replica Regions. If the Region that you choose doesn't comply with these requirements, attempts to replicate a key fail.

- **One related multi-Region key per Region** — You can't create a replica key in the same Region as its primary key, or in the same Region as another replica of the primary key.

  If you try to replicate a primary key in a Region that already has a replica of that primary key, the attempt fails. If the current replica key in the Region is in the `PendingDeletion` key state, you can cancel the replica key deletion or wait until the replica key is deleted.
- **Multiple unrelated multi-Region keys in the same Region** — You can have multiple unrelated multi-Region keys in the same Region. For example, you can have two multi-Region primary keys in the `us-east-1` Region. Each of the primary keys can have a replica key in `us-west-2` Region.
- **Regions in the same partition** — The replica key Region must be in the same AWS partition as the primary key Region.
- **Region must be enabled** — If a Region is disabled by default, you cannot create any resources in that Region until it is enabled for your AWS account.

Creating replica keys (console)

In the AWS KMS console, you can create one or many replicas of a multi-Region primary key in the same operation.

This procedure is similar to creating a standard single-Region KMS key in the console. However, because a replica key is based on the primary key, you do not select values for shared properties, such as the key spec (symmetric or asymmetric), key usage, or key origin.

You do specify properties that are not shared, including an alias, tags, a description, and a key policy. As a convenience, the console displays the current property values of the primary key, but you can change them. Even if you keep the primary key values, AWS KMS does not keep these values synchronized.

> ⚠ **Important**
>
> Do not include confidential or sensitive information in the alias, description, or tags. These fields may appear in plain text in CloudTrail logs and other output.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Select the key ID or alias of a multi-Region primary key. This opens the key details page for the KMS key.

   To identify a multi-Region primary key, use the tool icon in the upper right corner to add the **Regionality** column to the table.
5. Choose the **Regionality** tab.
6. In the **Related multi-Region keys** section, choose **Create new replica keys**.

   The **Related multi-Region keys** section displays the Region of the primary key and its replica keys. You can use this display to help you choose the Region for your new replica key.
7. Choose one or more AWS Regions. This procedure creates a replica key in each of the Regions you select.

   The menu includes only Regions in the same AWS partition as the primary key. Regions that already have a related multi-Region key are displayed, but not selectable. You might not have permission to replicate a key into all of the Regions on the menu.

   When you are finished choosing Regions, close the menu. The Regions you chose are displayed. To cancel replication into a Region, choose the **X** beside the Region name.
8. Type an alias for the replica key.

   The console displays one of the current aliases of the primary key, but you can change it. You can give your multi-Region primary key and its replicas the same alias or different aliases. Aliases are not a shared property of multi-Region keys. AWS KMS does not synchronize the aliases of multi-Region keys.

Adding, deleting, or updating an alias can allow or deny permission to the KMS key. For details, see ABAC for AWS KMS and Using aliases to control access to KMS keys.

9. (Optional) Type a description of the replica key.

   The console displays the current description of the primary key, but you can change it. Descriptions are not a shared property of multi-Region keys. You can give your multi-Region primary key and its replicas the same description or different descriptions. AWS KMS does not synchronize the key descriptions of multi-Region keys.

10. (Optional) Type a tag key and an optional tag value. To assign more than one tag to the replica key, choose **Add tag**.

    The console displays the tags currently attached to the primary key, but you can change them. Tags are not a shared property of multi-Region keys. You can give your multi-Region primary key and its replicas the same tags or different tags. AWS KMS does not synchronize the tags of multi-Region keys.

    Tagging or untagging a KMS key can allow or deny permission to the KMS key. For details, see ABAC for AWS KMS and Using tags to control access to KMS keys.

11. Select the IAM users and roles that can administer the replica key.

    > **Note**
    >
    > IAM policies can give other IAM users and roles permission to manage the replica keys. IAM best practices discourage the use of IAM users with long-term credentials. Whenever possible, use IAM roles, which provide temporary credentials. For details, see Security best practices in IAM in the *IAM User Guide*.

    This step begins the process of creating a key policy for the replica key. The console displays the current key policy of the primary key, but you can change it. Key policies are not a shared property of multi-Region keys. You can give your multi-Region primary key and its replicas the same key policy or different key policies. AWS KMS does not synchronize key policies. You can change the key policy of any KMS key at any time.

12. Complete the steps for creating the key policy, including selecting key users. After you review the key policy, choose **Finish** to create the replica key.

Creating a replica key (AWS KMS API)

To create a multi-Region replica key, use the ReplicateKey operation. You cannot use the CreateKey operation to create a replica key. This operation creates one replica key at a time. The Region that you specify must comply with the Region requirements for replica keys.

When you use the `ReplicateKey` operation, you don't specify values for any shared properties of multi-Region keys. Shared property values are copied from the primary key and kept synchronized. However, you can specify values for properties that are not shared. Otherwise, AWS KMS applies the standard default values for KMS keys, not the values of the primary key.

> **Note**
>
> If you don't specify values for the `Description`, `KeyPolicy`, or `Tags` parameters, AWS KMS creates the replica key with an empty string description, the default key policy, and no tags.
> Do not include confidential or sensitive information in the `Description` or `Tags` fields. These fields may appear in plain text in CloudTrail logs and other output.

For example, the following command creates a multi-Region replica key in the Asia Pacific (Sydney) Region (ap-southeast-2). This replica key is modeled on the primary key in the US East (N. Virginia) Region (us-east-1), which is identified by the value of the `KeyId` parameter. This example accepts default values for all other properties, including the key policy.

The response describes the new replica key. It includes fields for shared properties, such as the `KeyId`, `KeySpec`, `KeyUsage`, and key material origin (`Origin`). It also includes properties that are independent of the primary key, such as the `Description`, key policy (`ReplicaKeyPolicy`), and tags (`ReplicaTags`).

The response also includes the key ARN and region of the primary key and all of its replica keys, including the one that was just created in the ap-southeast-2 Region. In this example, the `ReplicaKey` element shows that this primary key was already replicated in the Europe (Ireland) Region (eu-west-1).

```
$ aws kms replicate-key \
    --key-id arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab \
    --replica-region ap-southeast-2
{
    "ReplicaKeyMetadata": {
        "MultiRegion": true,
        "MultiRegionConfiguration": {
            "MultiRegionKeyType": "REPLICA",
            "PrimaryKey": {
                "Arn": "arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
                "Region": "us-east-1"
            },
            "ReplicaKeys": [
                {
                    "Arn": "arn:aws:kms:ap-southeast-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
                    "Region": "ap-southeast-2"
                },
                {
                    "Arn": "arn:aws:kms:eu-west-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
                    "Region": "eu-west-1"
                }
            ]
        },
        "AWSAccountId": "111122223333",
        "Arn": "arn:aws:kms:ap-southeast-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
        "CreationDate": 1607472987.918,
        "Description": "",
        "Enabled": true,
        "KeyId": "mrk-1234abcd12ab34cd56ef1234567890ab",
        "KeyManager": "CUSTOMER",
        "KeySpec": "SYMMETRIC_DEFAULT",
        "KeyState": "Enabled",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "Origin": "AWS_KMS",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ]
    },
    "ReplicaKeyPolicy": "{\n  \"Version\" : \"2012-10-17\",\n  \"Id\" : \"key-
default-1\",...,
    "ReplicaTags": []
}
```

Viewing multi-Region keys

You can view single-Region and multi-Region keys in the AWS KMS console and by using the AWS KMS API operations.

Topics

- Viewing multi-Region keys in the console
- Viewing multi-Region keys in the API

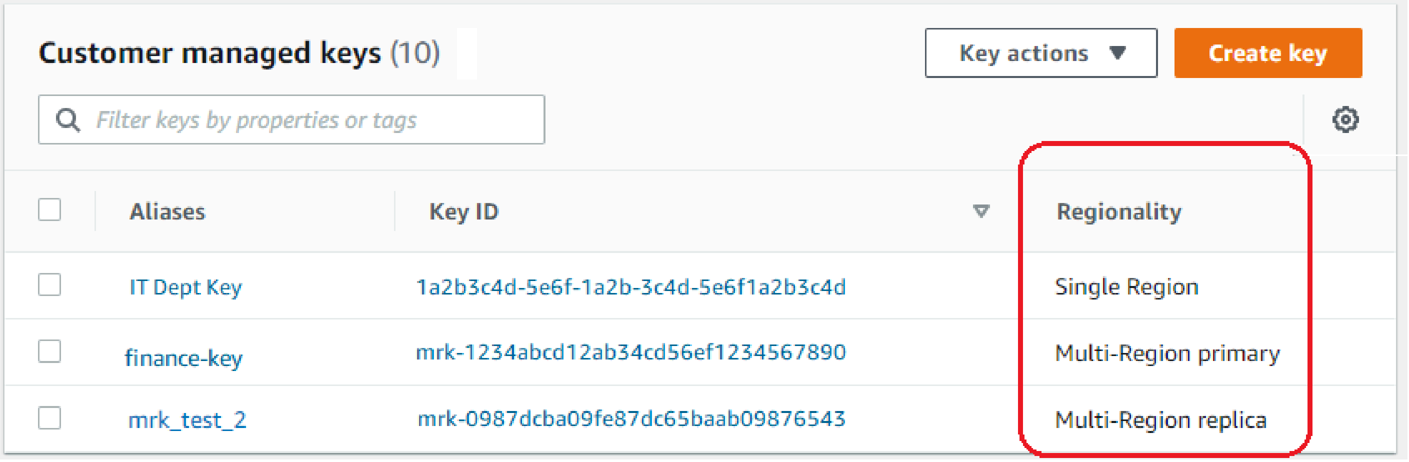
Viewing multi-Region keys in the console

In the AWS KMS console, you can view KMS keys in the selected Region. However, if you have a multi-Region key, you can see its related multi-Region keys in other AWS Regions.

The **Customer managed keys** table in the AWS KMS console displays only KMS keys in the selected Region. You can view multi-Region primary and replica keys in the selected Region. To change the AWS Region, use the Region selector in the upper-right corner of the page.

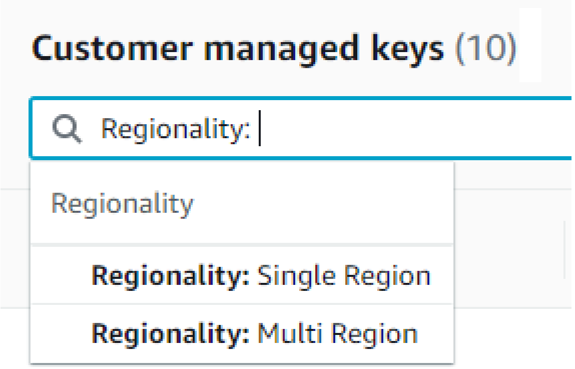
The AWS managed keys table does not have the regionality features because AWS managed keys are always single-Region keys.

- To make it easy to identify your multi-Region keys, add the **Regionality** column to your key table. For help, see Customizing your KMS key tables.



□	Aliases	Key ID	Regionality
□	IT Dept Key	1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d	Single Region
□	finance-key	mrk-1234abcd12ab34cd56ef1234567890	Multi-Region primary
□	mrk_test_2	mrk-0987dcba09fe87dc65baab09876543	Multi-Region replica

- To display only single-Region keys or only multi-Region keys in your key table, filter your keys by the **Regionality** property of each key. For help, see Sorting and filtering your KMS keys.



Customer managed keys (10)

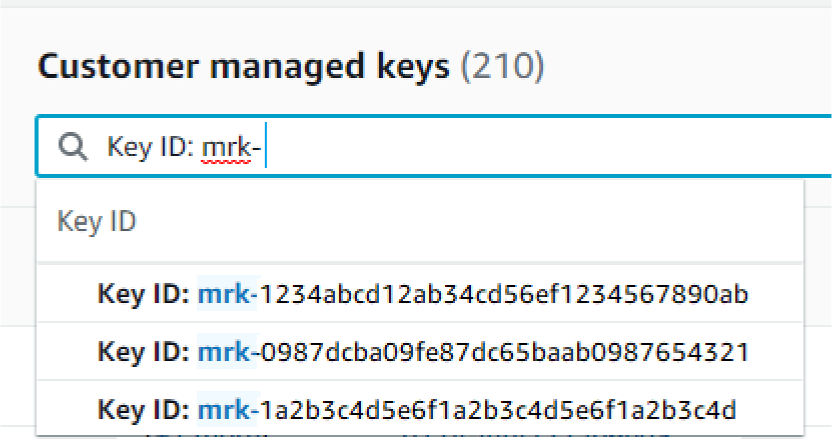
Regionality:

Regionality

Regionality: Single Region

Regionality: Multi Region

- You can also sort and filter your **Customer managed keys** table for the distinctive **mrk-** key ID prefix.



Customer managed keys (210)

Key ID: mrk-

Key ID

Key ID: mrk-1234abcd12ab34cd56ef1234567890ab

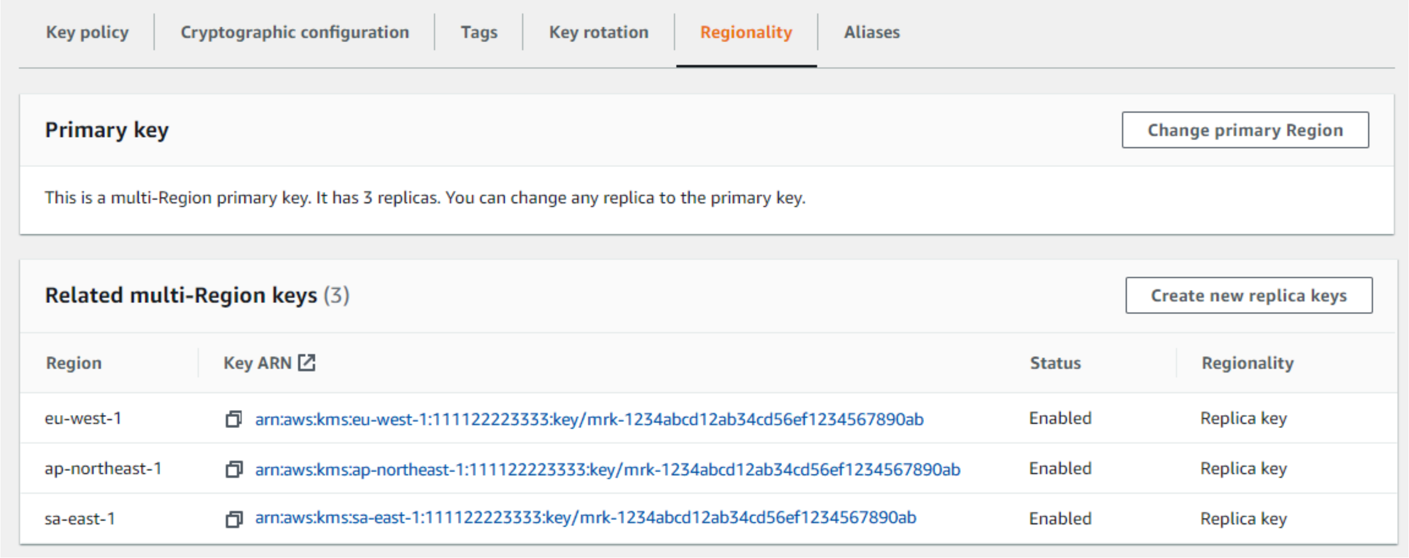
Key ID: mrk-0987dcba09fe87dc65baab0987654321

Key ID: mrk-1a2b3c4d5e6f1a2b3c4d5e6f1a2b3c4d

- For details about a multi-Region primary key or replica key, go to the detail page for the key, and choose the **Regionality** tab.

  The **Regionality** tab for a primary key includes Change primary Region and Create new replica keys buttons. (The Regionality tab for a replica key has neither button.) The **Related multi-Region keys** section lists all multi-Region keys related to the current one. If the current key is a replica key, this list includes the primary key.

  If you choose a related multi-Region key from the **Related multi-Region keys** table, the AWS KMS console changes to the Region of the selected key and it opens the detail page for the key. For example, if you choose the replica key in the `sa-east-1` Region from the example **Related multi-Region keys** section below, the AWS KMS console changes to the `sa-east-1` Region to display the detail page for that replica key. You might do this to view the alias or key policy for the replica key. To change the Region again, use the Region selector at the top right corner of the page.



Primary key

This is a multi-Region primary key. It has 3 replicas. You can change any replica to the primary key.

Related multi-Region keys (3)

Region	Key ARN	Status	Regionality
eu-west-1	arn:aws:kms:eu-west-1:111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab	Enabled	Replica key
ap-northeast-1	arn:aws:kms:ap-northeast-1:111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab	Enabled	Replica key
sa-east-1	arn:aws:kms:sa-east-1:111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab	Enabled	Replica key

Viewing multi-Region keys in the API

To view multi-Region keys in the AWS KMS API, use the DescribeKey operation. It displays the specified key and all of its related multi-Region keys.

Like the AWS KMS console, AWS KMS API operations are Regional. For example, when you call the ListKeys or ListAliases operations, they return only the resources in the current or specified Region. But when you call the `DescribeKey` operation on a multi-Region key, the response includes all related multi-Region keys in other AWS Regions.

For example, the following `DescribeKey` request gets details about an example multi-Region replica key in the Asia Pacific (Tokyo) (`ap-northeast-1`) Region.

```
$ aws kms describe-key \
        --key-id arn:aws:kms:ap-northeast-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab \
        --region ap-northeast-1
```

Most of the `KeyMetadata` in the response describes the replica key in the Asia Pacific (Tokyo) Region that's the subject of the request. However, the `MultiRegionConfiguration` element describes the primary key in the US West (Oregon) (us-west-2) Region and its replica keys in other AWS Regions, including the replica in the Asia Pacific (Tokyo) Region. `DescribeKey` returns the same `MultiRegionConfiguration` value for all related multi-Region keys.

```
{
    "KeyMetadata": {
        "MultiRegion": true,
        "AWSAccountId": "111122223333",
        "Arn": "arn:aws:kms:ap-northeast-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
        "CreationDate": 1586329200.918,
        "Description": "",
        "Enabled": true,
        "KeyId": "mrk-1234abcd12ab34cd56ef1234567890ab",
        "KeyManager": "CUSTOMER",
        "KeyState": "Enabled",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "Origin": "AWS_KMS",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ],
        "MultiRegionConfiguration": {
            "MultiRegionKeyType": "PRIMARY",
            "PrimaryKey": {
                "Arn": "arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
                "Region": "us-west-2"
            },
            "ReplicaKeys": [
                {
                    "Arn": "arn:aws:kms:eu-west-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
                    "Region": "eu-west-1"
                },
                {
                    "Arn": "arn:aws:kms:ap-northeast-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
                    "Region": "ap-northeast-1"
                },
                {
                    "Arn": "arn:aws:kms:sa-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
                    "Region": "sa-east-1"
                }
            ]
        }
    }
```

```
}
```

Managing multi-Region keys

For most actions, you manage multi-Region keys in the same way that you use and manage single-Region keys. You can enable and disable the keys, set and update aliases, key policies, grants, and tags. However, management of multi-Region keys differs in the following ways.

- You can update the primary Region. This changes one of the replica keys to a primary key and the current primary key to a replica.
- You manage automatic key rotation only on the primary key.
- You can get the public key for an asymmetric multi-Region key from any of the related primary or replica keys.

The multi-Region property that you set when you create KMS key is immutable. You cannot convert a single-Region key to multi-Region key or a convert a multi-Region key to a single-Region key.

Updating the primary Region

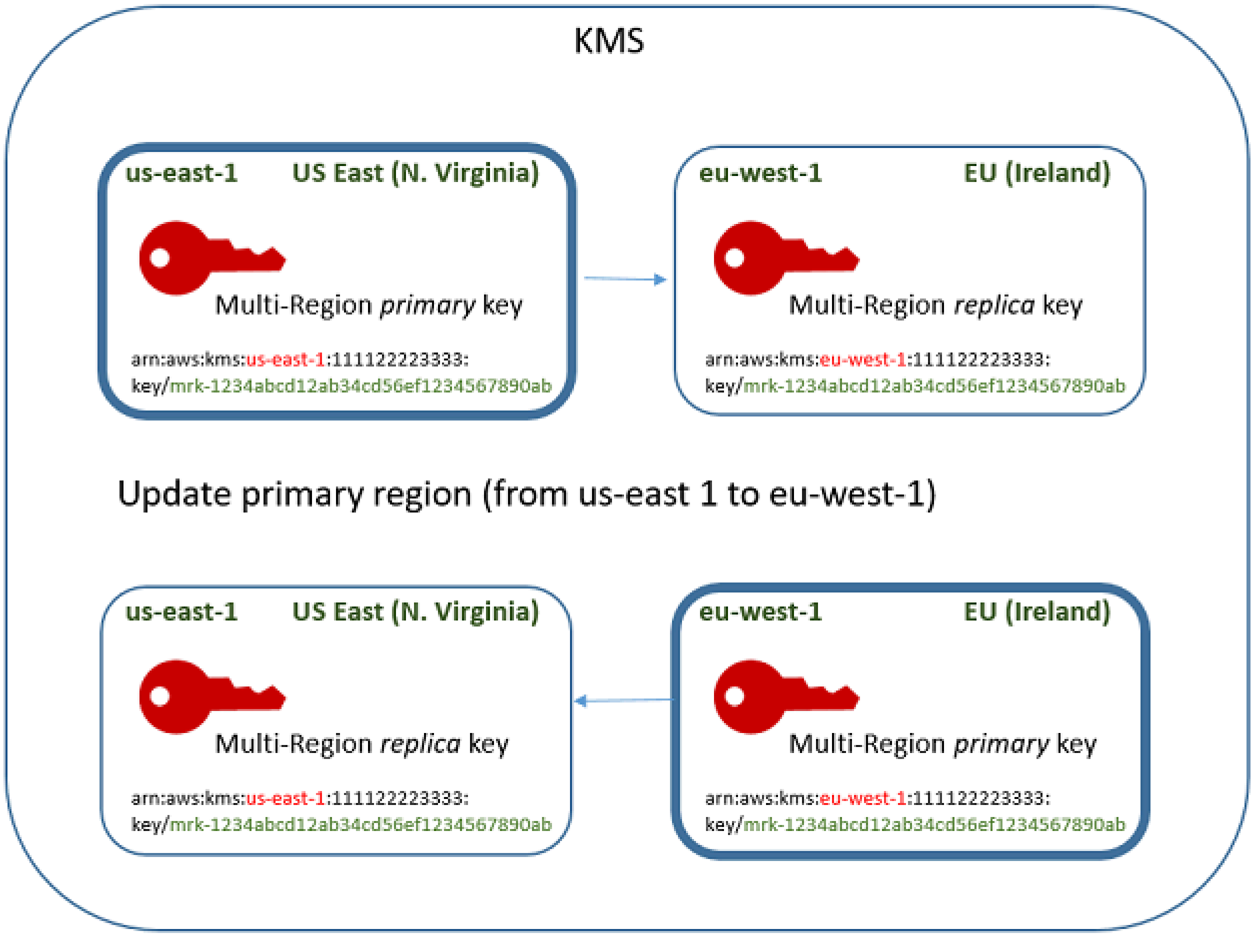
Every set of related multi-Region keys must have a primary key. But you can change the primary key. This action, known as *updating the primary Region*, converts the current primary key to a replica key and converts one of the related replica keys to the primary key. You might do this if you need to delete the current primary key while maintaining the replica keys, or to locate the primary key in the same Region as your key administrators.

You can select any related replica key to be the new primary key. Both the primary key and the replica key must be in the `Enabled` key state when the operation starts.

Even after this operation completes, the process of updating the primary Region might still be in progress for a few more seconds. During this time, the old and new primary keys have a transient key state of Updating. While the key state is `Updating`, you can use the keys in cryptographic operations, but you cannot replicate the new primary key or perform certain management operations, such as enabling or disabling these keys. Operations such as DescribeKey might display both the old and new primary keys as replicas. The `Enabled` key state is restored when the update is complete.

Suppose you have a primary key in US East (N. Virginia) (us-east-1) and a replica key in Europe (Ireland) (eu-west-1). You can use the update feature to change the primary key in US East (N.

Virginia) (us-east-1) to a replica key and change the replica key in Europe (Ireland) (eu-west-1) to the primary key.


KMS
us-east-1
US East (N. Virginia)
Multi-Region *primary* key
arn:aws:kms:us-east-1:111122223333:
key/mrk-1234abcd12ab34cd56ef1234567890ab
eu-west-1
EU (Ireland)
Multi-Region *replica* key
arn:aws:kms:eu-west-1:111122223333:
key/mrk-1234abcd12ab34cd56ef1234567890ab
Update primary region (from us-east 1 to eu-west-1)
us-east-1
US East (N. Virginia)
Multi-Region *replica* key
arn:aws:kms:us-east-1:111122223333:
key/mrk-1234abcd12ab34cd56ef1234567890ab
eu-west-1
EU (Ireland)
Multi-Region *primary* key
arn:aws:kms:eu-west-1:111122223333:
key/mrk-1234abcd12ab34cd56ef1234567890ab

When the update process completes, the multi-Region key in the Europe (Ireland) (eu-west-1) Region is a multi-Region primary key and the key in the US East (N. Virginia) (us-east-1) Region is its replica key. If there are other related replica keys, they become replicas of the new primary key. The next time that AWS KMS synchronizes the shared properties of the multi-Region keys, it will get the shared properties from the new primary key and copy them to its replica keys, including the former primary key.

The update operation has no effect on the key ARN of any multi-Region key. It also has no effect on shared properties, such as the key material, or on independent properties, such as the key policy. However, you might want to update the key policy of the new primary key. For example, you might want to add kms:ReplicateKey permission for trusted principals to the new primary key and remove it from the new replica key.

The `Updating` key state

The process of updating a primary Region takes a bit longer than the brief eventual consistency delay that affects most AWS KMS operations. The process might still be in progress after the `UpdatePrimaryRegion` operation returns or you've completed the update procedure in the console. Operations such as DescribeKey might display both the old and new primary keys as replicas until the process completes.

During the process of updating the primary Region, the old primary key and new primary key are in the `Updating` key state. When the update process completes successfully, both keys return to the `Enabled` key state. While in the `Updating` state, some management operations, such as enabling and disabling the keys, are not available. However, you can continue to use both keys in cryptographic operations without interruption. For information about the effect of the `Updating` key state, see Key states of AWS KMS keys.

Updating a primary Region (console)

You can update the primary key in the AWS KMS console. Start on the key details page for the current primary key.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Select the key ID or alias of the multi-Region primary key. This opens the key details page for the primary key.

   To identify a multi-Region primary key, use the tool icon in the upper right corner to add the **Regionality** column to the table.
5. Choose the **Regionality** tab.
6. In the **Primary key** section, choose **Change primary Region**.
7. Choose the Region of the new primary key. You can choose only one Region from the menu.

   The **Change primary Regions** menu includes only Regions that have a related multi-Region key. You might not have permission to update the primary Region in all of the Regions on the menu.
8. Choose **Change primary Region**.

Updating a primary Region (AWS KMS API)

To change the primary key in a set of related multi-Region keys, use the UpdatePrimaryRegion operation.

Use the `KeyId` parameter to identify the current primary key. Use the `PrimaryRegion` parameter to indicate the AWS Region of the new primary key. If the primary key doesn't already have a replica in the new primary Region, the operation fails.

The following example changes the primary key from the multi-Region key in the `us-west-2` Region to its replica in the `eu-west-1` Region. The `KeyId` parameter identifies the current primary key in the `us-west-2` Region. The `PrimaryRegion` parameter specifies the AWS Region of the new primary key, `eu-west-1`.

```
$ aws kms update-primary-region \
      --key-id arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab \
      --primary-region eu-west-1
```

When successful, this operation doesn't return any output; just the HTTP status code. To see the effect, call the DescribeKey operation on either of the multi-Region keys. You might want to wait until the key state returns to `Enabled`. While the key state is Updating, the values for the key might still be in flux.

For example, the following `DescribeKey` call gets the details about the multi-Region key in the `eu-west-1` Region. The output shows that the multi-Region key in the `eu-west-1` Region is now the primary key. The related multi-Region key (same key ID) in the `us-west-2` Region is now a replica key.

```
$ aws kms describe-key \
      --key-id arn:aws:kms:eu-west-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab \

{
    "KeyMetadata": {
        "AWSAccountId": "111122223333",
        "KeyId": "mrk-1234abcd12ab34cd56ef1234567890ab",
        "Arn": "arn:aws:kms:eu-west-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
        "CreationDate": 1609193147.831,
        "Enabled": true,
        "Description": "multi-region-key",
        "KeySpec": "SYMMETRIC_DEFAULT",
        "KeyState": "Enabled",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "Origin": "AWS_KMS",
        "KeyManager": "CUSTOMER",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ],
        "MultiRegion": true,
        "MultiRegionConfiguration": {
           "MultiRegionKeyType": "PRIMARY",
           "PrimaryKey": {
              "Arn": "arn:aws:kms:eu-west-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
              "Region": "eu-west-1"
           },
           "ReplicaKeys": [
              {
                 "Arn": "arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
                 "Region": "us-west-2"
              }
           ]
        }
    }
}
```

Rotating multi-Region keys

You can enable and disable automatic rotation and perform on-demand rotation of the key material in multi-Region keys. Key rotation is a shared property of multi-Region keys.

You enable and disable automatic key rotation only on the primary key. You initiate on-demand rotation only on the primary key.

- When AWS KMS synchronizes the multi-Region keys, it copies the key rotation property setting from the primary key to all of its related replica keys.
- When AWS KMS rotates the key material, it creates new key material for the primary key and then copies the new key material across Region boundaries to all related replica keys. The key

material never leaves AWS KMS unencrypted. This step is carefully controlled to ensure that key material is fully synchronized before any key is used in a cryptographic operation.

- AWS KMS does not encrypt any data with the new key material until that key material is available in the primary key and every one of its replica keys.
- When you replicate a primary key that has been rotated, the new replica key has the current key material and all previous versions of the key material for its related multi-Region keys.

This pattern ensures that related multi-Region keys are fully interoperable. Any multi-Region key can decrypt any ciphertext encrypted by a related multi-Region key, even if the ciphertext was encrypted before the key was created.

Automatic key rotation is not supported on asymmetric KMS keys or KMS keys with imported key material. For information about automatic and on-demand key rotation, see Rotating AWS KMS keys.

Downloading public keys

When you create a multi-Region asymmetric KMS key, AWS KMS creates an RSA or elliptic curve (ECC) key pair for the primary key. Then it copies that key pair to every replica of the primary key. As a result, you can download the public key from the primary key or any of its replica keys. You will always get the same key material.

For information about downloading and using public keys outside of AWS KMS, see Special considerations for downloading public keys. For instructions, see Downloading public keys.

Importing key material into multi-Region keys

You can import your own key material into a multi-Region KMS key. The multi-Region keys you create with your own key material are interoperable. You can encrypt data in one Region and decrypt it in any other Region with a related multi-Region key.

However, you must manage the key material.

- AWS KMS does not copy or synchronize the key material from a primary key with imported key material to its replica keys. You must import the same key material into related primary and replica keys.
- You set the expiration model and expiration dates for each key independently when you import the key material. You can configure the same or a different expiration model and expiration

dates for related multi-Region keys. If the key material approaches its expiration date, you must reimport the key material into the affected multi-Region key.

The key states of related multi-Region keys are independent of each other. For example, if the key material in the primary key expires, its replica keys are unaffected.

The same Region requirements for replica keys apply to multi-Region keys with imported key material. If you import the same key material into single-Region keys or unrelated multi-Region keys, these KMS keys are not interoperable.

You can create Multi-Region keys with imported symmetric, asymmetric, or HMAC key material. AWS KMS does not support imported key material in custom key stores. Also, you cannot enable automatic key rotation of any KMS key with imported key material.

Aside from their multi-Region features, multi-Region keys with imported key material are the same as other KMS keys with imported key material. For detailed information about creating and configuring single-Region keys with imported key material, see About imported key material.

Topics

Why aren't all KMS keys with imported key material interoperable?

Single-region KMS keys with imported key material are not interoperable, even when they have the same key material. When AWS KMS uses a KMS key to encrypt data, it cryptographically binds some of the key metadata to the ciphertext. This secures the ciphertext so that only the KMS key that encrypted data can decrypt that data.

Multi-Region keys are designed to be interoperable. In addition to having the same key material, they have the same key ID and other metadata. Thus, the ciphertexts they generate can be decrypted by any related multi-Region key. As a result, the trust properties of multi-Region keys are different than those of single-Region keys. But for some customers, the benefit of decrypting in multiple Regions outweighs the security value of a ciphertext reliant on a single KMS key in a single AWS Region.

Creating a primary key with imported key material

To create a primary key with imported key material, you start by creating a KMS key key with no key material. When you create the primary key with no key material, you must specify the key spec that reflects the type of key material you plan to import. Then, import your key material into the primary key.

The procedure for creating a multi-Region primary key with no key material is almost the same as the procedure for creating a single-Region key with no key material. The only difference is that you specify that the key is a multi-Region key.

The permissions for creating a multi-Region primary key with imported key material are the same as those required to create a multi-Region primary key with AWS KMS key material, including the kms:CreateKey and iam:CreateServiceLinkedRole permissions in an IAM policy. You can use the kms:MultiRegionKeyType and kms:KeyOrigin condition keys to allow or deny permission to create multi-Region primary keys with imported key material.

When creating a primary key with imported key material in the AWS KMS console, use the settings in the **Advanced options** section. You cannot change these properties after the KMS key is created.

- Set **Key material origin** to **External (Import key material)**.
- Set **Multi-Region replication** to **Allow this key to be replicated into other Regions**.

When using the CreateKey operation to create a primary key with imported key material, use the `Origin` and `MultiRegion` parameters and specify the `KeySpec` and the `KeyUsage`. The following example creates an `EXTERNAL` KMS key that can import `ECC_NIST_P384` key material.

```
$ aws kms create-key --origin EXTERNAL --key-spec ECC_NIST_P384 --key-usage SIGN_VERIFY
 --multi-region
```

The result is a multi-Region primary key with no key material and a key state of `PendingImport`.

To enable this KMS key, you must download a public key and import token, use the public key to encrypt your key material, and then import your key material. For instructions, see Importing key material for AWS KMS keys.

Creating a replica key with imported key material

You can create a multi-Region replica key in the AWS KMS console or by using the AWS KMS API operations. To replicate a multi-Region primary key with imported key material, you use the same procedure that you use to create a replica key with AWS KMS key material. However, the result is different. Instead of returning a replica key with the same key material as the primary key, the replicate process returns a replica key with no key material and a key state of `PendingImport`. To enable the replica key, you must import the same key material into the replica key that you imported into its primary key.

Although it doesn't replicate the key material, AWS KMS creates the replica key with the same key ID, key spec, key usage, and key material origin as the primary key. It also ensures that the key material that you import into the replica key is identical to the key material that you imported into the primary key.

To create a replica key with imported key material:

1. Create a multi-Region primary key with imported key material.
2. Do one of the following.

   In the AWS KMS console, choose a multi-Region primary key with imported key material. Then, on its **Regionality** tab, choose **Create new replica keys**. For instructions, see Creating replica keys (console).

   Or use the ReplicateKey operation. For the `KeyId` parameter, enter the key ID or key ARN of a multi-Region primary key with imported key material. For instructions, see Creating a replica key (AWS KMS API).
3. For each new replica key, follow the steps to download a public key and import token. Use the public key to encrypt the primary key's key material, and then import the primary key's key material in the replica key. You need a different public key and import token for each replica key.

   If the key material that you try to import into the replica key isn't the same the key material as its primary key, the operation fails. AWS KMS doesn't require that the expiration model and expiration dates be coordinated, but you might establish business rules for your multi-Region keys. For instructions, see Importing key material for AWS KMS keys.

Permissions to replicate keys with imported key materials

To create a replica key with imported key material, you must have the following permissions.

In the primary key Region:

- kms:ReplicateKey on the primary key (in the primary key's Region). Include this permission in the primary key's key policy or in an IAM policy.

In the replica key Region:

- kms:CreateKey in an IAM policy.
- kms:GetParametersForImport. You can include this permission in the key policy of the replica key or in an IAM policy.
- kms:ImportKeyMaterial. You can include this permission in the key policy of the replica key or in an IAM policy.
- kms:TagResource is required to assign tags when replicating. Include this permission in an IAM policy in the replica Region.
- kms:CreateAlias is required to replicate a key in the AWS KMS console. For details, see Controlling access to aliases.

Deleting multi-Region keys

When you are no longer using a multi-Region primary key or replica key, you can schedule its deletion.

Although deleting KMS keys should always be done with caution, deleting a replica of a multi-Region key is less risky, provided that the primary key still exists in AWS KMS. If you delete a replica key from its Region, but discover ciphertext that was encrypted under the deleted key, you can decrypt that ciphertext with any related multi-Region key. You can also recreate the replica key by replicating the primary key again into the replica key Region.

However, deleting a primary key and all of its replica key is a very dangerous operation — equivalent to deleting a single-Region key.

Warning

Deleting a KMS key is destructive and potentially dangerous. You should proceed only when you are sure that you don't need to use the KMS key anymore and won't need to use it in the future. If you are not sure, you should disable the KMS key instead of deleting it.

To delete a primary key, you must first delete all of its replica keys. If you must delete a primary key from a particular Region without deleting its replica keys, change the primary key to a replica key by updating the primary Region.

Before you schedule the deletion of any KMS key, review the cautions in the Deleting AWS KMS keys topic, and the topics that explain how to determine past use of a KMS key and how to set a CloudWatch alarm that alerts you to use of the KMS key during the waiting period. Before deleting the primary key of an asymmetric multi-Region key, review the Deleting asymmetric keys topic.

Topics

- Permissions for deleting multi-Region keys
- How to delete a replica key
- How to delete a primary key

Permissions for deleting multi-Region keys

To schedule the deletion of a multi-Region key, you need only the following permission.

- kms:ScheduleKeyDeletion — to schedule the deletion of the multi-Region key and set its waiting period.

We also strongly recommend that you have the following related permissions.

- kms:CancelKeyDeletion — to cancel the scheduled deletion of the multi-Region key.
- kms:DescribeKey — to view the key state of the multi-Region key and the list of related multi-Region keys.
- kms:DisableKey — to give you the option to disable a multi-Region key instead of deleting it.
- kms:EnableKey — to restore the functionality of a multi-Region key after canceling its deletion.

You might also include permission to replicate the primary key and change the primary key.

- kms:ReplicateKey
- kms:UpdateReplicaRegion

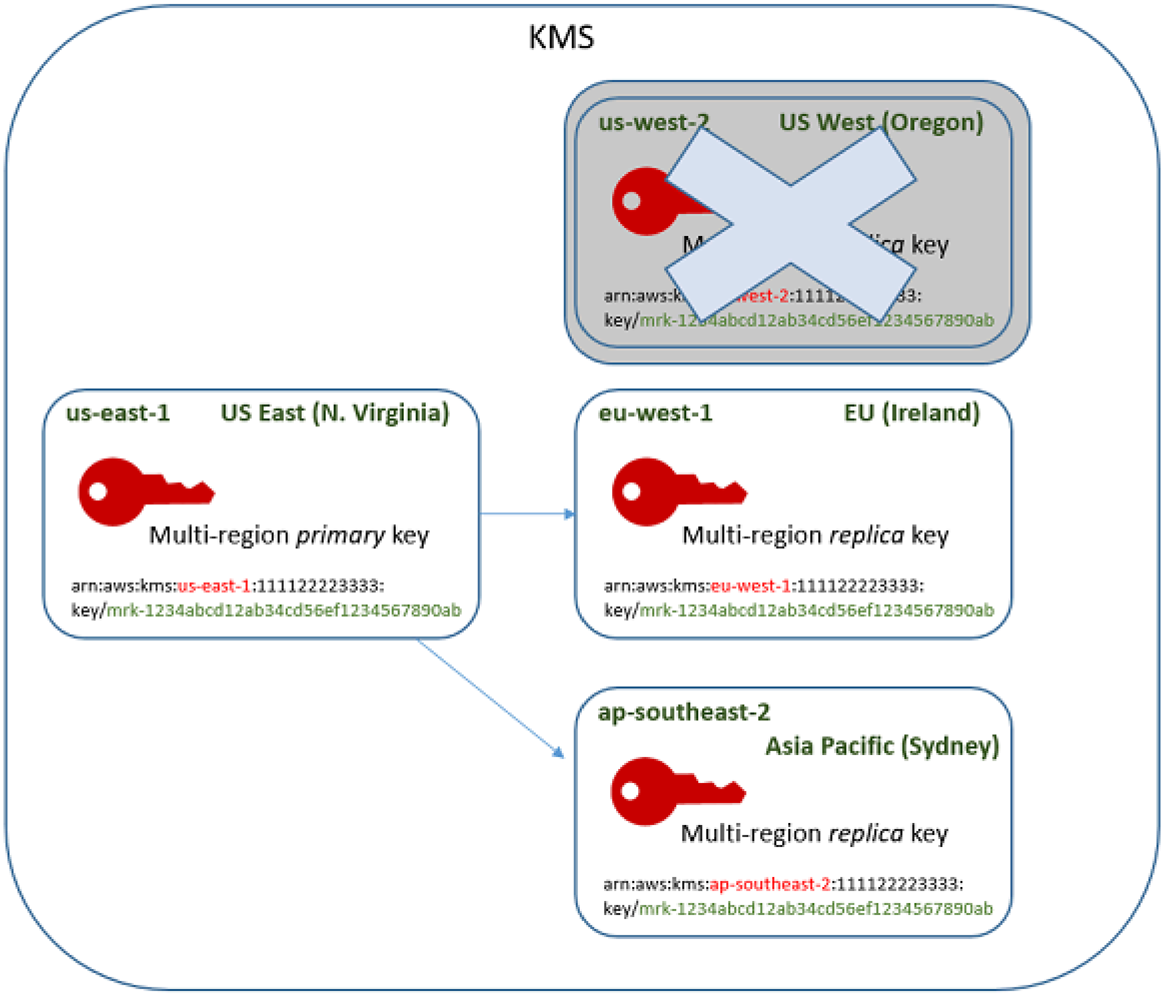
You can include these permissions in an IAM policy, but it's a best practice to put them in a key policy where they apply only to the KMS key that you need to manage.

How to delete a replica key

You can use the AWS KMS console or the AWS KMS API to delete a replica key. You can delete a replica key at any time. It doesn't depend on the key state of any other KMS key.

If you mistakenly delete a replica key, you can recreate it by replicating the same primary key in the same Region. The new replica key you create will have the same shared properties as the original replica key.

The procedure for deleting a multi-Region replica key is the same as deleting a single-Region key.


KMS
us-west-2
US West (Oregon)
arn:aws:k…west-2:1111…33:
key/mrk-12…4abcd12ab34cd56ef…234567890ab
us-east-1
US East (N. Virginia)
Multi-region *primary* key
arn:aws:kms:us-east-1:111122223333:
key/mrk-1234abcd12ab34cd56ef1234567890ab
eu-west-1
EU (Ireland)
Multi-region *replica* key
arn:aws:kms:eu-west-1:111122223333:
key/mrk-1234abcd12ab34cd56ef1234567890ab
ap-southeast-2
Asia Pacific (Sydney)
Multi-region *replica* key
arn:aws:kms:ap-southeast-2:111122223333:
key/mrk-1234abcd12ab34cd56ef1234567890ab

1. Schedule deletion of the replica key. Select a waiting period of 7-30 days. The default waiting period is 30 days.
2. During the waiting period, the key state of the replica key changes to `Pending deletion` (`PendingDeletion`) and you cannot use it in cryptographic operations.

3. You can cancel the scheduled deletion of the replica key at any point in the waiting period. The key state changes to `Disabled`, but you can re-enable the KMS key.
4. When the waiting period expires, AWS KMS deletes the replica key.

You can view a record of your actions in your AWS CloudTrail log. AWS KMS records the operations that schedule deletion of the KMS key and the action that deletes the KMS key.

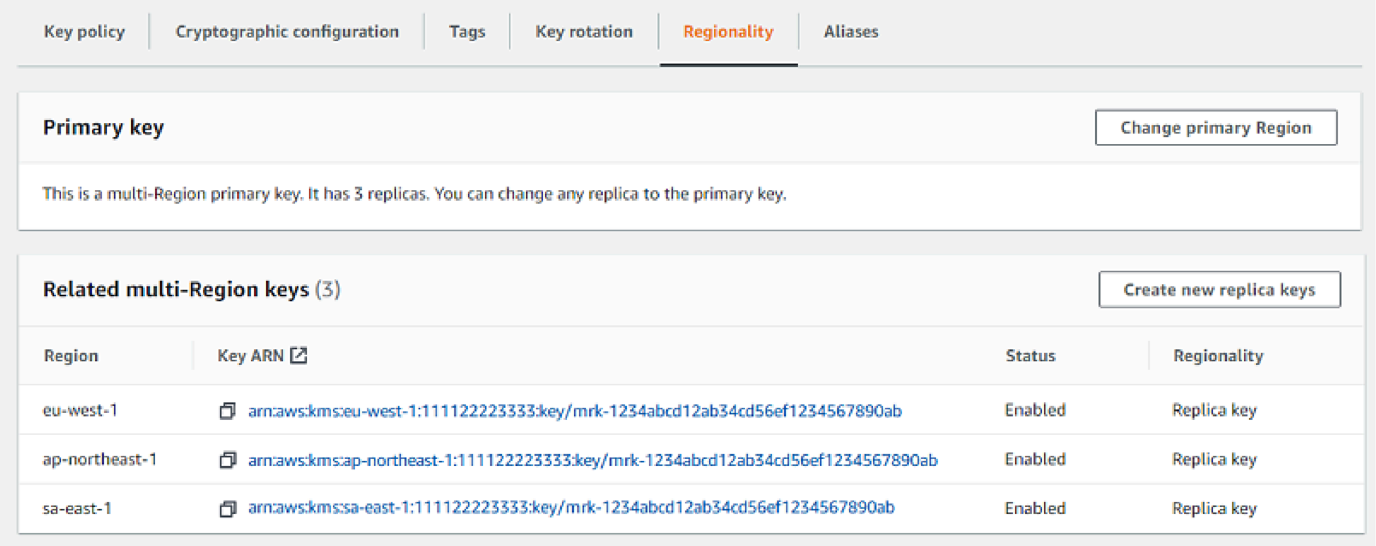
Deleting a replica key (console)

To schedule the deletion of a multi-Region replica key, use the same procedure you use to schedule the deletion of a single-Region key.

Because related replica keys are in different AWS Regions, you cannot schedule the deletion of more than one replica key at a time. To delete all related replica keys, use a pattern like the following one.

To schedule deletion of all related replica keys

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. In the navigation pane, choose **Customer managed keys**.
3. Use the Region selector in the upper-right corner to choose the Region of the multi-Region primary key.
4. Choose its alias or key ID of the primary key.
5. Choose the **Regionality** tab.



Primary key

This is a multi-Region primary key. It has 3 replicas. You can change any replica to the primary key.

Related multi-Region keys (3)

Region	Key ARN	Status	Regionality
eu-west-1	arn:aws:kms:eu-west-1:111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab	Enabled	Replica key
ap-northeast-1	arn:aws:kms:ap-northeast-1:111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab	Enabled	Replica key
sa-east-1	arn:aws:kms:sa-east-1:111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab	Enabled	Replica key

6. In the **Related multi-Region keys** section, choose the key ARN of a replica key.

   This action opens the key details page of the replica key in a new browser tab. The console is set to the replica key Region.
7. From the **Key actions** menu, choose **Schedule key deletion**.

   This action starts the process of scheduling deletion of the key. Complete the schedule key deletion process. For details, see Scheduling and canceling key deletion (console).
8. Return to the browser tab that displays the **Regionality** tab of the primary key. (You might need to refresh the page to see the updated status of the replica keys.) Choose the key ARN of another replica key and repeat the process of scheduling deletion of the replica key.

Deleting a replica key (AWS KMS API)

To schedule the deletion of a multi-Region replica key, use the ScheduleKeyDeletion operation. To specify the KMS key, use its key ID or key ARN. When working with multi-Region keys, you can reduce the incidence of errors by using the key ARN with its explicit Region value.

For example, this command deletes a replica key from the us-west-2 (US West (Oregon)) Region. Because the command doesn't specify a waiting period, the waiting period is set to the default of 30 days.

```
$ aws kms schedule-key-deletion \
    --region us-west-2 \
    --key-id arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab
```

When the command succeeds, it returns the key ARN (`KeyId`), the waiting period (`PendingWindowInDays`), the deletion date (`DeletionDate`), and the current key state (`KeyState`), which is expected to be `PendingDeletion`.

When deleting a multi-Region replica key, be sure to verify that the key ID and Region values in the key ARN are the ones that you expect.

```
{
    "KeyId": "arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
    "DeletionDate": 1599523200.0,
    "KeyState": "PendingDeletion",
    "PendingWindowInDays": 30
}
```

To delete all replicas of a multi-Region primary key programmatically, create a list of the Regions that contain replica keys. Then, for each Region in the list, call the `ScheduleKeyDeletion` operation, as shown above.

Unlike a single-Region key that is permanently deleted, you can restore a replica key by replicating the primary key into the Region where the deleted replica key was located.

To check the status of the replica key and view the primary key and replica keys of a multi-Region key, use the DescribeKey operation.

How to delete a primary key

You can schedule the deletion of a multi-Region primary key at any time. However, AWS KMS will not delete a multi-Region primary key that has replica keys, even if they are scheduled for deletion.

To delete a primary key, you must schedule the deletion all of its replica keys, and then wait for the replica keys to be deleted. The required waiting period for deleting a primary key begins when the last of its replica keys is deleted. If you must delete a primary key from a particular Region without deleting its replica keys, change the primary key to a replica key by updating the primary Region.

If a primary key has no replica keys, the process is identical to deleting a replica key or deleting any regional KMS key.

While a primary key is scheduled for deletion, you cannot use it in cryptographic operations and you cannot replicate it. However, unless they are also scheduled for deletion, its replica keys are unaffected.

You can use the AWS KMS console or the AWS KMS API to schedule the deletion of primary and replica keys. You can schedule deletion of the primary key before, after, or at the same time that you schedule deletion of the replica keys. The process might look something like the following one.

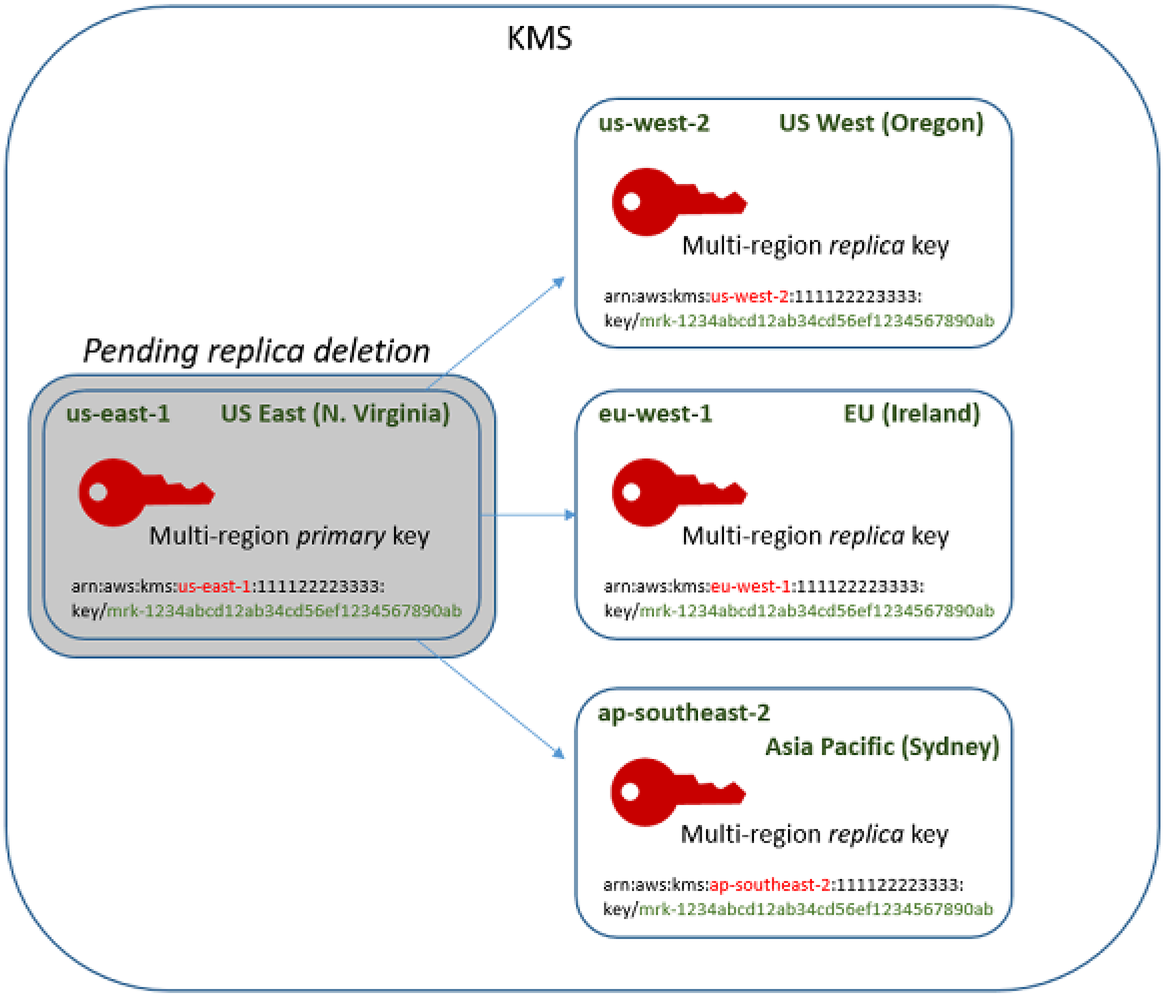
1. Schedule the deletion of the primary key. Select a waiting period of 7-30 days. The default waiting period is 30 days. However, the waiting period for the primary key does not begin until all replica keys are deleted.

   If any replica keys still exist, the key state of the primary key changes to `Pending replica deletion` (`PendingReplicaDeletion`). Otherwise, it changes to `Pending deletion`

(`PendingDeletion`). In either case, you cannot use the primary key in cryptographic operations and you cannot replicate it.

Scheduling the deletion of a primary key doesn't affect the replica keys. Their key state remains enabled and you can use them in cryptographic operations. If the replica keys are not deleted, the `Pending replica deletion` state of the primary key can persist indefinitely.

```
KMS key:                     Key state:
Primary (us-east-1)            Pending replica deletion (waiting period 30 days -- not
 started)
Replica (us-west-2)            Enabled
Replica (eu-west-1)            Enabled
Replica (ap-southeast-2)       Enabled
```


KMS
Pending replica deletion
us-east-1 US East (N. Virginia)
Multi-region primary key
arn:aws:kms:us-east-1:111122223333:
key/mrk-1234abcd12ab34cd56ef1234567890ab
us-west-2 US West (Oregon)
Multi-region replica key
arn:aws:kms:us-west-2:111122223333:
key/mrk-1234abcd12ab34cd56ef1234567890ab
eu-west-1 EU (Ireland)
Multi-region replica key
arn:aws:kms:eu-west-1:111122223333:
key/mrk-1234abcd12ab34cd56ef1234567890ab
ap-southeast-2 Asia Pacific (Sydney)
Multi-region replica key
arn:aws:kms:ap-southeast-2:111122223333:
key/mrk-1234abcd12ab34cd56ef1234567890ab

2. Schedule deletion of each replica key. Select a waiting period of 7-30 days. The default waiting period is 30 days. You can delete multiple replica keys at the same time. Their waiting periods

run concurrently. During the waiting period, the key state of the replica keys changes to `Pending deletion` (`PendingDeletion`) and you cannot use these KMS keys in cryptographic operations.

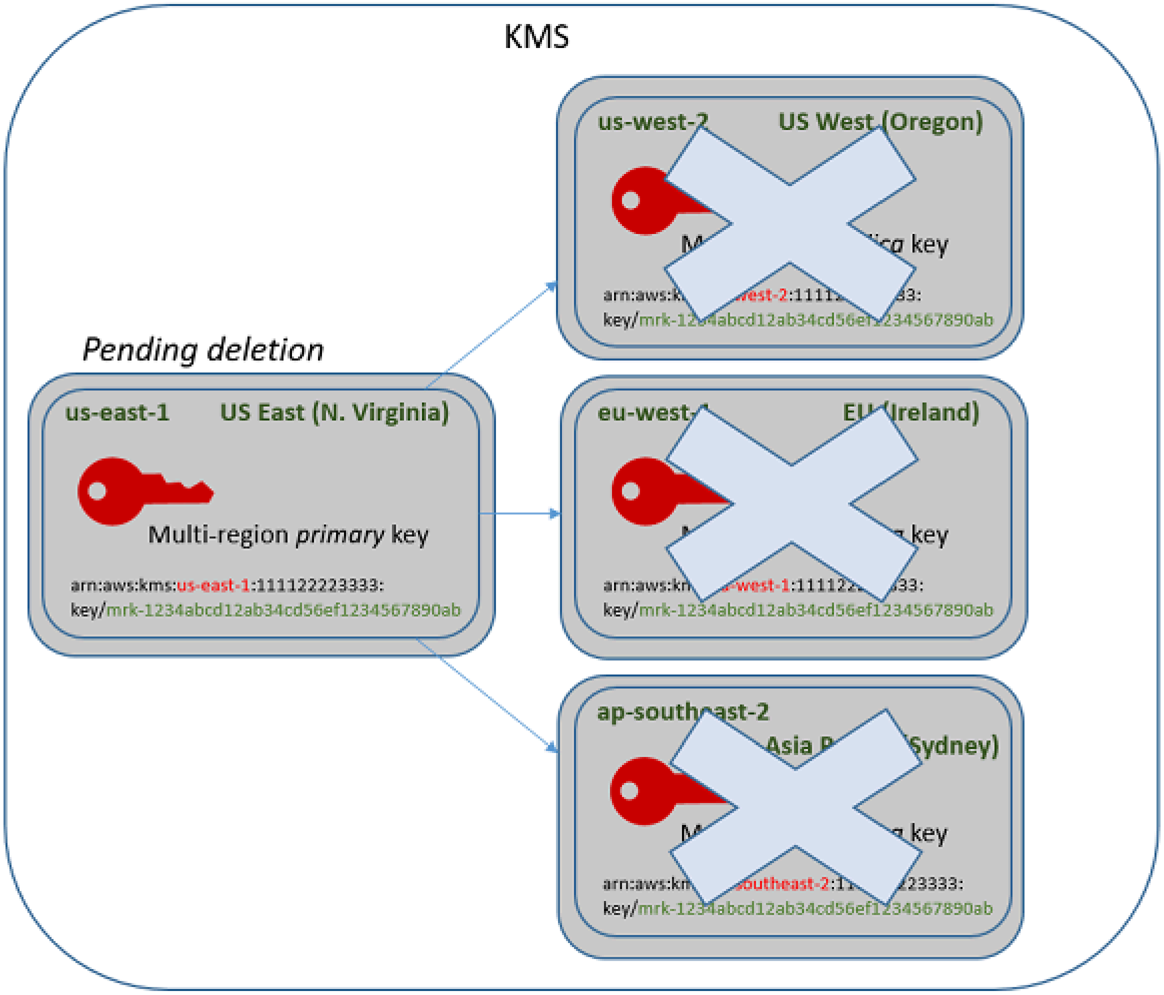
For example, if you have a three replica keys, you can schedule deletion of all three at the same time. They can have the same or different waiting periods. Notice that the waiting period on the primary key has not yet begun. Its key state is `PendingReplicaDeletion` because it has existing replica keys.

```
KMS key:                    Key state:
Primary key (us-east-1)       Pending replica deletion (waiting period 30 days -- not
 started)
Replica (us-west-2)           Pending deletion (7 days)
Replica (eu-west-1)           Pending deletion (7 days)
Replica (ap-southeast-2)      Pending deletion (30 days)
```

3. You can cancel the scheduled deletion of the primary key or any replica key until it is deleted. The key state changes to `Disabled`, but you can re-enable the KMS key.

4. When the waiting period of the last replica key expires, AWS KMS deletes the last replica key. The key state of the primary key changes from `Pending replica deletion` (`PendingReplicaDeletion`) to `Pending deletion` (`PendingDeletion`) and the 7-30 day waiting period for the primary key begins.

```
KMS key:                    Key state:
Primary key (us-east-1)       Pending deletion (waiting period 30 days)
```



5. When its waiting period expires, AWS KMS deletes the primary key.

The minimum time to delete a primary key with replicas is 14 days.

If you schedule key deletion of the primary key and all replica keys with a waiting period of 7 days, the replica keys are deleted after 7 days. The primary key is deleted on the 14th day.

- Day 1: Schedule the deletion of the primary and replica keys with the minimum waiting period of 7 days. The 7-day deletion waiting periods for the replica keys start. The deletion waiting period for the primary key does not yet start.
- Day 7: The deletion waiting periods for the replica keys end. AWS KMS deletes all replica keys. When the last replica key is deleted, the 7-day deletion waiting period for the primary key starts.
- Day 14: The deletion waiting period for the primary key ends. AWS KMS deletes the primary key.

You can view a record of your actions in your AWS CloudTrail log. AWS KMS records the operations that schedule deletion of each KMS key and the action that deletes the KMS key.

Deleting a primary key (console)

To delete a multi-Region primary key, use the following procedure.

To schedule key deletion

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Select the check box next to the primary key that you want to delete. You can also select one or more KMS keys, including the replicas of this primary key.
5. Choose **Key actions**, **Schedule key deletion**.
6. Read and consider the warning, and the information about canceling the deletion during the waiting period. If you decide to cancel the deletion, choose **Cancel**.
7. For **Waiting period (in days)**, enter a number of days between 7 and 30. If you selected multiple KMS keys, the waiting period that you choose applies to all selected KMS keys. The waiting period for replica keys runs concurrently, but the waiting period for the primary key does not begin until AWS KMS deletes the last of the replica keys.
8. Select the check box next to **Confirm that you want to delete this key in *`<number of days>`* days**.
9. Choose **Schedule deletion**.

To check the deletion status of your KMS keys, on the detail page for the primary key, see the **General configuration** section. The key state appears in the **Status** field. When the key state of the primary key changes to `Pending deletion` the **Scheduled deletion date** is displayed.

You can also check the key state (**Status**) of all primary and replica keys on the **Regionality** tab of the detail page for any multi-Region key. For details, see Viewing multi-Region keys.

Deleting a primary key (AWS KMS API)

To delete a multi-Region replica key, use the ScheduleKeyDeletion operation. To specify the KMS key, use its key ID or key ARN. When working with multi-Region keys, you can reduce the incidence of errors by using the key ARN with its explicit Region value.

For example, this command deletes a primary key from the us-east-1 (US East (N. Virginia)) Region. Because the command doesn't specify a waiting period, the waiting period is set to the default of 30 days.

```
$ aws kms schedule-key-deletion \
    --key-id arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab
```

When the command succeeds, it returns the key ARN, the resulting key state, and the waiting period (`PendingWindowInDays`).

If the primary key has no replicas, the key state of the primary key is `PendingDeletion` and the output includes the `DeletionDate` field. If any replica keys remain, the key state of the primary key is `PendingReplicaDeletion` and `DeletionDate` is omitted because it is uncertain. Even if the replica keys are also scheduled for deletion, you might cancel the scheduled deletion.

When deleting a multi-Region primary key, be sure to verify that the key ID and Region values in the key ARN are the ones that you expect.

```
{
    "KeyId": "arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
    "KeyState": "PendingReplicaDeletion",
    "PendingWindowInDays": 30
}
```

To check the deletion status of your KMS keys, use the DescribeKey operation on the primary key or any remaining replica keys. The waiting period clock for the primary key does not start until the last replica is deleted and the key state changes to `PendingDeletion`.

To calculate the expected deletion date of the primary key, loop through the replica key ARNs in the response, run `DescribeKey` on each one, get the latest `DeletionDate` value, and then add the `PendingDeletionWindowInDays` value for the primary key. The waiting periods for the replica keys run concurrently.

In the following example, the KMS key is a multi-Region primary key with existing replica keys. Because the key state is `PendingReplicaDeletion`, the response includes the waiting period (`PendingWindowInDays`), but not the `DeletionDate`. The actual deletion date of the primary key depends on when the replica keys are deleted.

```
$ aws kms describe-key \
    --key-id arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab

{
    "KeyMetadata": {
        "AWSAccountId": "111122223333",
        "KeyId": "mrk-1234abcd12ab34cd56ef1234567890ab",
        "Arn": "arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
        "CreationDate": 1597902361.481,
        "Enabled": false,
        "Description": "",
        "KeySpec": "SYMMETRIC_DEFAULT",
        "KeyState": "PendingReplicaDeletion",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "Origin": "AWS_KMS",
        "KeyManager": "CUSTOMER",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ],
        "MultiRegion": true,
        "MultiRegionConfiguration": {
            "MultiRegionKeyType": "PRIMARY",
            "PrimaryKey": {
                "Arn": "arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
                "Region": "us-east-1"
            },
            "ReplicaKeys": [
                {
                    "Arn": "arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
                    "Region": "us-west-2"
                },
{
                    "Arn": "arn:aws:kms:eu-west-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
                    "Region": "eu-west-1"
                },
                {
                    "Arn": "arn:aws:kms:ap-southeast-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
                    "Region": "ap-southeast-2"
                }
            ]
        },
        "PendingDeletionWindowInDays": 30
    }
}
```

When all replicas are deleted, the `DescribeKey` output shows the remaining primary key with a key state of `PendingDeletion`. While the key state is `PendingDeletion`, the `DeletionDate` field appears instead of the `PendingWindowInDays` field.

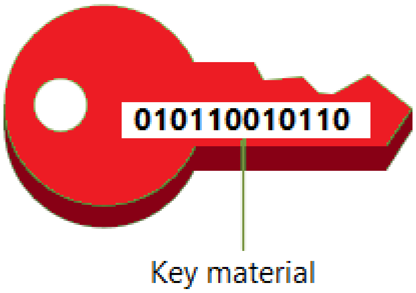
```
$ aws kms describe-key \
    --key-id arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab

{
    "KeyMetadata": {
        "AWSAccountId": "111122223333",
        "KeyId": "mrk-1234abcd12ab34cd56ef1234567890ab",
        "Arn": "",
        "CreationDate": 1597902361.481,
        "Enabled": false,
        "Description": "",
        "KeySpec": "SYMMETRIC_DEFAULT",
        "KeyState": "PendingDeletion",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "DeletionDate": 1597968000.0,
        "Origin": "AWS_KMS",
        "KeyManager": "CUSTOMER",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ],
        "MultiRegion": true,
        "MultiRegionConfiguration": {
            "MultiRegionKeyType": "PRIMARY",
            "PrimaryKey": {
                "Arn": "arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab",
                "Region": "us-east-1"
            },
            "ReplicaKeys": []
        }
    }
}
```

Importing key material for AWS KMS keys

You can create an AWS KMS keys (KMS key) with key material that you supply.

A KMS key is a logical representation of an encryption key. The metadata for a KMS key includes the ID of key material used to encrypt and decrypt data. When you create a KMS key, by default, AWS KMS generates the key material for that KMS key. But you can create a KMS key without key material and then import your own key material into that KMS key, a feature often known as "bring your own key" (BYOK).



> **Note**
>
> AWS KMS does not support decrypting any AWS KMS ciphertext outside of AWS KMS, even if the ciphertext was encrypted under a KMS key with imported key material. AWS KMS does not publish the ciphertext format this task requires, and the format might change without notice.

Imported key material is supported on all types of KMS keys except for KMS keys in custom key stores.

When you use imported key material, you remain responsible for the key material while allowing AWS KMS to use a copy of it. You might choose to do this for one or more of the following reasons:

- To prove the key material was generated using a source of entropy that meets your requirements.

- To use key material from your own infrastructure with AWS services, and to use AWS KMS to manage the lifecycle of that key material within AWS.
- To use existing, well-established keys in AWS KMS, such as keys for code signing, PKI certificate signing, and certificate pinned applications
- To set an expiration time for the key material in AWS and to manually delete it, but to also make it available again in the future. In contrast, scheduling key deletion requires a waiting period of 7 to 30 days, after which you cannot recover the deleted KMS key.
- To own the original copy of the key material, and to keep it outside of AWS for additional durability and disaster recovery during the complete lifecycle of the key material.
- For asymmetric keys and HMAC keys, importing creates compatible and interoperable keys that operate within and outside of AWS.

You can audit and monitor the use and management of a KMS key with imported key material. AWS KMS records an event in your AWS CloudTrail log when you create the KMS key, download the wrapping public key and import token, and import the key material. AWS KMS also records an event when you manually delete imported key material or when AWS KMS deletes expired key material.

For information about important differences between KMS keys with imported key material and those with key material generated by AWS KMS, see About imported key material.

Supported KMS keys

AWS KMS supports imported key material for the following types of KMS keys. You cannot import key material into KMS keys in custom key stores.

- Symmetric encryption KMS keys
- Asymmetric RSA KMS keys (for encryption or signing, but not both)
- Asymmetric elliptic curve (ECC) KMS keys (for signing or deriving shared secrets, but not both)
- Asymmetric SM2 KMS keys – China Regions only (for encryption, signing, or deriving shared secrets)
- HMAC KMS keys
- Multi-Region keys of all supported types.

Regions

Imported key material is supported in all AWS Regions that AWS KMS supports.

In China Regions, the key material requirements for symmetric encryption KMS keys differ from other Regions. For details, see Importing key material step 3: Encrypt the key material.

Topics

- Planning to import key material
- Managing imported key material
- Importing key material step 1: Create an AWS KMS key without key material
- Importing key material step 2: Download the wrapping public key and import token
- Importing key material step 3: Encrypt the key material
- Importing key material step 4: Import the key material

Planning to import key material

Imported key material lets you protect your AWS resources under cryptographic keys that you generate. The key material that you import is associated with a particular KMS key. You can reimport the same key material into the same KMS key, but you cannot import different key material into the KMS key and you cannot convert a KMS key designed for imported key material into a KMS key with AWS KMS key material.

Learn more:

- the section called “Select a wrapping public key spec”
- the section called “Select a wrapping algorithm”

Topics

- About imported key material
- Protecting imported key material
- Permissions for importing key material
- Requirements for imported key material

About imported key material

Before you decide to import key material into AWS KMS, you should understand the following characteristics of imported key material.

You generate the key material

You are responsible for generating the key material using a source of randomness that meets your security requirements.

You can delete the key material

You can delete imported key material from a KMS key, immediately rendering the KMS key unusable. Also, when you import key material into a KMS key, you can determine whether the key expires and set its expiration time. When the expiration time arrives, AWS KMS deletes the key material. Without key material, the KMS key cannot be used in any cryptographic operation. To restore the key, you must reimport the same key material into the key.

You cannot change the key material

When you import key material into a KMS key, the KMS key is permanently associated with that key material. You can reimport the same key material, but you cannot import different key material into that KMS key. Also, you cannot enable automatic key rotation for a KMS key with imported key material. However, you can manually rotate a KMS key with imported key material.

You cannot change the key material origin

KMS keys designed for imported key material have an origin value of `EXTERNAL` that cannot be changed. You cannot convert a KMS key for imported key material to use key material from any other source, including AWS KMS. Similarly, you cannot convert a KMS key with AWS KMS key material into one designed for imported key material.

You cannot export key material

You cannot export any key material that you imported. AWS KMS cannot return the imported key material to you in any form. You must maintain a copy of your imported key material outside of AWS, preferably in a key manager, such as a hardware security module (HSM), so you can re-import the key material if you delete it or it expires.

You can create multi-Region keys with imported key material

Multi-Region with imported key material have the features of KMS keys with imported key material, and can interoperate between AWS Regions. To create a multi-Region key with imported key material, you must import the same key material into the primary KMS key and into each replica key. For details, see Importing key material into multi-Region keys.

Asymmetric keys and HMAC keys are portable and interoperable

You can use your asymmetric key material and HMAC key material outside of AWS to interoperate with AWS KMS keys with the same imported key material.

Unlike the AWS KMS symmetric ciphertext, which is inextricably bound to the KMS key used in the algorithm, AWS KMS uses standard HMAC and asymmetric formats for encryption, signing, and MAC generation. As a result, the keys are portable and support traditional escrow key scenarios.

When your KMS key has imported key material, you can use the imported key material outside of AWS to perform the following operations.

- HMAC keys — You can verify a HMAC tag that was generated by the HMAC KMS key with imported key material. You can also use the HMAC KMS key with the imported key material to verify an HMAC tag that was generated by the key material outside of AWS.
- Asymmetric encryption keys — You can use your private asymmetric encryption key outside of AWS to decrypt a ciphertext encrypted by the KMS key with the corresponding public key. You can also use your asymmetric KMS key to decrypt an asymmetric ciphertext that was generated outside of AWS.
- Asymmetric signing keys — You can use your asymmetric signing KMS key with imported key material to verify digital signatures generated by your private signing key outside of AWS. You can also use your asymmetric public signing key outside of AWS to verify signatures generated by your asymmetric KMS key.
- Asymmetric key agreement keys — You can use your asymmetric public key agreement KMS key with imported key material to derive shared secrets with a peer outside of AWS.

If you import the same key material into different KMS keys in the same AWS Region, those keys are also interoperable. To create interoperable KMS keys in different AWS Regions, create a multi-Region key with imported key material.

Symmetric encryption keys are not portable or interoperable

The symmetric ciphertexts that AWS KMS produces are not portable or interoperable. AWS KMS does not publish the symmetric ciphertext format that portability requires, and the format might change without notice.

- AWS KMS cannot decrypt symmetric ciphertexts that you encrypt outside of AWS, even if you use key material that you have imported.
- AWS KMS does not support decrypting any AWS KMS symmetric ciphertext outside of AWS KMS, even if the ciphertext was encrypted under a KMS key with imported key material.
- KMS keys with the same imported key material are not interoperable. The symmetric ciphertext that AWS KMS generates ciphertext that is specific to each KMS key. This ciphertext format guarantees that only the KMS key that encrypted data can decrypt it.

Also, you cannot use any AWS tools, such as the AWS Encryption SDK or Amazon S3 client-side encryption, to decrypt AWS KMS symmetric ciphertexts.

As a result, you cannot use keys with imported key material to support key escrow arrangements where an authorized third party with conditional access to key material can decrypt certain ciphertexts outside of AWS KMS. To support key escrow, use the AWS Encryption SDK to encrypt your message under a key that is independent of AWS KMS.

You're responsible for availability and durability

AWS KMS is designed to keep imported key material highly available. But AWS KMS does not maintain the durability of imported key material at the same level as key material that AWS KMS generates. For details, see Protecting imported key material.

Protecting imported key material

The key material that you import is protected in transit and at rest. Before importing the key material, you encrypt (or "wrap") the key material with the public key of an RSA key pair generated in AWS KMS hardware security modules (HSMs) validated under the FIPS 140-2 Cryptographic Module Validation Program. You can encrypt the key material directly with the wrapping public key, or encrypt the key material with an AES symmetric key, and then encrypt the AES symmetric key with the RSA public key.

Upon receipt, AWS KMS decrypts the key material with the corresponding private key in a AWS KMS HSM and re-encrypts it under an AES symmetric key that exists only in the volatile memory of the HSM. Your key material never leaves the HSM in plain text. It is decrypted only while it is in use and only within AWS KMS HSMs.

Use of your KMS key with imported key material is determined solely by the access control policies that you set on the KMS key. In addition, you can use aliases and tags to identify and control access to the KMS key. You can enable and disable the key, view and edit its properties, and monitor it using services like AWS CloudTrail.

However, you maintain the only failsafe copy of your key material. In return for this extra measure of control, you are responsible for durability and overall availability of the imported key material. AWS KMS is designed to keep imported key material highly available. But AWS KMS does not maintain the durability of imported key material at the same level as key material that AWS KMS generates.

This difference in durability is meaningful in the following cases:

- When you set an expiration time for your imported key material, AWS KMS deletes the key material after it expires. AWS KMS does not delete the KMS key or its metadata. You can create a Amazon CloudWatch alarm that notifies you when imported key material is approaching its expiration date.

  You cannot delete key material that AWS KMS generates for a KMS key and you cannot set AWS KMS key material to expire, although you can rotate it.
- When you manually delete imported key material, AWS KMS deletes the key material but does not delete the KMS key or its metadata. In contrast, scheduling key deletion requires a waiting period of 7 to 30 days, after which AWS KMS permanently deletes the KMS key, its metadata, and its key material.
- In the unlikely event of certain region-wide failures that affect AWS KMS (such as a total loss of power), AWS KMS cannot automatically restore your imported key material. However, AWS KMS can restore the KMS key and its metadata.

You *must* retain a copy of the imported key material outside of AWS in a system that you control. We recommend that you store an exportable copy of the imported key material in a key management system, such as an HSM. If your imported key material is deleted or expires, its associated KMS key becomes unusable until you reimport the same key material. If your imported key material is permanently lost, any ciphertext encrypted under the KMS key is unrecoverable.

Permissions for importing key material

To create and manage KMS keys with imported key material, the user needs permission for the operations in this process. You can provide the `kms:GetParametersForImport`, `kms:ImportKeyMaterial`, and `kms:DeleteImportedKeyMaterial` permissions in the key policy when you create the KMS key. In the AWS KMS console, these permissions are added automatically for key administrators when you create a key with an **External** key material origin.

To create KMS keys with imported key material, the principal needs the following permissions.

- kms:CreateKey (IAM policy)
  - To limit this permission to KMS keys with imported key material, use the kms:KeyOrigin policy condition with a value of EXTERNAL.

    ```
    {
      "Sid": "CreateKMSKeysWithoutKeyMaterial",
      "Effect": "Allow",
      "Resource": "*",
      "Action": "kms:CreateKey",
      "Condition": {
        "StringEquals": {
          "kms:KeyOrigin": "EXTERNAL"
        }
      }
    }
    ```
- kms:GetParametersForImport (Key policy or IAM policy)
  - To limit this permission to requests that use a particular wrapping algorithm and wrapping key spec, use the kms:WrappingAlgorithm and kms:WrappingKeySpec policy conditions.
- kms:ImportKeyMaterial (Key policy or IAM policy)
  - To allow or prohibit key material that expires and control the expiration date, use the kms:ExpirationModel and kms:ValidTo policy conditions.

To reimport imported key material, the principal needs the kms:GetParametersForImport and kms:ImportKeyMaterial permissions.

To delete imported key material, the principal needs kms:DeleteImportedKeyMaterial permission.

For example, to give the example KMSAdminRole permission to manage all aspects of a KMS key with imported key material, include a key policy statement like the following one in the key policy of the KMS key.

```
{
  "Sid": "Manage KMS keys with imported key material",
```

```
    "Effect": "Allow",
    "Resource": "*",
    "Principal": {
      "AWS": "arn:aws:iam::111122223333:role/KMSAdminRole"
    },
    "Action": [
      "kms:GetParametersForImport",
      "kms:ImportKeyMaterial",
      "kms:DeleteImportedKeyMaterial"
    ]
  }
```

Requirements for imported key material

The key material that you import must be compatible with the key spec of the associated KMS key. For asymmetric key pairs, import only the private key of the pair. AWS KMS derives the public key from the private key.

AWS KMS supports the following key specs for KMS keys with imported key material.

KMS key key spec	Key material requirements
Symmetric encryption keys SYMMETRIC_DEFAULT	256-bits (32 bytes) of binary data In China Regions, it must be a 128-bits (16 bytes) of binary data.
HMAC keys HMAC_224 HMAC_256 HMAC_384 HMAC_512	HMAC key material must conform to RFC 2104. The key length must match the length specified by the key spec.
RSA asymmetric private key RSA_2048 RSA_3072	The RSA asymmetric private key that you import must be part of a key pair that conforms to RFC 3447. **Modulus**: 2048 bits, 3072 bits or 4096 bits

KMS key key spec	Key material requirements
RSA_4096	**Number of primes**: 2 (multi-prime RSA keys are not supported) Asymmetric key material must be BER-encoded or DER-encoded in Public-Key Cryptography Standards (PKCS) #8 format that complies with RFC 5208.
Elliptic curve asymmetric private key ECC_NIST_P256 (secp256r1) ECC_NIST_P384 (secp384r1) ECC_NIST_P521 (secp521r1) ECC_SECG_P256K1 (secp256k1)	The ECC asymmetric private key that you import must be part of a key pair that conforms to RFC 5915. **Curve**: NIST P-256, NIST P-384, NIST P-521, or Secp256k1 **Parameters**: Named curves only (ECC keys with explicit parameters are rejected) **Public point coordinates**: May be compressed, uncompressed, or projective Asymmetric key material must be BER-encoded or DER-encoded in Public-Key Cryptography Standards (PKCS) #8 format that complies with RFC 5208.

KMS key key spec	Key material requirements
SM2 asymmetric private key (China Regions only)	The SM2 asymmetric private key that you import must be part of a key pair that conforms to GM/T 0003. **Curve**: SM2 **Parameters**: Named curve only (SM2 keys with explicit parameters are rejected) **Public point coordinates**: May be compressed, uncompressed, or projective Asymmetric key material must be BER-encoded or DER-encoded in Public-Key Cryptography Standards (PKCS) #8 format that complies with RFC 5208.

Managing imported key material

These topics explain how to import and reimport key material into a KMS key and how to create imported key material that automatically expires.

Topics

Overview of importing key material

The following overview explains how to import your key material into AWS KMS. For more details about each step in the process, see the corresponding topic.

1. Create a KMS key with no key material – The origin must be EXTERNAL. A key origin of EXTERNAL indicates that the key is designed for imported key material and prevents AWS KMS from generating key material for the KMS key. In a later step you will import your own key material into this KMS key.

   The key material that you import must be compatable with the key spec of the associated AWS KMS key. For more information about compatability, see the section called "Requirements for imported key material".
2. Download the wrapping public key and import token – After completing step 1, download a wrapping public key and an import token. These items protect your key material while it's imported to AWS KMS.

   In this step, you choose the type ("key spec") of the RSA wrapping key and the wrapping algorithm that you'll use to encrypt your data in transit to AWS KMS. You can choose a different wrapping key spec and wrapping key algorithm each time you import or reimport the same key material.
3. Encrypt the key material – Use the wrapping public key that you downloaded in step 2 to encrypt the key material that you created on your own system.
4. Import the key material – Upload the encrypted key material that you created in step 3 and the import token that you downloaded in step 2.

   At this stage, you can set an optional expiration time. When imported key material expires, AWS KMS deletes it, and the KMS key becomes unusable. To continue to use the KMS key, you must reimport the **same** key material.

   When the import operation completes successfully, the key state of the KMS key changes from `PendingImport` to `Enabled`. You can now use the KMS key in cryptographic operations.

AWS KMS records an entry in your AWS CloudTrail log when you create the KMS key, download the wrapping public key and import token, and import the key material. AWS KMS also records an entry when you delete imported key material or when AWS KMS deletes expired key material.

Reimporting key material

If you manage a KMS key with imported key material, you might need to reimport the key material. You might reimport key material to replace expiring or deleted key material, or to change the expiration model or expiration date of the key material.

When you import key material into a KMS key, the KMS key is permanently associated with that key material. You can reimport the same key material, but you cannot import different key material into that KMS key. You cannot rotate the key material and AWS KMS cannot create key material for a KMS key with imported key material.

You can reimport key material at any time, on any schedule that meets your security requirements. You do not have to wait until the key material is at or close to its expiration time.

To reimport key material, use the same procedure that you used to import the key material the first time, with the following exceptions.

- Use an existing KMS key, instead of creating a new KMS key. You can skip Step 1 of the import procedure.
- When you reimport key material, you can change the expiration model and expiration date.

Each time you import key material to a KMS key, you need to download and use a new wrapping key and import token for the KMS key. The wrapping procedure does not affect the content of the key material, so you can use different wrapping public keys and different wrapping algorithms to import the same key material.

Identifying KMS keys with imported key material

When you create a KMS key with no key material, the value of the `Origin` property of the KMS key is `EXTERNAL`, and it cannot be changed. Unlike the key state, the `Origin` value doesn't depend on the presence or absence of key material.

You can use the `EXTERNAL` origin value to identify KMS keys designed for imported key material. You can find the key origin in the AWS KMS console or by using the DescribeKey operation. You can also view the properties of the key material, such as whether and when it expires by using the console or the APIs.

To identify KMS keys with imported key material (console)

1. Open the AWS KMS console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. Use either of the following techniques to view the `Origin` property of your KMS keys.

- To add an **Origin** column to your KMS key table, in the upper right corner, choose the **Settings** icon. Choose **Origin** and choose **Confirm**. The **Origin** column makes it easy to identify KMS keys with an **External (Import Key material)** origin property value.
- To find the value of the `Origin` property of a particular KMS key, choose the key ID or alias of the KMS key. Then choose the **Cryptographic configuration** tab. The tabs are below the **General configuration** section.

4. To view detailed information about the key material, choose the **Key material** tab. This tab appears on the detail page only for KMS keys with imported key material.

To identify KMS keys with imported key material (AWS KMS API)

Use the DescribeKey operation. The response includes the `Origin` property of the KMS key, the expiration model, and the expiration date, as shown in the following example.

```
$  aws kms describe-key --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
{
    "KeyMetadata": {
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "Origin": "EXTERNAL",
        "ExpirationModel": "KEY_MATERIAL_EXPIRES"
        "ValidTo": 2023-06-05T12:00:00+00:00,
        "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "AWSAccountId": "111122223333",
        "CreationDate": 2018-06-09T00:06:50.831000+00:00,
        "Enabled": false,
        "MultiRegion": false,
        "Description": "",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "PendingImport",
        "KeyManager": "CUSTOMER",
        "KeySpec": "SYMMETRIC_DEFAULT",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ]
    }
}
```

Creating a CloudWatch alarm for expiration of imported key material

You can create a CloudWatch alarm that notifies you when the imported key material in a KMS key is approaching its expiration time. For example, the alarm can notify you when the time to expire is less than 30 days away.

When you import key material into a KMS key, you can optionally specify a date and time when the key material expires. When the key material expires, AWS KMS deletes the key material and the KMS key becomes unusable. To use the KMS key again, you must reimport the key material. However, if you reimport the key material before it expires, you can avoid disrupting processes that use that KMS key.

This alarm uses the `SecondsUntilKeyMaterialExpires` metric that AWS KMS publishes to CloudWatch for KMS keys with imported key material that expires. Each alarm uses this metric to monitor the imported key material for a particular KMS key. You cannot create a single alarm for all KMS keys with expiring key material or an alarm for KMS keys that you might create in the future.

Requirements

The following resources are required for a CloudWatch alarm that monitors the expiration of imported key material.

- A KMS key with imported key material that expires. For help, see Identifying KMS keys with imported key material.
- An Amazon SNS topic. For details, see Creating an Amazon SNS topic in the *Amazon CloudWatch User Guide*.

Create the alarm

Follow the instructions in Create a CloudWatch alarm based on a static threshold using the following required values. For other fields, accept the default values and provide names as requested.

Field	Value
Select metric	Choose **KMS**, then choose **Per-Key Metrics**. Choose the row with the KMS key and the `SecondsUntilKeyMaterialExpires` metric. Then choose **Select metric**.

Field	Value
	The **Metrics** list displays the `SecondsUntilKeyMaterialExpires` metric only for KMS keys with imported key material that expires. If you don't have KMS keys with these properties in the account and Region, this list is empty.
Statistic	Minimum
Period	1 minute
Threshold type	Static
Whenever ...	**Whenever** *`metric-name`* is **Greater** than 1

Deleting imported key material

You can delete the imported key material from a KMS key at any time. Also, when imported key material with an expiration date expires, AWS KMS deletes the key material. In either case, when the key material is deleted, the key state of the KMS key changes to *pending import*, and the KMS key can't be used in any cryptographic operations until you reimport the same key material. (You cannot import any other key material into the KMS key.)

Along with disabling the KMS key and withdrawing permissions, deleting key material can be used as a strategy to quickly, but temporarily, halt the use of the KMS key. In contrast, scheduling the deletion of a KMS key with imported key material also quickly halts the use of the KMS key. However, if the deletion is not canceled during the waiting period, the KMS key, the key material, and all key metadata are permanently deleted. For details, see the section called "Deleting a KMS key with imported key material".

To delete key material, you can use the AWS KMS console or the DeleteImportedKeyMaterial API operation. AWS KMS records an entry in your AWS CloudTrail log when you delete imported key material and when AWS KMS deletes expired key material.

Topics

How deleting key material affects AWS services

When you delete key material, the KMS key with no key material becomes unusable right away (subject to eventual consistency). However, resources encrypted with data keys protected by the KMS key are not affected until the KMS key is used again, such as to decrypt the data key. This issue affects AWS services, many of which use data keys to protect your resources. For details, see How unusable KMS keys affect data keys.

Delete key material (console)

You can use the AWS Management Console to delete key material.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Do one of the following:
   - Select the check box for a KMS key with imported key material. Choose **Key actions**, **Delete key material**.
   - Choose the alias or key ID of a KMS key with imported key material. Choose the **Key material** tab and then choose **Delete key material**.
5. Confirm that you want to delete the key material and then choose **Delete key material**. The KMS key's status, which corresponds to its key state, changes to **Pending import**.

Delete key material (AWS KMS API)

To use the AWS KMS API to delete key material, send a DeleteImportedKeyMaterial request. The following example shows how to do this with the AWS CLI.

Replace *1234abcd-12ab-34cd-56ef-1234567890ab* with the key ID of the KMS key whose key material you want to delete. You can use the KMS key's key ID or ARN but you cannot use an alias for this operation.

```
$ aws kms delete-imported-key-material --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
```

Deleting a KMS key with imported key material

Deleting the key material of a KMS key with imported key material is temporary and reversible. To restore the key, reimport its key material.

In contrast, deleting a KMS key is irreversible. If you schedule key deletion and the required waiting period expires, AWS KMS permanently and irreversibly deletes the KMS key, its key material, and all metadata associated with the KMS key.

However, the risk and consequence of deleting a KMS key with imported key material depends on the type ("key spec") of the KMS key.

- Symmetric encryption keys — If you delete a symmetric encryption KMS key, all remaining ciphertexts encrypted by that key are unrecoverable. You cannot create a new symmetric encryption KMS key that can decrypt the ciphertexts of a deleted symmetric encryption KMS key, even if you have the same key material. Metadata unique to each KMS key is cryptographically bound to each symmetric ciphertext. This security feature guarantees that only the KMS key that encrypted the symmetric ciphertext can decrypt it, but it prevents you from recreating an equivalent KMS key.
- Asymmetric and HMAC keys — If you have the original key material, you can create a new KMS key with the same cryptographic properties as an asymmetric or HMAC KMS key that was deleted. AWS KMS generates standard RSA ciphertexts and signatures, ECC signatures, and HMAC tags, which do not include any unique security features. Also, you can use an HMAC key or the private key of an asymmetric key pair outside of AWS.

  A new KMS key that you create with the same asymmetric or HMAC key material will have a different key identifier. You will have to create a new key policy, recreate any aliases, and update existing IAM policies and grants to refer to the new key.

Importing key material step 1: Create an AWS KMS key without key material

By default, AWS KMS creates key material for you when you create a KMS key. To import your own key material instead, start by creating a KMS key with no key material. Then import the key material. To create a KMS key with no key material, use AWS KMS console or the CreateKey operation.

To create a key with no key material, specify an origin of EXTERNAL. The origin property of a KMS key is immutable. Once you create it, you cannot convert a KMS key designed for imported key material into a KMS key with key material from AWS KMS or any other source.

The key state of a KMS key with an EXTERNAL origin and no key material is PendingImport. A KMS key can remain in PendingImport state indefinitely. However, you cannot use a KMS key in PendingImport state in cryptographic operations. When you import key material, the key state of the KMS key changes to Enabled, and you can use it in cryptographic operations.

AWS KMS records an event in your AWS CloudTrail log when you create the KMS key, download the public key and import token, and import the key material. AWS KMS also records a CloudTrail event when you delete imported key material or when AWS KMS deletes expired key material.

For information about creating multi-Region keys with imported key material, see Importing key material into multi-Region keys.

Topics

- Creating a KMS key with no key material (console)
- Creating a KMS key with no key material (AWS KMS API)

Creating a KMS key with no key material (console)

You only need to create a KMS key for the imported key material once. You can import and reimport the same key material into the existing KMS key as often as you need to, but you cannot import different key material into a KMS key. For details, see Step 2: Download the wrapping public key and import token.

To find existing KMS keys with imported key material in your **Customer managed keys** table, use the gear icon in the upper right corner to show the **Origin** column in the list of KMS keys. Imported keys have an **Origin** value of **External (Import Key material)**.

To create a KMS key with imported key material, begin by following the basic instructions for creating a KMS key of your preferred key type, with the following exception.

After choosing the key usage, do the following:

1. Expand **Advanced options**.
2. For **Key material origin**, choose **External (Import key material)**.

3. Choose the check box next to **I understand the security and durability implications of using an imported key** to indicate that you understand the implications of using imported key material. To read about these implications, see Protecting imported key material.
4. Return to the basic instructions. The remaining steps of the basic procedure are the same for all KMS keys of that type.

When you choose **Finish**, you have created a KMS key with no key material and a status (key state) of **Pending import**.

However. instead of returning to the **Customer managed keys** table, the console displays a page where you can download the public key and import token that you need to import your key material. You can continue with the download step now, or choose **Cancel** to stop at this point. You can return to this download step at any time.

Next: Step 2: Download the wrapping public key and import token.

Creating a KMS key with no key material (AWS KMS API)

To use the AWS KMS API to create a symmetric encryption KMS key with no key material, send a CreateKey request with the `Origin` parameter set to `EXTERNAL`. The following example shows how to do this with the AWS Command Line Interface (AWS CLI).

```
$ aws kms create-key --origin EXTERNAL
```

When the command is successful, you see output similar to the following. The AWS KMS key's `Origin` is `EXTERNAL` and its `KeyState` is `PendingImport`.

> **Tip**
>
> If the command does not succeed, you might see a `KMSInvalidStateException` or a `NotFoundException`. You can retry the request.

```
{
    "KeyMetadata": {
        "Origin": "EXTERNAL",
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "Description": "",
        "Enabled": false,
        "MultiRegion": false,
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "PendingImport",
        "CreationDate": 1568289600.0,
        "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "AWSAccountId": "111122223333",
        "KeyManager": "CUSTOMER",
        "KeySpec": "SYMMETRIC_DEFAULT",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ]
    }
}
```

Copy the `KeyId` value from your command output to use in later steps, and then proceed to Step 2: Download the wrapping public key and import token.

Note

This command creates a symmetric encryption KMS key with a `KeySpec` of `SYMMETRIC_DEFAULT` and `KeyUsage` of `ENCRYPT_DECRYPT`. You can use the optional parameters `--key-spec` and `--key-usage` to create an asymmetric or HMAC KMS key. For more information, see the CreateKey operation.

Importing key material step 2: Download the wrapping public key and import token

After you create a AWS KMS key with no key material, download a wrapping public key and an import token for that KMS key by using the AWS KMS console or the GetParametersForImport API. The wrapping public key and import token are an indivisible set that must be used together.

You will use the wrapping public key to encrypt your key material for transport. Before downloading an RSA wrapping key pair, you select the length (key spec) of the RSA wrapping key pair and the wrapping algorithm that you will use to encrypt your imported key material for transport in step 3. AWS KMS also supports the SM2 wrapping key spec (China Regions only).

Each wrapping public key and import token set is valid for 24 hours. If you don't use them to import key material within 24 hours of downloading them, you must download a new set. You can download new wrapping public key and import token sets at any time. This lets you change your RSA wrapping key length ("key spec") or replace a lost set.

You can also download a wrapping public key and import token set to reimport the same key material into a KMS key. You might do this to set or change the expiration time for the key material, or to restore expired or deleted key material. You must download and re-encrypt your key material every time you import it to AWS KMS.

Use of the wrapping public key

The download includes a public key that is unique to your AWS account, also called a *wrapping public key*.

Before you import key material, you encrypt the key material with the public wrapping key, and then upload the encrypted key material to AWS KMS. When AWS KMS receives your encrypted key material, it decrypts the key material with the corresponding private key, then reencrypts the key material under an AES symmetric key, all within an AWS KMS hardware security module (HSM).

Use of the import token

The download includes an import token with metadata that ensures that your key material is imported correctly. When you upload your encrypted key material to AWS KMS, you must upload the same import token that you downloaded in this step.

Select a wrapping public key spec

To protect your key material during import, you encrypt it using wrapping public key that you download from AWS KMS, and a supported wrapping algorithm. You select a key spec before you download your wrapping public key and import token. All wrapping key pairs are generated in AWS KMS hardware security modules (HSMs). The private key never leaves the HSM in plain text.

RSA wrapping key specs

The *key spec* of the wrapping public key determines the length of the keys in the RSA key pair that protects your key material during its transport to AWS KMS. In general, we recommend using the longest wrapping public key that is practical. We offer several wrapping public key specs to support a variety of HSMs and key managers.

AWS KMS supports the following key specs for the RSA wrapping keys used to import key material of all types, except as noted.

- RSA_4096 (preferred)
- RSA_3072
- RSA_2048

> **Note**
>
> The following combination is NOT supported: ECC_NIST_P521 key material, the RSA_2048 public wrapping key spec, and an RSAES_OAEP_SHA_* wrapping algorithm.
> You cannot directly wrap ECC_NIST_P521 key material with a RSA_2048 public wrapping key. Use a larger wrapping key or an RSA_AES_KEY_WRAP_SHA_* wrapping algorithm.

SM2 wrapping key spec (China Regions only)

AWS KMS supports the following key spec for the SM2 wrapping keys used to import asymmetric key material.

- SM2

Select a wrapping algorithm

To protect your key material during import, you encrypt it using the downloaded wrapping public key and a supported wrapping algorithm.

AWS KMS supports several standard RSA wrapping algorithms and a two-step hybrid wrapping algorithm. In general, we recommend using the most secure wrapping algorithm that is compatible with your imported key material and wrapping key spec. Typically, you choose an algorithm that is supported by the hardware security module (HSM) or key management system that protects your key material.

The following table shows the wrapping algorithms that are supported for each type of key material and KMS key. The algorithms are listed in preference order.

Key material	Supported wrapping algorithm and spec
Symmetric encryption key 256-bit AES key 128-bit SM4 key (China Regions only)	Wrapping algorithms: RSAES_OAEP_SHA_256 RSAES_OAEP_SHA_1 Deprecated wrapping algorithms: RSAES_PKCS1_V1 **Note** As of October 10, 2023, AWS KMS does not support the RSAES_PKCS1_V1_5 wrapping algorithm. Wrapping key specs: RSA_2048 RSA_3072 RSA_4096
Asymmetric RSA private key	Wrapping algorithms: RSA_AES_KEY_WRAP_SHA_256 RSA_AES_KEY_WRAP_SHA_1 SM2PKE (China Regions only) Wrapping key specs: RSA_2048 RSA_3072 RSA_4096

Key material	Supported wrapping algorithm and spec
	SM2 (China Regions only)
Asymmetric elliptic curve (ECC) private key You cannot use the RSAES_OAEP_SHA_* wrapping algorithms with the RSA_2048 wrapping key spec to wrap ECC_NIST_P521 key material.	Wrapping algorithms: RSA_AES_KEY_WRAP_SHA_256 RSA_AES_KEY_WRAP_SHA_1 RSAES_OAEP_SHA_256 RSAES_OAEP_SHA_1 SM2PKE (China Regions only) Wrapping key specs: RSA_2048 RSA_3072 RSA_4096 SM2 (China Regions only)
Asymmetric SM2 private key (China Regions only)	Wrapping algorithms: RSAES_OAEP_SHA_256 RSAES_OAEP_SHA_1 SM2PKE (China Regions only) Wrapping key specs: RSA_2048 RSA_3072 RSA_4096 SM2 (China Regions only)

Key material	Supported wrapping algorithm and spec
HMAC key	Wrapping algorithms: RSAES_OAEP_SHA_256 RSAES_OAEP_SHA_1 Wrapping key specs: RSA_2048 RSA_3072 RSA_4096

> **Note**
>
> The RSA_AES_KEY_WRAP_SHA_256 and RSA_AES_KEY_WRAP_SHA_1 wrapping algorithms are not supported in China Regions.

- RSA_AES_KEY_WRAP_SHA_256 – A two-step hybrid wrapping algorithm that combines encrypting your key material with an AES symmetric key that you generate, and then encrypting the AES symmetric key with the downloaded RSA public wrapping key and the RSAES_OAEP_SHA_256 wrapping algorithm.

  An RSA_AES_KEY_WRAP_SHA_* wrapping algorithm is required for wrapping RSA private key material, except in China Regions, where you must use the SM2PKE wrapping algorithm.
- RSA_AES_KEY_WRAP_SHA_1 – A two-step hybrid wrapping algorithm that combines encrypting your key material with an AES symmetric key that you generate, and then encrypting the AES symmetric key with the downloaded RSA wrapping public key and the RSAES_OAEP_SHA_1 wrapping algorithm.

  An RSA_AES_KEY_WRAP_SHA_* wrapping algorithm is required for wrapping RSA private key material, except in China Regions, where you must use the SM2PKE wrapping algorithm.
- RSAES_OAEP_SHA_256 – The RSA encryption algorithm with Optimal Asymmetric Encryption Padding (OAEP) with the SHA-256 hash function.

- `RSAES_OAEP_SHA_1` – The RSA encryption algorithm with Optimal Asymmetric Encryption Padding (OAEP) with the SHA-1 hash function.
- `RSAES_PKCS1_V1_5` (Deprecated; as of October 10, 2023, AWS KMS does not support the RSAES_PKCS1_V1_5 wrapping algorithm) – The RSA encryption algorithm with the padding format defined in PKCS #1 Version 1.5.
- `SM2PKE` (China Regions only) – An elliptic curve based encryption algorithm defined by OSCCA in GM/T 0003.4-2012.

Topics

Downloading the wrapping public key and import token (console)

You can use the AWS KMS console to download the wrapping public key and import token.

1. If you just completed the steps to create a KMS key with no key material and you are on the **Download wrapping key and import token** page, skip to Step 9.
2. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
3. To change the AWS Region, use the Region selector in the upper-right corner of the page.
4. In the navigation pane, choose **Customer managed keys**.

> **Tip**
>
> You can import key material only into an KMS key with an **Origin** of **External (Import key material)**. This indicates that the KMS key was created with no key material. To add the **Origin** column to your table, in the upper-right corner of the page, choose the settings icon (⚙). Turn on **Origin**, and then choose **Confirm**.

5. Choose the alias or key ID of the KMS key that is pending import.
6. Choose the **Cryptographic configuration** tab and view its values. The tabs are below the **General configuration** section.

You can only import key material into KMS keys an **Origin** of **External (Import Key material)**. For information about creating KMS keys with imported key material, see, Importing key material for AWS KMS keys.

7. Choose the **Key material** tab and then choose **Import key material**.

   The **Key material** tab appears only for KMS keys that have an **Origin** value of **External (Import Key material)**.

8. For **Select wrapping key spec**, choose the configuration for your KMS key. After you create this key, you can't change the key spec.

9. For **Select wrapping algorithm**, choose the option that you will use to encrypt your key material. For more information about the options, see Select a Wrapping Algorithm.

10. Choose **Download wrapping public key and import token**, and then save the file.

    If you have a **Next** option, to continue the process now, choose **Next**. To continue later, choose **Cancel**.

11. Decompress the `.zip` file that you saved in the previous step (`Import_Parameters_`*`<key_id>`*`_`*`<timestamp>`*).

    The folder contains the following files:

    - A wrapping public key in a file named `WrappingPublicKey.bin`.
    - An import token in a file named `ImportToken.bin`.
    - A text file named README.txt. This file contains information about the wrapping public key, the wrapping algorithm to use to encrypt your key material, and the date and time when the wrapping public key and import token expire.

12. To continue the process, see encrypt your key material.

Downloading the wrapping public key and import token (AWS KMS API)

To download the public key and import token, use the GetParametersForImport API. Specify the KMS key that will be associated with the imported key material. This KMS key must have an Origin value of `EXTERNAL`.

This example specifies the `RSA_AES_KEY_WRAP_SHA_256` wrapping algorithm, the `RSA_3072` wrapping public key spec, and an example key ID. Replace these example values with valid values for your download. For the key ID, you can use a key ID or key ARN, but you cannot use an alias name or alias ARN in this operation.

```
$ aws kms get-parameters-for-import \
    --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
    --wrapping-algorithm RSA_AES_KEY_WRAP_SHA_256 \
    --wrapping-key-spec RSA_3072
```

When the command is successful, you see output similar to the following:

```
{
    "ParametersValidTo": 1568290320.0,
    "PublicKey": "public key (base64 encoded)",
    "KeyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "ImportToken": "import token (base64 encoded)"
}
```

To prepare the data for the next step, base64 decode the public key and import token and save the decoded values in files.

To base64 decode the public key and import token:

1. Copy the base64 encoded public key (represented by *`public key (base64 encoded)`* in the example output), paste it into a new file, and then save the file. Give the file a descriptive name, such as `PublicKey.b64`.
2. Use OpenSSL to base64 decode the file's contents and save the decoded data to a new file. The following example decodes the data in the file that you saved in the previous step (`PublicKey.b64`) and saves the output to a new file named `WrappingPublicKey.bin`.

   ```
   $ openssl enc -d -base64 -A -in PublicKey.b64 -out WrappingPublicKey.bin
   ```

3. Copy the base64 encoded import token (represented by *`import token (base64 encoded)`* in the example output), paste it into a new file, and then save the file. Give the file a descriptive name, for example `importtoken.b64`.
4. Use OpenSSL to base64 decode the file's contents and save the decoded data to a new file. The following example decodes the data in the file that you saved in the previous step (`ImportToken.b64`) and saves the output to a new file named `ImportToken.bin`.

```
$ openssl enc -d -base64 -A -in importtoken.b64 -out ImportToken.bin
```

Proceed to Step 3: Encrypt the key material.

Importing key material step 3: Encrypt the key material

After you download the public key and import token, encrypt your key material using the public key that you downloaded and the wrapping algorithm that you specified. If you need to replace the public key or import token, or change the wrapping algorithm, you must download a new public key and import token. For information about the public keys and wrapping algorithms that AWS KMS supports, see Select a wrapping public key spec and Select a wrapping algorithm.

The key material must be in binary format. For detailed information, see Requirements for imported key material.

Note

For asymmetric key pairs, encrypt and import only the private key. AWS KMS derives the public key from the private key.
The following combination is NOT supported: ECC_NIST_P521 key material, the RSA_2048 public wrapping key spec, and an RSAES_OAEP_SHA_* wrapping algorithm.
You cannot directly wrap ECC_NIST_P521 key material with a RSA_2048 public wrapping key. Use a larger wrapping key or an RSA_AES_KEY_WRAP_SHA_* wrapping algorithm.
The RSA_AES_KEY_WRAP_SHA_256 and RSA_AES_KEY_WRAP_SHA_1 wrapping algorithms are not supported in China Regions.

Typically, you encrypt your key material when you export it from your hardware security module (HSM) or key management system. For information about how to export key material in binary format, see the documentation for your HSM or key management system. You can also refer to the following section that provides a proof of concept demonstration using OpenSSL.

When you encrypt your key material, use the same wrapping algorithm that you specified when you downloaded the public key and import token. To find the wrapping algorithm that you specified, see the CloudTrail log event for the associated GetParametersForImport request.

Generate key material for testing

The following OpenSSL commands generate key material of each supported type for testing. These examples are provided only for testing and proof-of-concept demonstrations. For production systems, use a more secure method to generate your key material, such as a hardware security module or key management system.

To convert the private keys of asymmetric key pairs into DER-encoded format, pipe the key material generation command to the following `openssl pkcs8` command. The `topk8` parameter directs OpenSSL to take a private key as input and return a PKCS#8 formatted key. (The default behavior is the opposite.)

```
openssl pkcs8 -topk8 -outform der -nocrypt
```

The following commands generate test key material for each of the supported key types.

- Symmetric encryption key (32 bytes)

  This command generates a 256-bit symmetric key (32-byte random string) and saves it in the `PlaintextKeyMaterial.bin` file. You do not need to encode this key material.

  ```
  openssl rand -out PlaintextKeyMaterial.bin 32
  ```

  In China Regions only, you must generate a 128-bit symmetric key (16-byte random string).

  ```
  openssl rand -out PlaintextKeyMaterial.bin 16
  ```

- HMAC keys

  This command generates a random byte string of the specified size. You do not need to encode this key material.

  The length of your HMAC key must match the length defined by the key spec of the KMS key. For example, if the KMS key is HMAC_384, you must import a 384-bit (48-byte) key.

  ```
  openssl rand -out HMAC_224_PlaintextKey.bin 28

  openssl rand -out HMAC_256_PlaintextKey.bin 32

  openssl rand -out HMAC_384_PlaintextKey.bin 48
  ```

```
openssl rand -out HMAC_512_PlaintextKey.bin 64
```

- RSA private keys

```
openssl genpkey -algorithm rsa -pkeyopt rsa_keygen_bits:2048 | openssl pkcs8 -topk8 -
outform der -nocrypt > RSA_2048_PrivateKey.der

openssl genpkey -algorithm rsa -pkeyopt rsa_keygen_bits:3072 | openssl pkcs8 -topk8 -
outform der -nocrypt > RSA_3072_PrivateKey.der

openssl genpkey -algorithm rsa -pkeyopt rsa_keygen_bits:4096 | openssl pkcs8 -topk8 -
outform der -nocrypt > RSA_4096_PrivateKey.der
```

- ECC private keys

```
openssl genpkey -algorithm ec -pkeyopt ec_paramgen_curve:P-256 | openssl pkcs8 -topk8
 -outform der -nocrypt > ECC_NIST_P256_PrivateKey.der

openssl genpkey -algorithm ec -pkeyopt ec_paramgen_curve:P-384 | openssl pkcs8 -topk8
 -outform der -nocrypt > ECC_NIST_P384_PrivateKey.der

openssl genpkey -algorithm ec -pkeyopt ec_paramgen_curve:P-521 | openssl pkcs8 -topk8
 -outform der -nocrypt > ECC_NIST_P521_PrivateKey.der

openssl genpkey -algorithm ec -pkeyopt ec_paramgen_curve:secp256k1 | openssl pkcs8 -
topk8 -outform der -nocrypt > ECC_SECG_P256K1_PrivateKey.der
```

- SM2 private keys (China Regions only)

```
openssl genpkey -algorithm ec -pkeyopt ec_paramgen_curve:sm2 | openssl pkcs8 -topk8 -
outform der -nocrypt > SM2_PrivateKey.der
```

Examples of encrypting key material with OpenSSL

The following examples show how to use OpenSSL to encrypt your key material with the public key that you downloaded. To encrypt your key material using an SM2 public key (China Regions only), use the `SM2OfflineOperationHelper` class.

> ⚠ **Important**
>
> These examples are a proof of concept demonstration only. For production systems, use a more secure method (such as a commercial HSM or key management system) to generate and store your key material.
> The following combination is NOT supported: ECC_NIST_P521 key material, the RSA_2048 public wrapping key spec, and an RSAES_OAEP_SHA_* wrapping algorithm.
> You cannot directly wrap ECC_NIST_P521 key material with a RSA_2048 public wrapping key. Use a larger wrapping key or an RSA_AES_KEY_WRAP_SHA_* wrapping algorithm.

RSAES_OAEP_SHA_1

AWS KMS supports the RSAES_OAEP_SHA_1 for symmetric encryption keys (SYMMETRIC_DEFAULT), elliptic curve (ECC) private keys, SM2 private keys, and HMAC keys.

RSAES_OAEP_SHA_1 is not supported for RSA private keys. Also, you cannot use an RSA_2048 public wrapping key with any RSAES_OAEP_SHA_* wrapping algorithm to wrap an ECC_NIST_P521 (secp521r1) private key. You must use a larger public wrapping key or an RSA_AES_KEY_WRAP wrapping algorithm.

The following example encrypts your key material with the public key that you downloaded and the RSAES_OAEP_SHA_1 wrapping algorithm, and saves it in the `EncryptedKeyMaterial.bin` file.

In this example:

- *`WrappingPublicKey.bin`* is the file that contains the downloaded wrapping public key.
- *`PlaintextKeyMaterial.bin`* is the file that contains the key material that you are encrypting, such as `PlaintextKeyMaterial.bin`, `HMAC_384_PlaintextKey.bin` or `ECC_NIST_P521_PrivateKey.der`.

```
$ openssl pkeyutl \
    -encrypt \
    -in PlaintextKeyMaterial.bin \
    -out EncryptedKeyMaterial.bin \
    -inkey WrappingPublicKey.bin \
    -keyform DER \
    -pubin \
    -pkeyopt rsa_padding_mode:oaep \
    -pkeyopt rsa_oaep_md:sha1
```

RSAES_OAEP_SHA_256

AWS KMS supports the RSAES_OAEP_SHA_256 for symmetric encryption keys (SYMMETRIC_DEFAULT), elliptic curve (ECC) private keys, SM2 private keys, and HMAC keys.

RSAES_OAEP_SHA_256 is not supported for RSA private keys. Also, you cannot use an RSA_2048 public wrapping key with any RSAES_OAEP_SHA_* wrapping algorithm to wrap an ECC_NIST_P521 (secp521r1) private key. You must use a larger public key or an RSA_AES_KEY_WRAP wrapping algorithm.

The following example encrypts key material with the public key that you downloaded and the RSAES_OAEP_SHA_256 wrapping algorithm, and saves it in the `EncryptedKeyMaterial.bin` file.

In this example:

- *`WrappingPublicKey.bin`* is the file that contains the downloaded public wrapping key. If you downloaded the public key from the console, this file is named `wrappingKey_`*`KMS key_key_ID_timestamp`* (for example, `wrappingKey_f44c4e20-f83c-48f4-adc6-a1ef38829760_0809092909`).
- *`PlaintextKeyMaterial.bin`* is the file that contains the key material that you are encrypting, such as `PlaintextKeyMaterial.bin`, `HMAC_384_PlaintextKey.bin`, or `ECC_NIST_P521_PrivateKey.der`.

```
$ openssl pkeyutl \
    -encrypt \
    -in PlaintextKeyMaterial.bin \
    -out EncryptedKeyMaterial.bin \
    -inkey WrappingPublicKey.bin \
    -keyform DER \
    -pubin \
    -pkeyopt rsa_padding_mode:oaep \
    -pkeyopt rsa_oaep_md:sha256 \
    -pkeyopt rsa_mgf1_md:sha256
```

RSA_AES_KEY_WRAP_SHA_1

The RSA_AES_KEY_WRAP_SHA_1 wrapping algorithm involves two encryption operations.

1. Encrypt your key material with an AES symmetric key that you generate and an AES symmetric encryption algorithm.
2. Encrypt the AES symmetric key that you used with the public key that you downloaded and the RSAES_OAEP_SHA_1 wrapping algorithm.

AWS KMS supports RSA_AES_KEY_WRAP_SHA_* wrapping algorithms for all supported types of imported key material and all supported public key specs. The RSA_AES_KEY_WRAP_SHA_* algorithms are the only wrapping algorithms supported for wrapping RSA key material.

The RSA_AES_KEY_WRAP_SHA_1 wrapping algorithm requires OpenSSL version 3.*x* or later.

1. **Generate a 256-bit AES symmetric encryption key**

   This command generates an AES symmetric encryption key consisting of 256 random bits, and saves it in the `aes-key.bin` file

   ```
   # Generate a 32-byte AES symmetric encryption key
   $ openssl rand -out aes-key.bin 32
   ```

2. **Encrypt your key material with the AES symmetric encryption key**

   This command encrypts your key material with the AES symmetric encryption key and saves the encrypted key material in the `key-material-wrapped.bin` file.

   In this example command:

   - *`PlaintextKeyMaterial.bin`* is the file that contains the key material that you are importing, such as `PlaintextKeyMaterial.bin`, `HMAC_384_PlaintextKey.bin`, `RSA_3072_PrivateKey.der`, or `ECC_NIST_P521_PrivateKey.der`.
   - *`aes-key.bin`* is the file that contains 256-bit AES symmetric encryption key that you generated in the previous command.

   ```
   # Encrypt your key material with the AES symmetric encryption key
   $ openssl enc -id-aes256-wrap-pad \
           -K "$(xxd -p < aes-key.bin | tr -d '\n')" \
   ```

```
        -iv A65959A6 \
        -in PlaintextKeyMaterial.bin\
        -out key-material-wrapped.bin
```

3. **Encrypt your AES symmetric encryption key with the public key**

   This command encrypts your AES symmetric encryption key with the public key that you downloaded and the RSAES_OAEP_SHA_1 wrapping algorithm, DER-encodes it, and save it in the `aes-key-wrapped.bin` file.

   In this example command:

   - *`WrappingPublicKey.bin`* is the file that contains the downloaded public wrapping key. If you downloaded the public key from the console, this file is named `wrappingKey_`*`KMS key_key_ID_timestamp`* (for example, `wrappingKey_f44c4e20-f83c-48f4-adc6-a1ef38829760_0809092909`
   - *`aes-key.bin`* is the file that contains 256-bit AES symmetric encryption key that you generated in the first command in this example sequence.

```
# Encrypt your AES symmetric encryption key with the downloaded public key
$ openssl pkeyutl \
    -encrypt \
    -in aes-key.bin \
    -out aes-key-wrapped.bin \
    -inkey WrappingPublicKey.bin \
    -keyform DER \
    -pubin \
    -pkeyopt rsa_padding_mode:oaep \
    -pkeyopt rsa_oaep_md:sha1 \
    -pkeyopt rsa_mgf1_md:sha1
```

4. **Generate the file to import**

   Concatenate the file with the encrypted key material and the file with the encrypted AES key. Save them in the `EncryptedKeyMaterial.bin` file, which is the file that you'll import in the Step 4: Import the key material.

   In this example command:

   - *`key-material-wrapped.bin`* is the file that contains your encrypted key material.

- *aes-key-wrapped.bin* is the file that contains the encrypted AES encryption key.

```
# Combine the encrypted AES key and encrypted key material in a file
$ cat aes-key-wrapped.bin key-material-wrapped.bin > EncryptedKeyMaterial.bin
```

RSA_AES_KEY_WRAP_SHA_256

The RSA_AES_KEY_WRAP_SHA_256 wrapping algorithm involves two encryption operations.

1. Encrypt your key material with an AES symmetric key that you generate and an AES symmetric encryption algorithm.
2. Encrypt the AES symmetric key that you used with the public key that you downloaded and the RSAES_OAEP_SHA_256 wrapping algorithm.

AWS KMS supports RSA_AES_KEY_WRAP_SHA_* wrapping algorithms for all supported types of imported key material and all supported public key specs. The RSA_AES_KEY_WRAP_SHA_* algorithms are the only wrapping algorithms supported for wrapping RSA key material.

The RSA_AES_KEY_WRAP_SHA_256 wrapping algorithm requires OpenSSL version 3.*x* or later.

1. **Generate a 256-bit AES symmetric encryption key**

   This command generates an AES symmetric encryption key consisting of 256 random bits, and saves it in the `aes-key.bin` file

   ```
   # Generate a 32-byte AES symmetric encryption key
   $ openssl rand -out aes-key.bin 32
   ```

2. **Encrypt your key material with the AES symmetric encryption key**

   This command encrypts your key material with the AES symmetric encryption key and saves the encrypted key material in the `key-material-wrapped.bin` file.

   In this example command:

   - *PlaintextKeyMaterial.bin* is the file that contains the key material that you are importing, such as `PlaintextKeyMaterial.bin`, `HMAC_384_PlaintextKey.bin`, `RSA_3072_PrivateKey.der`, or `ECC_NIST_P521_PrivateKey.der`.

- *aes-key.bin* is the file that contains 256-bit AES symmetric encryption key that you generated in the previous command.

```
# Encrypt your key material with the AES symmetric encryption key
$ openssl enc -id-aes256-wrap-pad \
        -K "$(xxd -p < aes-key.bin | tr -d '\n')" \
        -iv A65959A6 \
        -in PlaintextKeyMaterial.bin\
        -out key-material-wrapped.bin
```

3. **Encrypt your AES symmetric encryption key with the public key**

   This command encrypts your AES symmetric encryption key with the public key that you downloaded and the RSAES_OAEP_SHA_256 wrapping algorithm, DER-encodes it, and save it in the `aes-key-wrapped.bin` file.

   In this example command:

   - *WrappingPublicKey.bin* is the file that contains the downloaded public wrapping key. If you downloaded the public key from the console, this file is named `wrappingKey_`*KMS key_key_ID_timestamp* (for example, `wrappingKey_f44c4e20-f83c-48f4-adc6-a1ef38829760_0809092909`
   - *aes-key.bin* is the file that contains 256-bit AES symmetric encryption key that you generated in the first command in this example sequence.

```
# Encrypt your AES symmetric encryption key with the downloaded public key
$ openssl pkeyutl \
    -encrypt \
    -in aes-key.bin \
    -out aes-key-wrapped.bin \
    -inkey WrappingPublicKey.bin \
    -keyform DER \
    -pubin \
    -pkeyopt rsa_padding_mode:oaep \
    -pkeyopt rsa_oaep_md:sha256 \
    -pkeyopt rsa_mgf1_md:sha256
```

4. **Generate the file to import**

   Concatenate the file with the encrypted key material and the file with the encrypted AES key. Save them in the `EncryptedKeyMaterial.bin` file, which is the file that you'll import in the Step 4: Import the key material.

   In this example command:

   - *`key-material-wrapped.bin`* is the file that contains your encrypted key material.
   - *`aes-key-wrapped.bin`* is the file that contains the encrypted AES encryption key.

   ```
   # Combine the encrypted AES key and encrypted key material in a file
   $ cat aes-key-wrapped.bin key-material-wrapped.bin > EncryptedKeyMaterial.bin
   ```

Proceed to Step 4: Import the key material.

Importing key material step 4: Import the key material

After you encrypt your key material, you can import the key material to use with an AWS KMS key. To import key material, you upload the encrypted key material from Step 3: Encrypt the key material and the import token that you downloaded at Step 2: Download the wrapping public key and import token. You must import key material into the same KMS key that you specified when you downloaded the public key and import token. When key material is successfully imported, the key state of the KMS key changes to `Enabled`, and you can use the KMS key in cryptographic operations.

When you import key material, you can set an optional expiration time for the key material. When the key material expires, AWS KMS deletes the key material and the KMS key becomes unusable. To use the KMS key in cryptographic operations, you must reimport the same key material. After you import your key material, you cannot set, change, or cancel the expiration date for the current import. To change these values, you must delete and reimport the same key material.

To import key material, you can use the AWS KMS console or the ImportKeyMaterial API. You can use the API directly by making HTTP requests, or by using an AWS SDKs, AWS Command Line Interface or AWS Tools for PowerShell.

When you import the key material, an ImportKeyMaterial entry is added to your AWS CloudTrail log to record the ImportKeyMaterial operation. The CloudTrail entry is the same whether you use the AWS KMS console or the AWS KMS API.

Setting an expiration time (optional)

When you import the key material for your KMS key, you can set an optional expiration date and time for the key material of up to 365 days from the import date. When imported key material expires, AWS KMS deletes it. This action changes the key state of the KMS key to PendingImport, which prevents it from being used in any cryptographic operation. To use the KMS key, you must reimport a copy of the original key material.

Ensuring that imported key material expires frequently can help you to satisfy regulatory requirements, but it introduces an additional a risk to data encrypted under the KMS key. Until you reimport a copy of the original key material, a KMS key with expired key material is unusable, and any data encrypted under the KMS key is inaccessible. If you fail to reimport the key material for any reason, including losing your copy of the original key material, the KMS key is permanently unusable, and data encrypted under the KMS key is unrecoverable.

To mitigate this risk, make sure that your copy of the imported key material is accessible, and design a system to delete and reimport the key material before it expires and interrupts your AWS workload. We recommend that you set an alarm for the expiration of your imported key material that gives you plenty of time to reimport the key material before it expires. You can also use your CloudTrail logs to audit operations that import (and reimport) key material and delete imported key material, and the AWS KMS operation to delete expired key material.

You cannot import different key material into the KMS key, and AWS KMS cannot restore, recover, or reproduce the deleted key material. Instead of setting an expiration time, you can programmatically delete and reimport the imported key material periodically, but the requirements for retaining a copy of the original key material are the same.

You determine whether and when imported key material expires when you import the key material. But you can turn expiration on and off, or set a new expiration time by deleting and reimporting the key material. Use the ExpirationModel parameter of ImportKeyMaterial to turn expiration on (KEY_MATERIAL_EXPIRES) and off (KEY_MATERIAL_DOES_NOT_EXPIRE) and the ValidTo parameter to set the expiration time. The maximum time is 365 days from the import data; there is no minimum, but the time must be in the future.

Import key material (console)

You can use the AWS Management Console to import key material.

1. If you are on the **Upload your wrapped key material** page, skip to Step 8.
2. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
3. To change the AWS Region, use the Region selector in the upper-right corner of the page.
4. In the navigation pane, choose **Customer managed keys**.
5. Choose the key ID or alias of the KMS key for which you downloaded the public key and import token.
6. Choose the **Cryptographic configuration** tab and view its values. The tabs are on the detail page for a KMS key below the **General configuration** section.

   You can only import key material into KMS keys with an **Origin** of **External (Import key material)**. For information about creating KMS keys with imported key material, see Importing key material for AWS KMS keys.
7. Choose the **Key material** tab and then choose **Import key material**. The **Key material** tab appears only for KMS keys with an **Origin** value of **External (Import key material)**.

   If you downloaded the key material, import token, and encrypted the key material, choose **Next**.
8. In the **Encrypted key material and import token** section, do the following.

   a. Under **Wrapped key material**, choose **Choose file**. Then upload the file that contains your wrapped (encrypted) key material.

   b. Under **Import token**, choose **Choose file**. Upload the file that contains the import token that you downloaded.
9. In the **Expiration option** section, you determine whether the key material expires. To set an expiration date and time, choose **Key material expires**, and use the calendar to select a date and time. You can specify a date up to 365 days from the current date and time.
10. Choose **Upload key material**.

Import key material (AWS KMS API)

To import key material, use the ImportKeyMaterial operation. The following example uses the AWS CLI, but you can use any supported programming language.

To use this example:

1. Replace *1234abcd-12ab-34cd-56ef-1234567890ab* with a key ID of the KMS key that you specified when you downloaded the public key and import token. To identify the KMS key, use its key ID or key ARN. You cannot use an alias name or alias ARN for this operation.
2. Replace *EncryptedKeyMaterial.bin* with the name of the file that contains the encrypted key material.
3. Replace *ImportToken.bin* with the name of the file that contains the import token.
4. If you want the imported key material to expire, set the value of the `expiration-model` parameter to its default value, `KEY_MATERIAL_EXPIRES`, or omit the `expiration-model` parameter. Then, replace the value of the `valid-to` parameter with the date and time that you want the key material to expire. The date and time can be up to 365 days from the time of the request.

```
$ aws kms import-key-material --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
    --encrypted-key-material fileb://EncryptedKeyMaterial.bin \
    --import-token fileb://ImportToken.bin \
    --expiration-model KEY_MATERIAL_EXPIRES \
    --valid-to 2023-06-17T12:00:00-08:00
```

If you do not want the imported key material to expire, set the value of the `expiration-model` parameter to `KEY_MATERIAL_DOES_NOT_EXPIRE` and omit the `valid-to` parameter from the command.

```
$ aws kms import-key-material --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
    --encrypted-key-material fileb://EncryptedKeyMaterial.bin \
    --import-token fileb://ImportToken.bin \
    --expiration-model KEY_MATERIAL_DOES_NOT_EXPIRE
```

Tip

If the command does not succeed, you might see a `KMSInvalidStateException` or a `NotFoundException`. You can retry the request.

Custom key stores

A *key store* is a secure location for storing cryptographic keys. The default key store in AWS KMS also supports methods for generating and managing the keys that it stores. By default, the cryptographic key material for the AWS KMS keys that you create in AWS KMS is generated in and protected by hardware security modules (HSMs) that are FIPS 140-2 validated cryptographic modules. Key material for your KMS keys never leave the HSMs unencrypted.

However, if you require even more control of the HSMs, you can create a custom key store.

A *custom key store* is a logical key store within AWS KMS that is backed by a key manager outside of AWS KMS that you own and manage. Custom key stores combine the convenient and comprehensive key management interface of AWS KMS with the ability to own and control the key material and cryptographic operations. When you use a KMS key in a custom key store, the cryptographic operations are performed by your key manager using your cryptographic keys. As a result, you assume more responsibility for the availability and durability of cryptographic keys, and for the operation of the HSMs.

AWS KMS supports two types of custom key stores.

- An AWS CloudHSM key store is an AWS KMS custom key store backed by an AWS CloudHSM cluster. When you create a KMS key in your AWS CloudHSM key store, AWS KMS generates a 256-bit, persistent, non-exportable Advanced Encryption Standard (AES) symmetric key in the associated AWS CloudHSM cluster. This key material never leaves your AWS CloudHSM clusters unencrypted. When you use a KMS key in AWS CloudHSM key store, the cryptographic operations are performed in the HSMs in the cluster. AWS CloudHSM clusters are backed by hardware security modules (HSMs) certified at FIPS 140-2 Level 3.
- An external key store is an AWS KMS custom key store backed by an external key manager outside of AWS that you own and control. When you use a KMS key in your external key store, all encryption and decryption operations are performed by your external key manager using your cryptographic keys. External key stores are designed to support a variety of external key managers from different vendors.

AWS KMS never directly views, accesses, or interacts with your external key manager or cryptographic keys. When you encrypt or decrypt with a KMS key in an external key store, the operation is performed by your external key manager using your external keys. You retain full control over your cryptographic keys, including the ability to refuse or halt a cryptographic operation without interacting with AWS. However, due to distance and extra processing, KMS keys in an external key store might have poorer latency and performance, and might have different availability characteristics than KMS keys with key material in AWS KMS. For more information about key managers compatible with the AWS KMS external key store feature, see Which external vendors support the XKS Proxy specification? in the *AWS Key Management Service FAQs*.

These two types of custom key stores are quite different from the standard AWS KMS key store and from each other. Their security models, focus of responsibility, performance, price, and the use cases are also very different. Before choosing a custom key store, read the related documentation and confirm that the extra configuration and maintenance responsibility is a wise trade-off for the extra control. However, if the rules and regulations under which you operate require direct control of key material, a custom key store might be a good choice for you.

Unsupported features

AWS KMS does not support the following features in custom key stores.

- Asymmetric KMS keys
- Asymmetric data key pairs
- HMAC KMS keys
- KMS keys with imported key material
- Automatic key rotation
- Multi-Region keys

Topics

AWS CloudHSM key stores

An AWS CloudHSM key store is a custom key store backed by a AWS CloudHSM cluster. When you create an AWS KMS key in a custom key store, AWS KMS generates and stores non-extractable key material for the KMS key in an AWS CloudHSM cluster that you own and manage. When you use a KMS key in a custom key store, the cryptographic operations are performed in the HSMs in the cluster. This feature combines the convenience and widespread integration of AWS KMS with the added control of an AWS CloudHSM cluster in your AWS account.

AWS KMS provides full console and API support for creating, using, and managing your custom key stores. You can use the KMS keys in your custom key store the same way that you use any KMS key. For example, you can use the KMS keys to generate data keys and encrypt data. You can also use the KMS keys in your custom key store with AWS services that support customer managed keys.

Do I need a custom key store?

For most users, the default AWS KMS key store, which is protected by FIPS 140-2 validated cryptographic modules, fulfills their security requirements. There is no need to add an extra layer of maintenance responsibility or a dependency on an additional service.

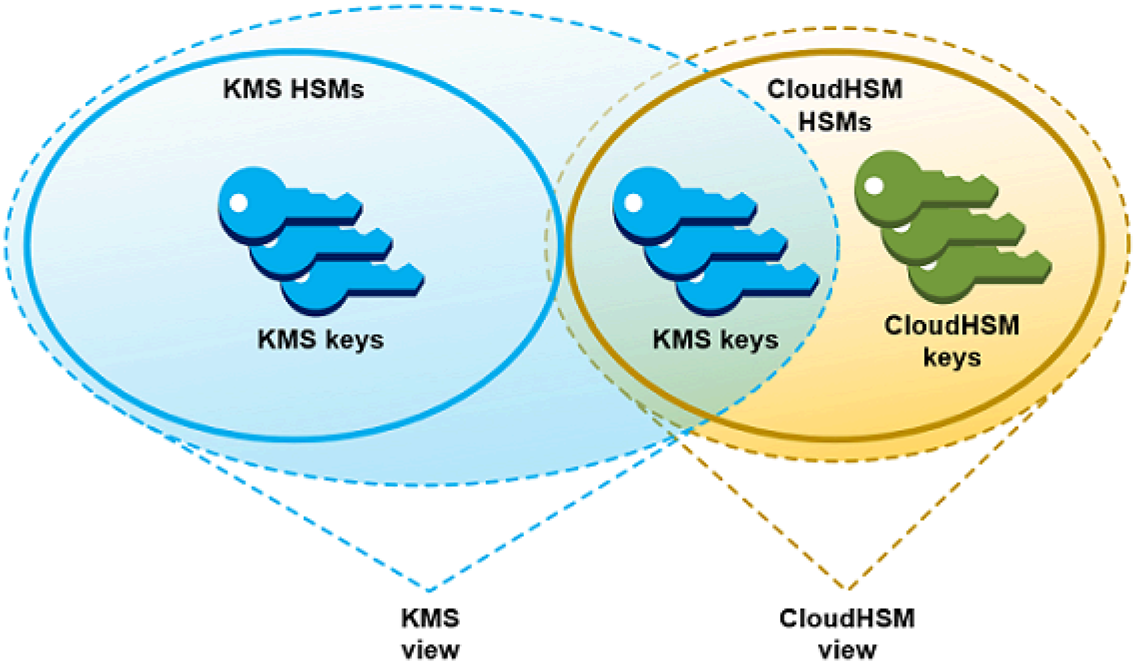
However, you might consider creating a custom key store if your organization has any of the following requirements:

- You have keys that are explicitly required to be protected in a single tenant HSM or in an HSM that you have direct control over.
- You need the ability to immediately remove key material from AWS KMS.
- You need to be able to audit all use of your keys independently of AWS KMS or AWS CloudTrail.

How do custom key stores work?

Each custom key store is associated with an AWS CloudHSM cluster in your AWS account. When you connect the custom key store to its cluster, AWS KMS creates the network infrastructure to support the connection. Then it logs into the key AWS CloudHSM client in the cluster using the credentials of a dedicated crypto user in the cluster.

You create and manage your custom key stores in AWS KMS and create and manage your HSM clusters in AWS CloudHSM. When you create AWS KMS keys in an AWS KMS custom key store, you view and manage the KMS keys in AWS KMS. But you can also view and manage their key material in AWS CloudHSM, just as you would do for other keys in the cluster.



You can create symmetric encryption KMS keys with key material generated by AWS KMS in your custom key store. Then use the same techniques to view and manage the KMS keys in your custom key store that you use for KMS keys in the AWS KMS key store. You can control access with IAM and key policies, create tags and aliases, enable and disable the KMS keys, and schedule key deletion. You can use the KMS keys for cryptographic operations and use them with AWS services that integrate with AWS KMS.

In addition, you have full control over the AWS CloudHSM cluster, including creating and deleting HSMs and managing backups. You can use the AWS CloudHSM client and supported software libraries to view, audit, and manage the key material for your KMS keys. While the custom key store is disconnected, AWS KMS cannot access it, and users cannot use the KMS keys in the custom key store for cryptographic operations. This added layer of control makes custom key stores a powerful solution for organizations that require it.

Where do I start?

To create and manage an AWS CloudHSM key store, you use features of AWS KMS and AWS CloudHSM.

1. Start in AWS CloudHSM. Create an active AWS CloudHSM cluster or select an existing cluster. The cluster must have at least two active HSMs in different Availability Zones. Then create a dedicated crypto user (CU) account in that cluster for AWS KMS.

2. In AWS KMS, create a custom key store that is associated with your selected AWS CloudHSM cluster. AWS KMS provides a complete management interface that lets you create, view, edit, and delete your custom key stores.
3. When you're ready to use your custom key store, connect it to its associated AWS CloudHSM cluster. AWS KMS creates the network infrastructure that it needs to support the connection. It then logs in to the cluster using the dedicated crypto user account credentials so it can generate and manage key material in the cluster.
4. Now, you can create symmetric encryption KMS keys in your custom key store. Just specify the custom key store when you create the KMS key.

If you get stuck at any point, you can find help in the Troubleshooting a custom key store topic. If your question is not answered, use the feedback link at the bottom of each page of this guide or post a question on the AWS Key Management Service Discussion Forum.

Quotas

AWS KMS allows up to 10 custom key stores in each AWS account and Region, including both AWS CloudHSM key stores and external key stores, regardless of their connection state. In addition, there are AWS KMS request quotas on the use of KMS keys in an AWS CloudHSM key store.

Pricing

For information on the cost of AWS KMS custom key stores and customer managed keys in a custom key store, see AWS Key Management Service pricing. For information about the cost of AWS CloudHSM clusters and HSMs, see AWS CloudHSM Pricing.

Regions

AWS KMS supports AWS CloudHSM key stores in all AWS Regions where AWS KMS is supported, except for Asia Pacific (Melbourne), China (Beijing), China (Ningxia), and Europe (Spain).

Unsupported features

AWS KMS does not support the following features in custom key stores.

- Asymmetric KMS keys
- Asymmetric data key pairs
- HMAC KMS keys

- KMS keys with imported key material
- Automatic key rotation
- Multi-Region keys

Topics

AWS CloudHSM key store concepts

This topic explains some of the concepts used in AWS CloudHSM key stores.

AWS CloudHSM key store

An *AWS CloudHSM key store* is a custom key store associated with an AWS CloudHSM cluster that you own and manage. AWS CloudHSM clusters are backed by hardware security modules (HSMs) certified at FIPS 140-2 Level 3.

When you create a KMS key in your AWS CloudHSM key store, AWS KMS generates a 256-bit, persistent, non-exportable Advanced Encryption Standard (AES) symmetric key in the associated AWS CloudHSM cluster. This key material never leaves your HSMs unencrypted. When you use a KMS key in an AWS CloudHSM key store, the cryptographic operations are performed in the HSMs in the cluster.

AWS CloudHSM key stores combine the convenient and comprehensive key management interface of AWS KMS with the additional controls provided by an AWS CloudHSM cluster in your AWS account. This integrated feature lets you create, manage, and use KMS keys in AWS KMS while maintaining full control of the HSMs that store their key material, including managing clusters, HSMs, and backups. You can use the AWS KMS console and APIs to manage the AWS CloudHSM key store and its KMS keys. You can also use the AWS CloudHSM console, APIs, client software, and associated software libraries to manage the associated cluster.

You can view and manage your AWS CloudHSM key store, edit its properties, and connect and disconnect it from its associated AWS CloudHSM cluster. If you need to delete an AWS CloudHSM key store, you must first delete the KMS keys in the AWS CloudHSM key store by scheduling their deletion and waiting until the grace period expires. Deleting the AWS CloudHSM key store removes the resource from AWS KMS, but it does not affect your AWS CloudHSM cluster.

AWS CloudHSM cluster

Every AWS CloudHSM key store is associated with one AWS CloudHSM cluster. When you create an AWS KMS key in your AWS CloudHSM key store, AWS KMS creates its key material in the associated cluster. When you use a KMS key in your AWS CloudHSM key store, the cryptographic operation is performed in the associated cluster.

Each AWS CloudHSM cluster can be associated with only one AWS CloudHSM key store. The cluster that you choose cannot be associated with another AWS CloudHSM key store or share a backup history with a cluster that is associated with another AWS CloudHSM key store. The cluster must be initialized and active, and it must be in the same AWS account and Region as the AWS CloudHSM key store. You can create a new cluster or use an existing one. AWS KMS does not need exclusive use of the cluster. To create KMS keys in the AWS CloudHSM key store, its associated cluster it must contain at least two active HSMs. All other operations require only one HSM.

You specify the AWS CloudHSM cluster when you create the AWS CloudHSM key store, and you cannot change it. However, you can substitute any cluster that shares a backup history with the original cluster. This lets you delete the cluster, if necessary, and replace it with a cluster created from one of its backups. You retain full control of the associated AWS CloudHSM cluster so you can manage users and keys, create and delete HSMs, and use and manage backups.

When you are ready to use your AWS CloudHSM key store, you connect it to its associated AWS CloudHSM cluster. You can connect and disconnect your custom key store at any time. When a custom key store is connected, you can create and use its KMS keys. When it is disconnected, you can view and manage the AWS CloudHSM key store and its KMS keys. But you cannot create new KMS keys or use the KMS keys in the AWS CloudHSM key store for cryptographic operations.

`kmsuser` Crypto user

To create and manage key material in the associated AWS CloudHSM cluster on your behalf, AWS KMS uses a dedicated AWS CloudHSM crypto user (CU) in the cluster named `kmsuser`. The `kmsuser` CU is a standard CU account that is automatically synchronized to all HSMs in the cluster and is saved in cluster backups.

Before you create your AWS CloudHSM key store, you create a `kmsuser` CU account in your AWS CloudHSM cluster using the createUser command in cloudhsm_mgmt_util. Then when you create the AWS CloudHSM key store, you provide the kmsuser account password to AWS KMS. When you connect the custom key store, AWS KMS logs into the cluster as the kmsuser CU and rotates its password. AWS KMS encrypts your kmsuser password before it stores it securely. When the password is rotated, the new password is encrypted and stored in the same way.

AWS KMS remains logged in as kmsuser as long as the AWS CloudHSM key store is connected. You should not use this CU account for other purposes. However, you retain ultimate control of the kmsuser CU account. At any time, you can find the key handles of keys that kmsuser owns. If necessary, you can disconnect the custom key store, change the kmsuser password, log into the cluster as kmsuser, and view and manage the keys that kmsuser owns.

For instructions on creating your kmsuser CU account, see Create the kmsuser Crypto User.

KMS keys in an AWS CloudHSM key store

You can use the AWS KMS or AWS KMS API to create a AWS KMS keys in an AWS CloudHSM key store. You use the same technique that you would use on any KMS key. The only difference is that you must identify the AWS CloudHSM key store and specify that the origin of the key material is the AWS CloudHSM cluster.

When you create a KMS key in an AWS CloudHSM key store, AWS KMS creates the KMS key in AWS KMS and it generates a 256-bit, persistent, non-exportable Advanced Encryption Standard (AES) symmetric key material in its associated cluster. When you use the AWS KMS key in a cryptographic operation, the operation is performed in the AWS CloudHSM cluster using the cluster-based AES key. Although AWS CloudHSM supports symmetric and asymmetric keys of different types, AWS CloudHSM key stores support only AES symmetric encryption keys.

You can view the KMS keys in an AWS CloudHSM key store in the AWS KMS console, and use the console options to display the custom key store ID. You can also use the DescribeKey operation to find the AWS CloudHSM key store ID and AWS CloudHSM cluster ID.

The KMS keys in an AWS CloudHSM key store work just like any KMS keys in AWS KMS. Authorized users need the same permissions to use and manage the KMS keys. You use the same console procedures and API operations to view and manage the KMS keys in an AWS CloudHSM key store. These include enabling and disabling KMS keys, creating and using tags and aliases, and setting and changing IAM and key policies. You can use the KMS keys in an AWS CloudHSM key store for cryptographic operations, and use them with integrated AWS services that support the use of customer managed keys However, you cannot enable automatic key rotation or import key material into a KMS key in an AWS CloudHSM key store.

You also use the same process to schedule deletion of a KMS key in an AWS CloudHSM key store. After the waiting period expires, AWS KMS deletes the KMS key from KMS. Then it makes a best effort to delete the key material for the KMS key from the associated AWS CloudHSM cluster. However, you might need to manually delete the orphaned key material from the cluster and its backups.

Controlling access to your AWS CloudHSM key store

You use IAM policies to control access to your AWS CloudHSM key store and your AWS CloudHSM cluster. You can use key policies, IAM policies, and grants to control access to the AWS KMS keys in your AWS CloudHSM key store. We recommend that you provide users, groups, and roles only the permissions that they require for the tasks that they are likely to perform.

Topics

Authorizing AWS CloudHSM key store managers and users

When designing your AWS CloudHSM key store, be sure that the principals who use and manage it have only the permissions that they require. The following list describes the minimum permissions required for AWS CloudHSM key store managers and users.

- Principals who create and manage your AWS CloudHSM key store require the following permission to use the AWS CloudHSM key store API operations.
  - `cloudhsm:DescribeClusters`
  - `kms:CreateCustomKeyStore`
  - `kms:ConnectCustomKeyStore`
  - `kms:DeleteCustomKeyStore`
  - `kms:DescribeCustomKeyStores`
  - `kms:DisconnectCustomKeyStore`
  - `kms:UpdateCustomKeyStore`
  - `iam:CreateServiceLinkedRole`
- Principals who create and manage the AWS CloudHSM cluster that is associated with your AWS CloudHSM key store need permission to create and initialize an AWS CloudHSM cluster. This includes permission to create or use an Amazon Virtual Private Cloud (VPC), create subnets, and

create an Amazon EC2 instance. They might also need to create and delete HSMs, and manage backups. For lists of the required permissions, see Identity and access management for AWS CloudHSM in the *AWS CloudHSM User Guide*.

- Principals who create and manage AWS KMS keys in your AWS CloudHSM key store require the same permissions as those who create and manage any KMS key in AWS KMS. The default key policy for a KMS key in an AWS CloudHSM key store is identical to the default key policy for KMS keys in AWS KMS. Attribute-based access control (ABAC), which uses tags and aliases to control access to KMS keys, is also effective on KMS keys in AWS CloudHSM key stores.
- Principals who use the KMS keys in your AWS CloudHSM key store for cryptographic operations need permission to perform the cryptographic operation with the KMS key, such as kms:Decrypt. You can provide these permissions in a key policy, IAM policy. But, they do not need any additional permissions to use a KMS key in an AWS CloudHSM key store.

Authorizing AWS KMS to manage AWS CloudHSM and Amazon EC2 resources

To support your AWS CloudHSM key stores, AWS KMS needs permission to get information about your AWS CloudHSM clusters. It also needs permission to create the network infrastructure that connects your AWS CloudHSM key store to its AWS CloudHSM cluster. To get these permissions, AWS KMS creates the **AWSServiceRoleForKeyManagementServiceCustomKeyStores** service-linked role in your AWS account. Users who create AWS CloudHSM key stores must have the `iam:CreateServiceLinkedRole` permission that allows them to create service-linked roles.

Topics

About the AWS KMS service-linked role

A service-linked role is an IAM role that gives one AWS service permission to call other AWS services on your behalf. It's designed to make it easier for you to use the features of multiple integrated AWS services without having to create and maintain complex IAM policies. For more information, see Using service-linked roles for AWS KMS.

For AWS CloudHSM key stores, AWS KMS creates the **AWSServiceRoleForKeyManagementServiceCustomKeyStores** service-linked role with the **AWSKeyManagementServiceCustomKeyStoresServiceRolePolicy** policy. This policy grants the role the following permissions:

- cloudhsm:Describe* – detects changes in the AWS CloudHSM cluster that is attached to your custom key store.
- ec2:CreateSecurityGroup – used when you connect an AWS CloudHSM key store to create the security group that enables network traffic flow between AWS KMS and your AWS CloudHSM cluster.
- ec2:AuthorizeSecurityGroupIngress – used when you connect an AWS CloudHSM key store to allow network access from AWS KMS into the VPC that contains your AWS CloudHSM cluster.
- ec2:CreateNetworkInterface – used when you connect an AWS CloudHSM key store to create the network interface used for communication between AWS KMS and the AWS CloudHSM cluster.
- ec2:RevokeSecurityGroupEgress – used when you connect an AWS CloudHSM key store to remove all outbound rules from the security group that AWS KMS created.
- ec2:DeleteSecurityGroup – used when you disconnect an AWS CloudHSM key store to delete security groups that were created when you connected the AWS CloudHSM key store.
- ec2:DescribeSecurityGroups – used to monitor changes in the security group that AWS KMS created in the VPC that contains your AWS CloudHSM cluster so that AWS KMS can provide clear error messages in case of failures.
- ec2:DescribeVpcs – used to monitor changes in the VPC that contains your AWS CloudHSM cluster so that AWS KMS can provide clear error messages in case of failures.
- ec2:DescribeNetworkAcls – used to monitor changes in the network ACLs for the VPC that contains your AWS CloudHSM cluster so that AWS KMS can provide clear error messages in case of failures.
- ec2:DescribeNetworkInterfaces – used to monitor changes in the network interfaces that AWS KMS created in the VPC that contains your AWS CloudHSM cluster so that AWS KMS can provide clear error messages in case of failures.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudhsm:Describe*",
        "ec2:CreateNetworkInterface",
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:CreateSecurityGroup",
        "ec2:DescribeSecurityGroups",
        "ec2:RevokeSecurityGroupEgress",
        "ec2:DeleteSecurityGroup",
        "ec2:DescribeVpcs",
        "ec2:DescribeNetworkAcls",
        "ec2:DescribeNetworkInterfaces"
      ],
      "Resource": "*"
    }
  ]
}
```

Because the **AWSServiceRoleForKeyManagementServiceCustomKeyStores** service-linked role trusts only `cks.kms.amazonaws.com`, only AWS KMS can assume this service-linked role. This role is limited to the operations that AWS KMS needs to view your AWS CloudHSM clusters and to connect an AWS CloudHSM key store to its associated AWS CloudHSM cluster. It does not give AWS KMS any additional permissions. For example, AWS KMS does not have permission to create, manage, or delete your AWS CloudHSM clusters, HSMs, or backups.

Regions

Like the AWS CloudHSM key stores feature, the **AWSServiceRoleForKeyManagementServiceCustomKeyStores** role is supported in all AWS Regions where AWS KMS and AWS CloudHSM are available. For a list of AWS Regions that each service supports, see AWS Key Management Service Endpoints and Quotas and AWS CloudHSM endpoints and quotas in the *Amazon Web Services General Reference*.

For more information about how AWS services use service-linked roles, see Using service-linked roles in the IAM User Guide.

Create the service-linked role

AWS KMS automatically creates the **AWSServiceRoleForKeyManagementServiceCustomKeyStores** service-linked role in your AWS account when you create an AWS CloudHSM key store, if the role does not already exist. You cannot create or re-create this service-linked role directly.

Edit the service-linked role description

You cannot edit the role name or the policy statements in the **AWSServiceRoleForKeyManagementServiceCustomKeyStores** service-linked role, but you can edit role description. For instructions, see Editing a service-linked role in the *IAM User Guide*.

Delete the service-linked role

AWS KMS does not delete the **AWSServiceRoleForKeyManagementServiceCustomKeyStores** service-linked role from your AWS account even if you have deleted all of your AWS CloudHSM key stores. Although there is currently no procedure for deleting the **AWSServiceRoleForKeyManagementServiceCustomKeyStores** service-linked role, AWS KMS does not assume this role or use its permissions unless you have active AWS CloudHSM key stores.

Managing a CloudHSM custom key store

Using the AWS Management Console and the AWS KMS API, you can manage a custom key store. For example, you can view a custom key store, edit its properties, connect and disconnect it from its associated AWS CloudHSM cluster, and delete the custom key store.

Topics

Creating an AWS CloudHSM key store

You can create one or several AWS CloudHSM key stores in your account. Each AWS CloudHSM key store is associated with one AWS CloudHSM cluster in the same AWS account and Region. Before you create your AWS CloudHSM key store, you need to assemble the prerequisites. Then, before you can use your AWS CloudHSM key store, you must connect it to its AWS CloudHSM cluster.

> **Note**
>
> If you try to create an AWS CloudHSM key store with all of the same property values as an existing *disconnected* AWS CloudHSM key store, AWS KMS does not create a new AWS

CloudHSM key store, and it does not throw an exception or display an error. Instead, AWS KMS recognizes the duplicate as the likely consequence of a retry, and it returns the ID of the existing AWS CloudHSM key store.

Tip

You do not have to connect your AWS CloudHSM key store immediately. You can leave it in a disconnected state until you are ready to use it. However, to verify that it is configured properly, you might want to connect it, view its connection state, and then disconnect it.

Topics

- Assemble the prerequisites
- Create an AWS CloudHSM key store (console)
- Create an AWS CloudHSM key store (API)

Assemble the prerequisites

Each AWS CloudHSM key store is backed by an AWS CloudHSM cluster. To create an AWS CloudHSM key store, you must specify an active AWS CloudHSM cluster that is not already associated with another key store. You also need to create a dedicated crypto user (CU) in the cluster's HSMs that AWS KMS can use to create and manage keys on your behalf.

Before you create an AWS CloudHSM key store, do the following:

Select an AWS CloudHSM cluster

Every AWS CloudHSM key store is associated with exactly one AWS CloudHSM cluster. When you create a AWS KMS keys in your AWS CloudHSM key store, AWS KMS creates the KMS key metadata, such as an ID and Amazon Resource Name (ARN) in AWS KMS. It then creates the key material in the HSMs of the associated cluster. You can create a new AWS CloudHSM cluster or use an existing one. AWS KMS does not require exclusive access to the cluster.

The AWS CloudHSM cluster that you select is permanently associated with the AWS CloudHSM key store. After you create the AWS CloudHSM key store, you can change the cluster ID of the associated cluster, but the cluster that you specify must share a backup history with the original cluster. To use an unrelated cluster, you need to create a new AWS CloudHSM key store.

The AWS CloudHSM cluster that you select must have the following characteristics:

- **The cluster must be active**.

  You must create the cluster, initialize it, install the AWS CloudHSM client software for your platform, and then activate the cluster. For detailed instructions, see Getting started with AWS CloudHSM in the *AWS CloudHSM User Guide*.
- **The cluster must be in the same account and Region** as the AWS CloudHSM key store. You cannot associate an AWS CloudHSM key store in one Region with a cluster in a different Region. To create a key infrastructure in multiple Regions, you must create AWS CloudHSM key stores and clusters in each Region.
- **The cluster cannot be associated with another custom key store** in the same account and Region. Each AWS CloudHSM key store in the account and Region must be associated with a different AWS CloudHSM cluster. You cannot specify a cluster that is already associated with a custom key store or a cluster that shares a backup history with an associated cluster. Clusters that share a backup history have the same cluster certificate. To view the cluster certificate of a cluster, use the AWS CloudHSM console or the DescribeClusters operation.

  If you back up an AWS CloudHSM cluster to a different Region, it is considered to be different cluster, and you can associate the backup with a custom key store in its Region. However, KMS keys in the two custom key stores are not interoperable, even if they have the same backing key. AWS KMS binds metadata to the ciphertext so it can be decrypted only by the KMS key that encrypted it.
- The cluster must be configured with private subnets in **at least two Availability Zones** in the Region. Because AWS CloudHSM is not supported in all Availability Zones, we recommend that you create private subnets in all Availability Zones in the region. You cannot reconfigure the subnets for an existing cluster, but you can create a cluster from a backup with different subnets in the cluster configuration.

  > ⚠ **Important**
  >
  > After you create your AWS CloudHSM key store, do not delete any of the private subnets configured for its AWS CloudHSM cluster. If AWS KMS cannot find all of the subnets in the cluster configuration, attempts to connect to the custom key store fail with a `SUBNET_NOT_FOUND` connection error state. For details, see How to fix a connection failure.

- The security group for the cluster (`cloudhsm-cluster-<cluster-id>-sg`) must include inbound rules and outbound rules that allow TCP traffic on ports 2223-2225. The **Source** in the inbound rules and the **Destination** in the outbound rules must match the security group ID. These rules are set by default when you create the cluster. Do not delete or change them.
- **The cluster must contain at least two active HSMs** in different Availability Zones. To verify the number of HSMs, use the AWS CloudHSM console or the DescribeClusters operation. If necessary, you can add an HSM.

Find the trust anchor certificate

When you create a custom key store, you must upload the trust anchor certificate for the AWS CloudHSM cluster to AWS KMS. AWS KMS needs the trust anchor certificate to connect the AWS CloudHSM key store to its associated AWS CloudHSM cluster.

Every active AWS CloudHSM cluster has a *trust anchor certificate*. When you initialize the cluster, you generate this certificate, save it in the `customerCA.crt` file, and copy it to hosts that connect to the cluster.

Create the `kmsuser` crypto user for AWS KMS

To administer your AWS CloudHSM key store, AWS KMS logs into the `kmsuser` crypto user (CU) account in the selected cluster. Before you create your AWS CloudHSM key store, you must create the `kmsuser` CU. Then when you create your AWS CloudHSM key store, you provide the password for `kmsuser` to AWS KMS. Whenever you connect the AWS CloudHSM key store to its associated AWS CloudHSM cluster, AWS KMS logs in as the `kmsuser` and rotates the `kmsuser` password

> ⚠ **Important**
>
> Do not specify the `2FA` option when you create the `kmsuser` CU. If you do, AWS KMS cannot log in and your AWS CloudHSM key store cannot be connected to this AWS CloudHSM cluster. Once you specify 2FA, you cannot undo it. Instead, you must delete the CU and recreate it.

To create the `kmsuser` CU, use the following procedure.

1. Start cloudhsm_mgmt_util as described in the Getting started with CloudHSM Management Utility (CMU) topic of the *AWS CloudHSM User Guide*.

2. Use the createUser command in cloudhsm_mgmt_util to create a CU named `kmsuser`. The password must consist of 7-32 alphanumeric characters. It is case-sensitive and cannot contain any special characters.

   For example, the following example command creates a `kmsuser` CU with a password of `kmsPswd`.

   ```
   aws-cloudhsm> createUser CU kmsuser kmsPswd
   ```

Create an AWS CloudHSM key store (console)

When you create an AWS CloudHSM key store in the AWS Management Console, you can add and create the prerequisites as part of your workflow. However, the process is quicker when you have assembled them in advance.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Custom key stores**, **AWS CloudHSM key stores**.
4. Choose **Create a key store**.
5. Enter a friendly name for the custom key store. The name must be unique among all custom key stores in your account.

   ⚠ **Important**

   Do not include confidential or sensitive information in this field. This field may be displayed in plaintext in CloudTrail logs and other output.

6. Select an AWS CloudHSM cluster for the AWS CloudHSM key store. Or, to create a new AWS CloudHSM cluster, choose the **Create an AWS CloudHSM cluster** link.

   The menu displays the AWS CloudHSM clusters in your account and region that are not already associated with an AWS CloudHSM key store. The cluster must fulfill the requirements for association with a custom key store.

7. Choose **Choose file**, and then upload the trust anchor certificate for the AWS CloudHSM cluster that you chose. This is the `customerCA.crt` file that you created when you initialized the cluster.

8. Enter the password of the `kmsuser` crypto user (CU) that you created in the selected cluster.

9. Choose **Create**.

When the procedure is successful, the new AWS CloudHSM key store appears in the list of AWS CloudHSM key stores in the account and Region. If it is unsuccessful, an error message appears that describes the problem and provides help on how to fix it. If you need more help, see Troubleshooting a custom key store.

If you try to create an AWS CloudHSM key store with all of the same property values as an existing *disconnected* AWS CloudHSM key store, AWS KMS does not create a new AWS CloudHSM key store, and it does not throw an exception or display an error. Instead, AWS KMS recognizes the duplicate as the likely consequence of a retry, and it returns the ID of the existing AWS CloudHSM key store.

Next: New AWS CloudHSM key stores are not automatically connected. Before you can create AWS KMS keys in the AWS CloudHSM key store, you must connect the custom key store to its associated AWS CloudHSM cluster.

Create an AWS CloudHSM key store (API)

You can use the CreateCustomKeyStore operation to create a new AWS CloudHSM key store that is associated with an AWS CloudHSM cluster in the account and Region. These examples use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

The `CreateCustomKeyStore` operation requires the following parameter values.

- CustomKeyStoreName – A friendly name for the custom key store that is unique in the account.

> ⚠ **Important**
>
> Do not include confidential or sensitive information in this field. This field may be displayed in plaintext in CloudTrail logs and other output.

- CloudHsmClusterId – The cluster ID of an AWS CloudHSM cluster that fulfills the requirements for an AWS CloudHSM key store.
- KeyStorePassword – The password of `kmsuser` CU account in the specified cluster.
- TrustAnchorCertificate – The content of the `customerCA.crt` file that you created when you initialized the cluster.

The following example uses a fictitious cluster ID. Before running the command, replace it with a valid cluster ID.

```
$ aws kms create-custom-key-store
        --custom-key-store-name ExampleCloudHSMKeyStore \
        --cloud-hsm-cluster-id cluster-1a23b4cdefg \
        --key-store-password kmsPswd \
        --trust-anchor-certificate <certificate-goes-here>
```

If you are using the AWS CLI, you can specify the trust anchor certificate file, instead of its contents. In the following example, the `customerCA.crt` file is in the root directory.

```
$ aws kms create-custom-key-store
        --custom-key-store-name ExampleCloudHSMKeyStore \
        --cloud-hsm-cluster-id cluster-1a23b4cdefg \
        --key-store-password kmsPswd \
        --trust-anchor-certificate file://customerCA.crt
```

When the operation is successful, `CreateCustomKeyStore` returns the custom key store ID, as shown in the following example response.

```
{
    "CustomKeyStoreId": cks-1234567890abcdef0
}
```

If the operation fails, correct the error indicated by the exception, and try again. For additional help, see Troubleshooting a custom key store.

If you try to create an AWS CloudHSM key store with all of the same property values as an existing *disconnected* AWS CloudHSM key store, AWS KMS does not create a new AWS CloudHSM key store, and it does not throw an exception or display an error. Instead, AWS KMS recognizes the duplicate as the likely consequence of a retry, and it returns the ID of the existing AWS CloudHSM key store.

Next: To use the AWS CloudHSM key store, connect it to its AWS CloudHSM cluster.

Viewing an AWS CloudHSM key store

You can view the AWS CloudHSM key stores in each account and Region by using the AWS KMS console or the DescribeCustomKeyStores operation.

See also:

- Viewing an external key store
- Viewing KMS keys in an AWS CloudHSM key store
- Logging AWS KMS API calls with AWS CloudTrail

Topics

View an AWS CloudHSM key store (console)

When you view the AWS CloudHSM key stores in the AWS Management Console, you can see the following:

- The custom key store name and ID
- The ID of associated AWS CloudHSM cluster
- The number of HSMs in the cluster
- The current connection state

A connection state (**Status**) value of **Disconnected** indicates that the custom key store is new and has never been connected, or it was intentionally disconnected from its AWS CloudHSM cluster. However, if your attempts to use a KMS key in a connected custom key store fail, that might indicate a problem with the custom key store or its AWS CloudHSM cluster. For help, see How to fix a failing KMS key.

To view the AWS CloudHSM key stores in a given account and Region, use the following procedure.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Custom key stores**, **AWS CloudHSM key stores**.

To customize the display, click the gear icon that appears below the **Create key store** button.

View an AWS CloudHSM key store (API)

To view your AWS CloudHSM key stores, use the DescribeCustomKeyStores operation. By default, this operation returns all custom key stores in the account and Region. But you can use either the `CustomKeyStoreId` or `CustomKeyStoreName` parameter (but not both) to limit the output to a particular custom key store. For AWS CloudHSM key stores, the output consists of the custom key store ID and name, the custom key store type, the ID of the associated AWS CloudHSM cluster, and the connection state. If the connection state indicates an error, the output also includes an error code that describes the reason for the error.

The examples in this section use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

For example, the following command returns all custom key stores in the account and Region. You can use the `Limit` and `Marker` parameters to page through the custom key stores in the output.

```
$ aws kms describe-custom-key-stores
```

The following example command uses the `CustomKeyStoreName` parameter to get only the custom key store with the `ExampleCloudHSMKeyStore` friendly name. You can use either the `CustomKeyStoreName` or `CustomKeyStoreId` parameter (but not both) in each command.

The following example output represents an AWS CloudHSM key store that is connected to its AWS CloudHSM cluster.

Note

The `CustomKeyStoreType` field was added to the `DescribeCustomKeyStores` response to distinguish AWS CloudHSM key stores from external key stores.

```
$ aws kms describe-custom-key-stores --custom-key-store-name ExampleCloudHSMKeyStore
{
   "CustomKeyStores": [
      {
         "CloudHsmClusterId": "cluster-1a23b4cdefg",
         "ConnectionState": "CONNECTED",
         "CreationDate": "1.499288695918E9",
         "CustomKeyStoreId": "cks-1234567890abcdef0",
         "CustomKeyStoreName": "ExampleCloudHSMKeyStore",
         "CustomKeyStoreType": "AWS_CLOUDHSM",
         "TrustAnchorCertificate": "<certificate appears here>"
      }
   ]
}
```

A ConnectionState of Disconnected indicates that a custom key store has never been connected or it was intentionally disconnected from its AWS CloudHSM cluster. However, if attempts to use a KMS key in a connected AWS CloudHSM key store fail, that might indicate a problem with the AWS CloudHSM key store or its AWS CloudHSM cluster. For help, see How to fix a failing KMS key.

If the ConnectionState of the custom key store is FAILED, the DescribeCustomKeyStores response includes a ConnectionErrorCode element that explains the reason for the error.

For example, in the following output, the INVALID_CREDENTIALS value indicates that the custom key store connection failed because the kmsuser password is invalid. For help with this and other connection error failures, see Troubleshooting a custom key store.

```
$ aws kms describe-custom-key-stores --custom-key-store-id cks-1234567890abcdef0
{
   "CustomKeyStores": [
      {
         "CloudHsmClusterId": "cluster-1a23b4cdefg",
         "ConnectionErrorCode": "INVALID_CREDENTIALS",
         "ConnectionState": "FAILED",
         "CustomKeyStoreId": "cks-1234567890abcdef0",
         "CustomKeyStoreName": "ExampleCloudHSMKeyStore",
         "CustomKeyStoreType": "AWS_CLOUDHSM",
         "CreationDate": "1.499288695918E9",
         "TrustAnchorCertificate": "<certificate appears here>"
      }
   ]
}
```

Editing AWS CloudHSM key store settings

You can change the settings of an existing AWS CloudHSM key store. The custom key store must be disconnected its AWS CloudHSM cluster.

To edit AWS CloudHSM key store settings:

1. Disconnect the custom key store from its AWS CloudHSM cluster. While the custom key store is disconnected, you cannot create AWS KMS keys (KMS keys) in the custom key store and you cannot use the KMS keys it contains for cryptographic operations.
2. Edit one or more of the AWS CloudHSM key store settings.
3. Reconnect the custom key store to its AWS CloudHSM cluster.

You can edit the following settings in a custom key store:

The friendly name of the custom key store.

Enter a new friendly name. The new name must be unique among all custom key stores in your AWS account.

> ⚠ **Important**
>
> Do not include confidential or sensitive information in this field. This field may be displayed in plaintext in CloudTrail logs and other output.

The cluster ID of the associated AWS CloudHSM cluster.

Edit this value to substitute a related AWS CloudHSM cluster for the original one. You can use this feature to repair a custom key store if its AWS CloudHSM cluster becomes corrupted or is deleted.

Specify an AWS CloudHSM cluster that shares a backup history with the original cluster and fulfills the requirements for association with a custom key store, including two active HSMs in different Availability Zones. Clusters that share a backup history have the same cluster certificate. To view the cluster certificate of a cluster, use the DescribeClusters operation. You cannot use the edit feature to associate the custom key store with an unrelated AWS CloudHSM cluster.

The current password of the `kmsuser` crypto user (CU).

Tells AWS KMS the current password of the `kmsuser` CU in the AWS CloudHSM cluster. This action does not change the password of the `kmsuser` CU in the AWS CloudHSM cluster.

If you change the password of the `kmsuser` CU in the AWS CloudHSM cluster, use this feature to tell AWS KMS the new `kmsuser` password. Otherwise, AWS KMS cannot log into the cluster and all attempts to connect the custom key store to the cluster fail.

Topics

- Edit an AWS CloudHSM key store (console)
- Edit an AWS CloudHSM key store (API)

Edit an AWS CloudHSM key store (console)

When you edit an AWS CloudHSM key store, you can change any or of the configurable values.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Custom key stores**, **AWS CloudHSM key stores**.
4. Choose the row of the AWS CloudHSM key store you want to edit.

   If the value in the **Connection state** column is not **Disconnected**, you must disconnect the custom key store before you can edit it. (From the **Key store actions** menu, choose **Disconnect**.)

   While an AWS CloudHSM key store is disconnected, you can manage the AWS CloudHSM key store and its KMS keys, but you cannot create or use KMS keys in the AWS CloudHSM key store.
5. From the **Key store actions** menu, choose **Edit**.
6. Do one or more of the following actions.

   - Type a new friendly name for the custom key store.
   - Type the cluster ID of a related AWS CloudHSM cluster.
   - Type the current password of the `kmsuser` crypto user in the associated AWS CloudHSM cluster.
7. Choose **Save**.

   When the procedure is successful, a message describes the settings that you edited. When it is unsuccessful, an error message appears that describes the problem and provides help on how to fix it. If you need more help, see Troubleshooting a custom key store.
8. Reconnect the custom key store.

   To use the AWS CloudHSM key store, you must reconnect it after editing. You can leave the AWS CloudHSM key store disconnected. But while it is disconnected, you cannot create KMS keys in the AWS CloudHSM key store or use the KMS keys in the AWS CloudHSM key store in cryptographic operations.

Edit an AWS CloudHSM key store (API)

To change the properties of an AWS CloudHSM key store, use the UpdateCustomKeyStore operation. You can change multiple properties of a custom key store in the same command. If the operation is successful, AWS KMS returns an HTTP 200 response and a JSON object with no properties. To verify that the changes are effective, use the DescribeCustomKeyStores operation.

The examples in this section use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

Begin by using DisconnectCustomKeyStore to disconnect the custom key store from its AWS CloudHSM cluster. Replace the example custom key store ID, cks-1234567890abcdef0, with an actual ID.

```
$ aws kms disconnect-custom-key-store --custom-key-store-id cks-1234567890abcdef0
```

The first example uses UpdateCustomKeyStore to change the friendly name of the AWS CloudHSM key store to `DevelopmentKeys`. The command uses the `CustomKeyStoreId` parameter to identify the AWS CloudHSM key store and the `CustomKeyStoreName` to specify the new name for the custom key store.

```
$ aws kms update-custom-key-store --custom-key-store-id cks-1234567890abcdef0 --new-
custom-key-store-name DevelopmentKeys
```

The following example changes the cluster that is associated with an AWS CloudHSM key store to another backup of the same cluster. The command uses the `CustomKeyStoreId` parameter to identify the AWS CloudHSM key store and the `CloudHsmClusterId` parameter to specify the new cluster ID.

```
$ aws kms update-custom-key-store --custom-key-store-id cks-1234567890abcdef0 --cloud-
hsm-cluster-id cluster-1a23b4cdefg
```

The following example tells AWS KMS that the current `kmsuser` password is `ExamplePassword`. The command uses the `CustomKeyStoreId` parameter to identify the AWS CloudHSM key store and the `KeyStorePassword` parameter to specify the current password.

```
$ aws kms update-custom-key-store --custom-key-store-id cks-1234567890abcdef0 --key-
store-password ExamplePassword
```

The final command reconnects the AWS CloudHSM key store to its AWS CloudHSM cluster. You can leave the custom key store in the disconnected state, but you must connect it before you can create new KMS keys or use existing KMS keys for cryptographic operations. Replace the example custom key store ID with an actual ID.

```
$ aws kms connect-custom-key-store --custom-key-store-id cks-1234567890abcdef0
```

Connecting and disconnecting an AWS CloudHSM key store

New AWS CloudHSM key stores are not connected. Before you can create and use AWS KMS keys in your AWS CloudHSM key store, you need to connect it to its associated AWS CloudHSM cluster. You can connect and disconnect your AWS CloudHSM key store at any time, and view its connection state.

You are not required to connect your AWS CloudHSM key store. You can leave an AWS CloudHSM key store in a disconnected state indefinitely and connect it only when you need to use it. However, you might want to test the connection periodically to verify that the settings are correct and it can be connected.

> **Note**
>
> AWS CloudHSM key stores have a `DISCONNECTED` connection state only when the key store has never been connected or you explicitly disconnect it. If your AWS CloudHSM key store connection state is `CONNECTED` but you are having trouble using it, make sure that its associated AWS CloudHSM cluster is active and contains at least one active HSMs. For help with connection failures, see the section called "Troubleshooting a custom key store".

Topics

- Connecting an AWS CloudHSM key store
- Disconnecting an AWS CloudHSM key store
- Connect an AWS CloudHSM key store (console)
- Connect a custom key store (API)

- Disconnect an AWS CloudHSM key store (console)
- Disconnect an AWS CloudHSM key store (API)

Connecting an AWS CloudHSM key store

When you connect an AWS CloudHSM key store, AWS KMS finds the associated AWS CloudHSM cluster, connects to it, logs into the AWS CloudHSM client as the `kmsuser` crypto user (CU), and then rotates the `kmsuser` password. AWS KMS remains logged into the AWS CloudHSM client as long as the AWS CloudHSM key store is connected.

To establish the connection, AWS KMS creates a security group named `kms-<custom key store ID>` in the virtual private cloud (VPC) of the cluster. The security group has a single rule that allows inbound traffic from the cluster security group. AWS KMS also creates an elastic network interface (ENI) in each Availability Zone of the private subnet for the cluster. AWS KMS adds the ENIs to the `kms-<cluster ID>` security group and the security group for the cluster. The description of each ENI is `KMS managed ENI for cluster <cluster-ID>`.

The connection process can take an extended amount of time to complete; up to 20 minutes.

Before you connect the AWS CloudHSM key store, verify that it meets the requirements.

- Its associated AWS CloudHSM cluster must contain at least one active HSM. To find the number of HSMs in the cluster, view the cluster in the AWS CloudHSM console or use the DescribeClusters operation. If necessary, you can add an HSM.
- The cluster must have a `kmsuser` crypto user (CU) account, but that CU cannot be logged into the cluster when you connect the AWS CloudHSM key store. For help with logging out, see How to log out and reconnect.
- The connection state of the AWS CloudHSM key store cannot be `DISCONNECTING` or `FAILED`. To view the connection state, use the AWS KMS console or the DescribeCustomKeyStores response. If the connection state is `FAILED`, disconnect the custom key store, fix the problem, and then connect it.

For help with connection failures, see How to fix a connection failure.

When your AWS CloudHSM key store is connected, you can create KMS keys in it and use existing KMS keys in cryptographic operations.

Disconnecting an AWS CloudHSM key store

When you disconnect an AWS CloudHSM key store, AWS KMS logs out of the AWS CloudHSM client, disconnects from the associated AWS CloudHSM cluster, and removes the network infrastructure that it created to support the connection.

While an AWS CloudHSM key store is disconnected, you can manage the AWS CloudHSM key store and its KMS keys, but you cannot create or use KMS keys in the AWS CloudHSM key store. The connection state of the key store is `DISCONNECTED` and the key state of KMS keys in the custom key store is `Unavailable`, unless they are `PendingDeletion`. You can reconnect the AWS CloudHSM key store at any time.

When you disconnect a custom key store, the KMS keys in the key store become unusable right away (subject to eventual consistency). However, resources encrypted with data keys protected by the KMS key are not affected until the KMS key is used again, such as to decrypt the data key. This issue affects AWS services, many of which use data keys to protect your resources. For details, see How unusable KMS keys affect data keys.

> **Note**
>
> While a custom key store is disconnected, all attempts to create KMS keys in the custom key store or to use existing KMS keys in cryptographic operations will fail. This action can prevent users from storing and accessing sensitive data.

To better estimate the effect of disconnecting your custom key store, identify the KMS keys in the custom key store and determine their past use.

You might disconnect an AWS CloudHSM key store for reasons such as the following:

- **To rotate of the `kmsuser` password.** AWS KMS changes the `kmsuser` password each time that it connects to the AWS CloudHSM cluster. To force a password rotation, just disconnect and reconnect.
- **To audit the key material** for the KMS keys in the AWS CloudHSM cluster. When you disconnect the custom key store, AWS KMS logs out of the `kmsuser` crypto user account in the AWS CloudHSM client. This allows you to log into the cluster as the `kmsuser` CU and audit and manage the key material for the KMS key.
- **To immediately disable all KMS keys** in the AWS CloudHSM key store. You can disable and re-enable KMS keys in an AWS CloudHSM key store by using the AWS Management Console or the

DisableKey operation. These operations complete quickly, but they act on one KMS key at a time. Disconnecting the AWS CloudHSM key store immediately changes the key state of all KMS keys in the AWS CloudHSM key store to `Unavailable`, which prevents them from being used in any cryptographic operation.

- **To repair a failed connection attempt**. If an attempt to connect an AWS CloudHSM key store fails (the connection state of the custom key store is `FAILED`), you must disconnect the AWS CloudHSM key store before you try to connect it again.

Connect an AWS CloudHSM key store (console)

To connect an AWS CloudHSM key store in the AWS Management Console, begin by selecting the AWS CloudHSM key store from the **Custom key stores** page. The connection process can take up to 20 minutes to complete.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Custom key stores**, **AWS CloudHSM key stores**.
4. Choose the row of the AWS CloudHSM key store you want to connect.

   If the connection state of the AWS CloudHSM key store is **Failed**, you must disconnect the custom key store before you connect it.
5. From the **Key store actions** menu, choose **Connect**.

AWS KMS begins the process of connecting your custom key store. It finds the associated AWS CloudHSM cluster, builds the required network infrastructure, connects to it, logs into the AWS CloudHSM cluster as the `kmsuser` CU, and rotates the `kmsuser` password. When the operation completes, the connection state changes to **Connected**.

If the operation fails, an error message appears that describes the reason for the failure. Before you try to connect again, view the connection state of your AWS CloudHSM key store. If it is **Failed**, you must disconnect the custom key store before you connect it again. If you need help, see Troubleshooting a custom key store.

Next: the section called “Creating KMS keys in an AWS CloudHSM key store”.

Connect a custom key store (API)

To connect a disconnected AWS CloudHSM key store, use the ConnectCustomKeyStore operation. The associated AWS CloudHSM cluster must contain at least one active HSM and the connection state cannot be FAILED.

The connection process takes an extended amount of time to complete; up to 20 minutes. Unless it fails quickly, the operation returns an HTTP 200 response and a JSON object with no properties. However, this initial response does not indicate that the connection was successful. To determine the connection state of the custom key store, see the DescribeCustomKeyStores response.

The examples in this section use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

To identify the AWS CloudHSM key store, use its custom key store ID. You can find the ID on the **Custom key stores** page in the console or by using the DescribeCustomKeyStores operation with no parameters. Before running this example, replace the example ID with a valid one.

```
$ aws kms connect-custom-key-store --custom-key-store-id cks-1234567890abcdef0
```

To verify that the AWS CloudHSM key store is connected, use the DescribeCustomKeyStores operation. By default, this operation returns all custom keys stores in your account and Region. But you can use either the CustomKeyStoreId or CustomKeyStoreName parameter (but not both) to limit the response to particular custom key stores. The ConnectionState value of CONNECTED indicates that the custom key store is connected to its AWS CloudHSM cluster.

> **Note**
>
> The CustomKeyStoreType field was added to the DescribeCustomKeyStores response to distinguish AWS CloudHSM key stores from external key stores.

```
$ aws kms describe-custom-key-stores --custom-key-store-id cks-1234567890abcdef0
{
   "CustomKeyStores": [
      "CustomKeyStoreId": "cks-1234567890abcdef0",
      "CustomKeyStoreName": "ExampleCloudHSMKeyStore",
      "CloudHsmClusterId": "cluster-1a23b4cdefg",
      "CustomKeyStoreType": "AWS_CLOUDHSM",
      "TrustAnchorCertificate": "<certificate string appears here>",
      "CreationDate": "1.499288695918E9",
      "ConnectionState": "CONNECTED"
   ],
}
```

If the ConnectionState value is failed, the ConnectionErrorCode element indicates the reason for the failure. In this case, AWS KMS could not find an AWS CloudHSM cluster in your account with the cluster ID cluster-1a23b4cdefg. If you deleted the cluster, you can restore it from a backup of the original cluster and then edit the cluster ID for the custom key store. For help responding to a connection error code, see How to fix a connection failure.

```
$ aws kms describe-custom-key-stores --custom-key-store-id cks-1234567890abcdef0
{
   "CustomKeyStores": [
      "CustomKeyStoreId": "cks-1234567890abcdef0",
      "CustomKeyStoreName": "ExampleKeyStore",
      "CloudHsmClusterId": "cluster-1a23b4cdefg",
      "CustomKeyStoreType": "AWS_CLOUDHSM",
      "TrustAnchorCertificate": "<certificate string appears here>",
      "CreationDate": "1.499288695918E9",
      "ConnectionState": "FAILED"
      "ConnectionErrorCode": "CLUSTER_NOT_FOUND"
   ],
}
```

Next: Creating KMS keys in an AWS CloudHSM key store.

Disconnect an AWS CloudHSM key store (console)

To disconnect a connected AWS CloudHSM key store in the AWS Management Console, begin by choosing the AWS CloudHSM key store from the **Custom Key Stores** page.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Custom key stores**, **AWS CloudHSM key stores**.
4. Choose the row of the external key store you want to disconnect.
5. From the **Key store actions** menu, choose **Disconnect**.

When the operation completes, the connection state changes from **Disconnecting** to **Disconnected**. If the operation fails, an error message appears that describes the problem and provides help on how to fix it. If you need more help, see Troubleshooting a custom key store.

Disconnect an AWS CloudHSM key store (API)

To disconnect a connected AWS CloudHSM key store, use the DisconnectCustomKeyStore operation. If the operation is successful, AWS KMS returns an HTTP 200 response and a JSON object with no properties.

The examples in this section use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

This example disconnects an AWS CloudHSM key store. Before running this example, replace the example ID with a valid one.

```
$ aws kms disconnect-custom-key-store --custom-key-store-id cks-1234567890abcdef0
```

To verify that the AWS CloudHSM key store is disconnected, use the DescribeCustomKeyStores operation. By default, this operation returns all custom keys stores in your account and Region. But you can use either the `CustomKeyStoreId` and `CustomKeyStoreName` parameter (but not both) to limit the response to particular custom key stores. The `ConnectionState` value of `DISCONNECTED` indicates that this example AWS CloudHSM key store is not connected to its AWS CloudHSM cluster.

```
$ aws kms describe-custom-key-stores --custom-key-store-id cks-1234567890abcdef0
{
   "CustomKeyStores": [
      "CloudHsmClusterId": "cluster-1a23b4cdefg",
      "ConnectionState": "DISCONNECTED",
      "CreationDate": "1.499288695918E9",
      "CustomKeyStoreId": "cks-1234567890abcdef0",
      "CustomKeyStoreName": "ExampleKeyStore",
      "CustomKeyStoreType": "AWS_CLOUDHSM",
      "TrustAnchorCertificate": "<certificate string appears here>"
   ],
}
```

Deleting an AWS CloudHSM key store

When you delete an AWS CloudHSM key store, AWS KMS deletes all metadata about the AWS CloudHSM key store from KMS, including information about its association with an AWS CloudHSM cluster. This operation does not affect the AWS CloudHSM cluster, its HSMs, or its users. You can create a new AWS CloudHSM key store that is associated with the same AWS CloudHSM cluster, but you cannot undo the delete operation.

You can only delete an AWS CloudHSM key store that is disconnected from its AWS CloudHSM cluster and does not contain any AWS KMS keys. Before you delete a custom key store, do the following.

- Verify that you will never need to use any of the KMS keys in the key store for any cryptographic operations. Then schedule deletion of all of the KMS keys from the key store. For help finding the KMS keys in an AWS CloudHSM key store, see Find the KMS keys in an AWS CloudHSM key store.
- Confirm that all KMS keys have been deleted. To view the KMS keys in an AWS CloudHSM key store, see Viewing KMS keys in an AWS CloudHSM key store.
- Disconnect the AWS CloudHSM key store from its AWS CloudHSM cluster.

Instead of deleting the AWS CloudHSM key store, consider disconnecting it from its associated AWS CloudHSM cluster. While an AWS CloudHSM key store is disconnected, you can manage the AWS CloudHSM key store and its AWS KMS keys. But you cannot create or use KMS keys in the AWS CloudHSM key store. You can reconnect the AWS CloudHSM key store at any time.

Topics

Delete an AWS CloudHSM key store (console)

To delete an AWS CloudHSM key store in the AWS Management Console, begin by selecting the AWS CloudHSM key store from the **Custom key stores** page.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.

3. In the navigation pane, choose **Custom key stores**, **AWS CloudHSM key stores**.
4. Find the row that represents the AWS CloudHSM key store that you want to delete. If the **Connection state** of the AWS CloudHSM key store is not **Disconnected**, you must disconnect the AWS CloudHSM key store before you delete it.
5. From the **Key store actions** menu, choose **Delete**.

When the operation completes, a success message appears and the AWS CloudHSM key store no longer appears in the key stores list. If the operation is unsuccessful, an error message appears that describes the problem and provides help on how to fix it. If you need more help, see Troubleshooting a custom key store.

Delete an AWS CloudHSM key store (API)

To delete an AWS CloudHSM key store, use the DeleteCustomKeyStore operation. If the operation is successful, AWS KMS returns an HTTP 200 response and a JSON object with no properties.

To begin, verify that the AWS CloudHSM key store does not contain any AWS KMS keys. You cannot delete a custom key store that contains KMS keys. The first example command uses ListKeys and DescribeKey to search for AWS KMS keys in the AWS CloudHSM key store with the example *cks-1234567890abcdef0* custom key store ID. In this case, the command does not return any KMS keys. If it does, use the ScheduleKeyDeletion operation to schedule deletion of each of the KMS keys.

Bash

```
for key in $(aws kms list-keys --query 'Keys[*].KeyId' --output text) ;
do aws kms describe-key --key-id $key |
grep '"CustomKeyStoreId": "cks-1234567890abcdef0"' --context 100; done
```

PowerShell

```
PS C:\> Get-KMSKeyList | Get-KMSKey | where CustomKeyStoreId -eq
 'cks-1234567890abcdef0'
```

Next, disconnect the AWS CloudHSM key store. This example command uses the DisconnectCustomKeyStore operation to disconnect an AWS CloudHSM key store from its AWS CloudHSM cluster. Before running this command, replace the example custom key store ID with a valid one.

Bash

```
$ aws kms disconnect-custom-key-store --custom-key-store-id cks-1234567890abcdef0
```

PowerShell

```
PS C:\> Disconnect-KMSCustomKeyStore -CustomKeyStoreId cks-1234567890abcdef0
```

After the custom key store is disconnected, you can use the DeleteCustomKeyStore operation to delete it.

Bash

```
$ aws kms delete-custom-key-store --custom-key-store-id cks-1234567890abcdef0
```

PowerShell

```
PS C:\> Remove-KMSCustomKeyStore -CustomKeyStoreId cks-1234567890abcdef0
```

Managing KMS keys in a CloudHSM key store

You can create, view, manage, use, and schedule deletion of the AWS KMS keys in an AWS CloudHSM key store. The procedures that you use are very similar to those you use for other KMS keys. The only difference is that you specify an AWS CloudHSM key store when you create the KMS key. Then, AWS KMS creates non-extractable key material for the KMS key in the AWS CloudHSM cluster that is associated with the AWS CloudHSM key store. When you use a KMS key in an AWS CloudHSM key store, the cryptographic operations are performed in the HSMs in the cluster.

Supported features

In addition to the procedures discussed in this section, you can do the following with KMS keys in an AWS CloudHSM key store:

- Use key policies, IAM policies, and grants to authorize access to the KMS keys.
- Enable and disable the KMS keys.
- Assign tags and create aliases, and use attribute-based access control (ABAC) to authorize access to the KMS keys.

- Use the KMS keys for cryptographic operations, including encrypting, decrypting, re-encrypting, and generating data keys.
- Use the KMS keys with AWS services that integrate with AWS KMS and support customer managed keys.
- Track use of your KMS keys in AWS CloudTrail logs and Amazon CloudWatch monitoring tools.

Unsupported features

- AWS CloudHSM key stores support only symmetric encryption KMS keys. You cannot create HMAC KMS keys, asymmetric KMS keys, or asymmetric data key pairs in an AWS CloudHSM key store.
- You cannot import key material into a KMS key in an AWS CloudHSM key store. AWS KMS generates the key material for the KMS key in the AWS CloudHSM cluster.
- You cannot enable or disable automatic rotation of the key material for a KMS key in an AWS CloudHSM key store.

Topics

Creating KMS keys in an AWS CloudHSM key store

After you have created an AWS CloudHSM key store, you can create AWS KMS keys in your key store. They must be symmetric encryption KMS keys with key material that AWS KMS generates. You cannot create asymmetric KMS keys, HMAC KMS keys or KMS keys with imported key material in a custom key store. Also, you cannot use symmetric encryption KMS keys in a custom key store to generate asymmetric data key pairs.

To create a KMS key in an AWS CloudHSM key store, the AWS CloudHSM key store must be connected to the associated AWS CloudHSM cluster and the cluster must contain at least two active HSMs in different Availability Zones. To find the connection state and number of HSMs, view the AWS CloudHSM key stores page in the AWS Management Console. When using the API operations, use the DescribeCustomKeyStores operation to verify that the AWS CloudHSM key store is connected. To verify the number of active HSMs in the cluster and their Availability Zones, use the AWS CloudHSM DescribeClusters operation.

When you create a KMS key in your AWS CloudHSM key store, AWS KMS creates the KMS key in AWS KMS. But, it creates the key material for the KMS key in the associated AWS CloudHSM cluster. Specifically, AWS KMS signs into the cluster as the `kmsuser` CU that you created. Then it creates a persistent, non-extractable, 256-bit Advanced Encryption Standard (AES) symmetric key in the cluster. AWS KMS sets the value of the key label attribute, which is visible only in the cluster, to Amazon Resource Name (ARN) of the KMS key.

When the command succeeds, the key state of the new KMS key is `Enabled` and its origin is `AWS_CLOUDHSM`. You cannot change the origin of any KMS key after you create it. When you view a KMS key in an AWS CloudHSM key store in the AWS KMS console or by using the DescribeKey operation, you can see typical properties, like its key ID, key state, and creation date. But you can also see the custom key store ID and (optionally) the AWS CloudHSM cluster ID. For details, see Viewing KMS keys in an AWS CloudHSM key store.

If your attempt to create a KMS key in your AWS CloudHSM key store fails, use the error message to help you determine the cause. It might indicate that the AWS CloudHSM key store is not connected (`CustomKeyStoreInvalidStateException`) or the associated AWS CloudHSM cluster doesn't have the two active HSMs that are required for this operation (`CloudHsmClusterInvalidConfigurationException`). For help see Troubleshooting a custom key store.

For an example of the AWS CloudTrail log of the operation that creates a KMS key in an AWS CloudHSM key store, see CreateKey.

Topics

- Create a KMS key in an AWS CloudHSM key store (console)
- Create a KMS key in an AWS CloudHSM key store (API)

Create a KMS key in an AWS CloudHSM key store (console)

Use the following procedure to create a symmetric encryption KMS key in an AWS CloudHSM key store.

Note

Do not include confidential or sensitive information in the alias, description, or tags. These fields may appear in plain text in CloudTrail logs and other output.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Choose **Create key**.
5. Choose **Symmetric**.
6. In **Key usage**, the **Encrypt and decrypt** option is selected for you. Do not change it.
7. Choose **Advanced options**.
8. For **Key material origin**, choose **AWS CloudHSM key store**.

   You cannot create a multi-Region key in an AWS CloudHSM key store.
9. Choose **Next**.
10. Select an AWS CloudHSM key store for your new KMS key. To create a new AWS CloudHSM key store, choose **Create custom key store**.

    The AWS CloudHSM key store that you select must have a status of **Connected**. Its associated AWS CloudHSM cluster must be active and contain at least two active HSMs in different Availability Zones.

    For help with connecting an AWS CloudHSM key store, see Connecting and disconnecting an AWS CloudHSM key store. For help with adding HSMs, see Adding an HSM in the *AWS CloudHSM User Guide*.
11. Choose **Next**.
12. Type an alias and an optional description for the KMS key.
13. (Optional). On the **Add Tags** page, add tags that identify or categorize your KMS key.

    When you add tags to your AWS resources, AWS generates a cost allocation report with usage and costs aggregated by tags. Tags can also be used to control access to a KMS key. For information about tagging KMS keys, see Tagging keys and ABAC for AWS KMS.

14. Choose **Next**.

15. In the **Key Administrators** section, select the IAM users and roles who can manage the KMS key. For more information, see Allows key administrators to administer the KMS key.

> **Note**
>
> IAM policies can give other IAM users and roles permission to use the KMS key. IAM best practices discourage the use of IAM users with long-term credentials. Whenever possible, use IAM roles, which provide temporary credentials. For details, see Security best practices in IAM in the *IAM User Guide*.

16. (Optional) To prevent these key administrators from deleting this KMS key, clear the box at the bottom of the page for **Allow key administrators to delete this key.**

17. Choose **Next**.

18. In the **This account** section, select the IAM users and roles in this AWS account that can use the KMS key in cryptographic operations. For more information, see Allows key users to use the KMS key.

> **Note**
>
> IAM policies can give other IAM users and roles permission to use the KMS key. IAM best practices discourage the use of IAM users with long-term credentials. Whenever possible, use IAM roles, which provide temporary credentials. For details, see Security best practices in IAM in the *IAM User Guide*.

19. (Optional) You can allow other AWS accounts to use this KMS key for cryptographic operations. To do so, in the **Other AWS accounts** section at the bottom of the page, choose **Add another AWS account** and enter the AWS account ID of an external account. To add multiple external accounts, repeat this step.

> **Note**
>
> Administrators of the other AWS accounts must also allow access to the KMS key by creating IAM policies for their users. For more information, see Allowing users in other accounts to use a KMS key.

20. Choose **Next**.

21. Review the key settings that you chose. You can still go back and change all settings.
22. When you're done, choose **Finish** to create the key.

When the procedure succeeds, the display shows the new KMS key in the AWS CloudHSM key store that you chose. When you choose the name or alias of the new KMS key, the **Cryptographic configuration** tab on its detail page displays the origin of the KMS key (**AWS CloudHSM**), the name, ID, and type of the custom key store, and the ID of the AWS CloudHSM cluster. If the procedure fails, an error message appears that describes the failure.

> **Tip**
>
> To make it easier to identify KMS keys in a custom key store, on the **Customer managed keys** page, add the **Custom key store ID** column to the display. Click the gear icon in the upper-right and select **Custom key store ID**. For details, see Customizing your KMS key tables.

Create a KMS key in an AWS CloudHSM key store (API)

To create a new AWS KMS key (KMS key) in your AWS CloudHSM key store, use the CreateKey operation. Use the `CustomKeyStoreId` parameter to identify your custom key store and specify an `Origin` value of `AWS_CLOUDHSM`.

You might also want to use the `Policy` parameter to specify a key policy. You can change the key policy (PutKeyPolicy) and add optional elements, such as a description and tags at any time.

The examples in this section use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

The following example begins with a call to the DescribeCustomKeyStores operation to verify that the AWS CloudHSM key store is connected to its associated AWS CloudHSM cluster. By default, this operation returns all custom keys stores in your account and Region. To describe only a particular AWS CloudHSM key store, use its `CustomKeyStoreId` or `CustomKeyStoreName` parameter (but not both).

Before running this command, replace the example custom key store ID with a valid ID.

Note

Do not include confidential or sensitive information in the `Description` or `Tags` fields. These fields may appear in plain text in CloudTrail logs and other output.

```
$ aws kms describe-custom-key-stores --custom-key-store-id cks-1234567890abcdef0
{
   "CustomKeyStores": [
      "CustomKeyStoreId": "cks-1234567890abcdef0",
      "CustomKeyStoreName": "ExampleKeyStore",
      "CustomKeyStoreType": "AWS CloudHSM key store",
      "CloudHsmClusterId": "cluster-1a23b4cdefg",
      "TrustAnchorCertificate": "<certificate string appears here>",
      "CreationDate": "1.499288695918E9",
      "ConnectionState": "CONNECTED"
   ],
}
```

The next example command uses the DescribeClusters operation to verify that the AWS CloudHSM cluster that is associated with the `ExampleKeyStore` (cluster-1a23b4cdefg) has at least two active HSMs. If the cluster has fewer than two HSMs, the `CreateKey` operation fails.

```
$ aws cloudhsmv2 describe-clusters
{
    "Clusters": [
        {
            "SubnetMapping": {
               ...
            },
            "CreateTimestamp": 1507133412.351,
            "ClusterId": "cluster-1a23b4cdefg",
            "SecurityGroup": "sg-865af2fb",
            "HsmType": "hsm1.medium",
            "VpcId": "vpc-1a2b3c4d",
            "BackupPolicy": "DEFAULT",
            "Certificates": {
                "ClusterCertificate": "-----BEGIN CERTIFICATE-----\...\n-----END
 CERTIFICATE-----\n"
            },
            "Hsms": [
                {
                    "AvailabilityZone": "us-west-2a",
                    "EniIp": "10.0.1.11",
                    "ClusterId": "cluster-1a23b4cdefg",
                    "EniId": "eni-ea8647e1",
                    "StateMessage": "HSM created.",
                    "SubnetId": "subnet-a6b10bd1",
                    "HsmId": "hsm-abcdefghijk",
                    "State": "ACTIVE"
                },
                {
                    "AvailabilityZone": "us-west-2b",
                    "EniIp": "10.0.0.2",
                    "ClusterId": "cluster-1a23b4cdefg",
                    "EniId": "eni-ea8647e1",
                    "StateMessage": "HSM created.",
                    "SubnetId": "subnet-b6b10bd2",
                    "HsmId": "hsm-zyxwvutsrqp",
                    "State": "ACTIVE"
                },
            ],
            "State": "ACTIVE"
        }
    ]
}
```

This example command uses the CreateKey operation to create a KMS key in an AWS CloudHSM key store. To create a KMS key in an AWS CloudHSM key store, you must provide the custom key store ID of the AWS CloudHSM key store and specify an `Origin` value of `AWS_CLOUDHSM`.

The response includes the IDs of the custom key store and the AWS CloudHSM cluster.

Before running this command, replace the example custom key store ID with a valid ID.

```
$ aws kms create-key --origin AWS_CLOUDHSM --custom-key-store-id cks-1234567890abcdef0
{
  "KeyMetadata": {
    "AWSAccountId": "111122223333",
    "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "CreationDate": 1.499288695918E9,
    "Description": "Example key",
    "Enabled": true,
    "MultiRegion": false,
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "KeyManager": "CUSTOMER",
    "KeyState": "Enabled",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "Origin": "AWS_CLOUDHSM"
    "CloudHsmClusterId": "cluster-1a23b4cdefg",
    "CustomKeyStoreId": "cks-1234567890abcdef0"
    "KeySpec": "SYMMETRIC_DEFAULT",
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "EncryptionAlgorithms": [
        "SYMMETRIC_DEFAULT"
    ]
  }
}
```

Viewing KMS keys in an AWS CloudHSM key store

To view the AWS KMS keys in an AWS CloudHSM key store, use the same techniques that you would use to view any AWS KMS customer managed keys. To learn the basics, see Viewing keys. To identify the keys in your AWS CloudHSM cluster that serve as key material for your KMS key, see Finding KMS keys and key material. For information about viewing the AWS CloudTrail logs that record all API operations on a custom key store, see Logging AWS KMS API calls with AWS CloudTrail.

In the AWS KMS console, the KMS keys in your custom key store are displayed on the Customer managed keys page, along with all other customer managed keys in your AWS account and Region.

However, the following values are specific to KMS keys in an AWS CloudHSM key store.

- The name and ID of the AWS CloudHSM key store that stores the KMS key.
- The cluster ID of the associated AWS CloudHSM cluster that contains their key material.
- An `Origin` value of `AWS CloudHSM` in the AWS KMS console or `AWS_CLOUDHSM` in API responses.
- The key state value can be `Unavailable`. For help resolving the status, see How to fix unavailable KMS keys.

To view the KMS keys in an AWS CloudHSM key store (Console)

1. Open the AWS KMS console at https://console.aws.amazon.com/kms.

2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. In the upper-right corner, choose the gear icon, choose **Custom key store ID** and **Origin**, then choose **Confirm**.
5. To identify KMS keys in any AWS CloudHSM key store, look for KMS keys with an **Origin** value of **AWS CloudHSM**. To identify KMS keys in a particular AWS CloudHSM key store, view the values in the **Custom key store ID** column.
6. Choose the alias or key ID of a KMS key in an AWS CloudHSM key store.

   This page displays detailed information about the KMS key, including its Amazon Resource Name (ARN), key policy, and tags.
7. Choose the **Cryptographic configuration** tab. The tabs are below the **General configuration** section.

   This section includes information about the AWS CloudHSM key store and AWS CloudHSM cluster associated with the KMS key.s

To view the KMS keys in a custom key store (API)

You use the same AWS KMS API operations to view the KMS keys in an AWS CloudHSM key store that you would use for any KMS key, including ListKeys, DescribeKey, and GetKeyPolicy. For example, the following `describe-key` operation in the AWS CLI shows the special fields for a KMS key in an AWS CloudHSM key store. Before running a command like this one, replace the example KMS key ID with a valid value.

```
$ aws kms describe-key --key-id 1234abcd-12ab-34cd-56ef-1234567890ab

{
   "KeyMetadata": {
      "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "AWSAccountId": "111122223333",
      "CloudHsmClusterId": "cluster-1a23b4cdefg",
      "CreationDate": 1537582718.431,
      "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
      "CustomKeyStoreId": "cks-1234567890abcdef0",
      "Description": "Key in custom key store",
      "Enabled": true,
      "EncryptionAlgorithms": [
         "SYMMETRIC_DEFAULT"
      ],
      "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
      "KeyManager": "CUSTOMER",
      "KeySpec": "SYMMETRIC_DEFAULT",
      "KeyState": "Enabled",
      "KeyUsage": "ENCRYPT_DECRYPT",
      "MultiRegion": false,
      "Origin": "AWS_CLOUDHSM"
   }
}
```

For help finding the KMS keys in an AWS CloudHSM key store or identifying the keys in your AWS CloudHSM cluster that serve as key material for your KMS key, see Finding KMS keys and key material.

Using KMS keys in an AWS CloudHSM key store

After you create a symmetric encryption KMS key in an AWS CloudHSM key store, you can use it for the following cryptographic operations:

- Encrypt
- Decrypt
- GenerateDataKey
- GenerateDataKeyWithoutPlaintext
- ReEncrypt

The operations that generate asymmetric data key pairs, GenerateDataKeyPair and GenerateDataKeyPairWithoutPlaintext, are *not* supported in custom key stores.

When you use your KMS key in a request, identify the KMS key by its ID or alias; you do not need to specify the AWS CloudHSM key store or AWS CloudHSM cluster. The response includes the same fields that are returned for any symmetric encryption KMS key.

However, when you use a KMS key in an AWS CloudHSM key store, the cryptographic operation is performed entirely within the AWS CloudHSM cluster that is associated with the AWS CloudHSM key store. The operation uses the key material in the cluster that is associated with the KMS key that you chose.

To make this possible, the following conditions are required.

- The key state of the KMS key must be Enabled. To find the key state, use the **Status** field in the AWS KMS console or the KeyState field in the DescribeKey response.
- The AWS CloudHSM key store must be connected to its AWS CloudHSM cluster. Its **Status** in the AWS KMS console or ConnectionState in the DescribeCustomKeyStores response must be CONNECTED.
- The AWS CloudHSM cluster that is associated with the custom key store must contain at least one active HSM. To find the number of active HSMs in the cluster, use the AWS KMS console, the AWS CloudHSM console, or the DescribeClusters operation.
- The AWS CloudHSM cluster must contain the key material for the KMS key. If the key material was deleted from the cluster, or an HSM was created from a backup that did not include the key material, the cryptographic operation will fail.

If these conditions are not met, the cryptographic operation fails, and AWS KMS returns a KMSInvalidStateException exception. Typically, you just need to reconnect the AWS CloudHSM key store. For additional help, see How to fix a failing KMS key.

When using the KMS keys in an AWS CloudHSM key store, be aware that the KMS keys in each AWS CloudHSM key store share a custom key store request quota for cryptographic operations. If you exceed the quota, AWS KMS returns a ThrottlingException. If the AWS CloudHSM cluster that is associated with the AWS CloudHSM key store is processing numerous commands, including those unrelated to the AWS CloudHSM key store, you might get a ThrottlingException at an even lower rate. If you get a ThrottlingException for any request, lower your request rate and try the commands again. For details about the custom key store request quota, see Custom key store request quotas.

Finding KMS keys and key material

If you manage an AWS CloudHSM key store, you might need to identify the KMS keys in each AWS CloudHSM key store. For example, you might need to do some of the following tasks.

- Track the KMS keys in AWS CloudHSM key store in AWS CloudTrail logs.
- Predict the effect on KMS keys of disconnecting an AWS CloudHSM key store.
- Schedule deletion of KMS keys before you delete an AWS CloudHSM key store.

In addition, you might want to identify the keys in your AWS CloudHSM cluster that serve as key material for your KMS keys. Although AWS KMS manages the KMS keys and the key material, you still retain control of and responsibility for the management of your AWS CloudHSM cluster, as well as the HSMs and backups and the keys in the HSMs. You might need to identify the keys in order to audit the key material, protect it from accidental deletion, or delete it from HSMs and cluster backups after deleting the KMS key.

All key material for the KMS keys in your AWS CloudHSM key store is owned by the `kmsuser` crypto user (CU). AWS KMS sets the key label attribute, which is viewable only in AWS CloudHSM, to the Amazon Resource Name (ARN) of the KMS key.

To find KMS keys and key material, use any of the following techniques.

- Find the KMS keys in an AWS CloudHSM key store — How to identify the KMS keys in one or all of your AWS CloudHSM key stores.
- Find all keys for an AWS CloudHSM key store — How to find all keys in your cluster that serve as key material for the KMS keys in your AWS CloudHSM key store.
- Find the AWS CloudHSM key for a KMS key — How to find the key in your cluster that serves as key material for a particular KMS key in your AWS CloudHSM key store.
- Find the KMS key for an AWS CloudHSM key — How to find the KMS key for a particular key in your cluster.

Find the KMS keys in an AWS CloudHSM key store

If you manage an AWS CloudHSM key store, you might need to identify the KMS keys in each AWS CloudHSM key store. You can use this information track the KMS key operations in AWS CloudTrail logs, predict the effect on KMS keys of disconnecting a custom key store, or schedule deletion of KMS keys before you delete an AWS CloudHSM key store.

To find the KMS keys in an AWS CloudHSM key store (console)

To find the KMS keys in a particular AWS CloudHSM key store, on the **Customer managed keys** page, view the values in the **Custom Key Store Name** or **Custom Key Store ID** fields. To identify KMS keys in any AWS CloudHSM key store, look for KMS keys with an **Origin** value of **AWS CloudHSM**. To add optional columns to the display, choose the gear icon in the upper right corner of the page.

To find the KMS keys in an AWS CloudHSM key store (API)

To find the KMS keys in an AWS CloudHSM key store, use the ListKeys and DescribeKey operations and then filter by `CustomKeyStoreId` value. Before running the examples, replace the fictitious custom key store ID values with a valid value.

Bash

To find KMS keys in a particular AWS CloudHSM key store, get all of your KMS keys in the account and Region. Then filter by the custom key store ID.

```
for key in $(aws kms list-keys --query 'Keys[*].KeyId' --output text) ;
do aws kms describe-key --key-id $key |
grep '"CustomKeyStoreId": "cks-1234567890abcdef0"' --context 100; done
```

To get KMS keys in any AWS CloudHSM key store in the account and Region, search for `CustomKeyStoreType` with a value of `AWS_CloudHSM`.

```
for key in $(aws kms list-keys --query 'Keys[*].KeyId' --output text) ;
do aws kms describe-key --key-id $key |
grep '"CustomKeyStoreType": "AWS_CloudHSM"' --context 100; done
```

PowerShell

To find KMS keys in a particular AWS CloudHSM key store, use the Get-KmsKeyList and Get-KmsKey cmdlets to get all of your KMS keys in the account and Region. Then filter by the custom key store ID.

```
PS C:\> Get-KMSKeyList | Get-KMSKey | where CustomKeyStoreId -eq
 'cks-1234567890abcdef0'
```

To get KMS keys in any AWS CloudHSM key store in the account and Region, filter for the CustomKeyStoreType value of `AWS_CLOUDHSM`.

```
PS C:\> Get-KMSKeyList | Get-KMSKey | where CustomKeyStoreType -eq 'AWS_CLOUDHSM'
```

Find all keys for an AWS CloudHSM key store

You can identify the keys in your AWS CloudHSM cluster that serve as key material for your AWS CloudHSM key store. To do that, use the findAllKeys command in cloudhsm_mgmt_util to find the key handles of all keys that kmsuser owns or shares. Unless you have logged in as kmsuser and created keys outside of AWS KMS, all of the keys that kmsuser owns represent key material for KMS keys.

Any crypto officer in the cluster can run this command without disconnecting the AWS CloudHSM key store.

1. Start cloudhsm_mgmt_util by using the procedure described in the Getting started with CloudHSM Management Utility (CMU) topic.
2. Log into cloudhsm_mgmt_util using a crypto officer (CO) account.
3. Use the listUsers command to find the user ID of the kmsuser crypto user.

   In this example, kmsuser has user ID 3.

```
aws-cloudhsm> listUsers
Users on server 0(10.0.0.1):
Number of users found:3

    User Id             User Type       User Name             MofnPubKey
 LoginFailureCnt         2FA
         1              PCO             admin                      NO
 0               NO
         2              AU              app_user                   NO
 0               NO
         3              CU              kmsuser                    NO
 0               NO
```

4. Use the findAllKeys command to find the key handles of all keys that kmsuser owns or shares. Replace the example user ID (3) with the actual user ID of kmsuser in your cluster.

   The example output shows that kmsuser owns keys with key handles 8, 9, and 262162 on both HSMs in the cluster.

```
aws-cloudhsm> findAllKeys 3 0
Keys on server 0(10.0.0.1):
Number of keys found 3
number of keys matched from start index 0::6
8,9,262162
findAllKeys success on server 0(10.0.0.1)

Keys on server 1(10.0.0.2):
```

```
Number of keys found 6
number of keys matched from start index 0::6
8,9,262162
findAllKeys success on server 1(10.0.0.2)
```

Find the KMS key for an AWS CloudHSM key

If you know the key handle of a key that `kmsuser` owns in the cluster, you can use the key label to identify the associated KMS key in your AWS CloudHSM key store.

When AWS KMS creates the key material for a KMS key in your AWS CloudHSM cluster, it writes the Amazon Resource Name (ARN) of the KMS key in the key label. Unless you have changed the label value, you can use the getAttribute command in key_mgmt_util or cloudhsm_mgmt_util to associate the key with its KMS key.

To run this procedure, you need to disconnect the AWS CloudHSM key store temporarily so you can log in as the `kmsuser` CU.

> **Note**
>
> While a custom key store is disconnected, all attempts to create KMS keys in the custom key store or to use existing KMS keys in cryptographic operations will fail. This action can prevent users from storing and accessing sensitive data.

1. Disconnect the AWS CloudHSM key store, if not already disconnected., then log into the key_mgmt_util as `kmsuser`, as explained in How to disconnect and log in.
2. Use the `getAttribute` command in key_mgmt_util or cloudhsm_mgmt_util to get the label attribute (`OBJ_ATTR_LABEL`, attribute 3) for a particular key handle.

   For example, this command uses `getAttribute` in cloudhsm_mgmt_util to get the label attribute (attribute 3) of the key with key handle `262162`. The output shows that key `262162` serves as key material for the KMS key with ARN `arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab`. Before running this command, replace the example key handle with a valid one.

   For a list of key attributes, use the listAttributes command or see the Key Attribute Reference in the *AWS CloudHSM User Guide*.

```
aws-cloudhsm> getAttribute 262162 3

Attribute Value on server 0(10.0.1.10):
OBJ_ATTR_LABEL
arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

3. Log out of key_mgmt_util or cloudhsm_mgmt_util and reconnect the AWS CloudHSM key store as explained in How to log out and reconnect.

Find the AWS CloudHSM key for a KMS key

You can use the KMS key ID of a KMS key in an AWS CloudHSM key store to identify the key in your AWS CloudHSM cluster that serves as its key material. Then you can use its key handle to identify the key in AWS CloudHSM client commands.

When AWS KMS creates the key material for a KMS key in your AWS CloudHSM cluster, it writes the Amazon Resource Name (ARN) of the KMS key in the key label. Unless you have changed the label value, you can use the findKey command in key_mgmt_util to get the key handle of the key material for the KMS key. To run this procedure, you need to disconnect the AWS CloudHSM key store temporarily so you can log in as the `kmsuser` CU.

> **Note**
>
> While a custom key store is disconnected, all attempts to create KMS keys in the custom key store or to use existing KMS keys in cryptographic operations will fail. This action can prevent users from storing and accessing sensitive data.

1. Disconnect the AWS CloudHSM key store, if it is not already disconnected, then log into the key_mgmt_util as `kmsuser`, as explained in How to disconnect and log in.

2. Use the findKey command in key_mgmt_util to search for a key with a label that matches the ARN of a KMS key in your AWS CloudHSM key store. Replace the example KMS key ARN in the value of the `-l` (lower-case L for 'label') parameter with a valid KMS key ARN.

   For example, this command finds the key with a label that matches the example KMS key ARN, `arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab`. The example output shows that the key with key handle 262162 has the specified KMS key ARN in its label. You can now use this key handle in other key_mgmt_util commands.

```
Command: findKey -l arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
Total number of keys present 1

 number of keys matched from start index 0::1
262162

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS

        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

3. Log out of key_mgmt_util and reconnect the custom key store as explained in How to log out and reconnect.

Scheduling deletion of KMS keys from an AWS CloudHSM key store

When you are certain that you will not need to use an AWS KMS key for any cryptographic operation, you can schedule the deletion of the KMS key. Use the same procedure that you would use to schedule the deletion of any KMS key from AWS KMS. In addition, keep your AWS CloudHSM key store connected so AWS KMS can delete the corresponding key material from the associated AWS CloudHSM cluster when the waiting period expires.

You can monitor the scheduling, cancellation, and deletion of the KMS key in your AWS CloudTrail logs.

Warning

Deleting a KMS key is a destructive and potentially dangerous operation that prevents you from recovering all data encrypted under the KMS key. Before scheduling deletion of the KMS key, examine past usage of the KMS key and create a Amazon CloudWatch alarm that alerts you when someone tries to use the KMS key while it is pending deletion. Whenever possible, disable the KMS key, instead of deleting it.

When you schedule deletion of a KMS key from an AWS CloudHSM key store, its key state changes to **Pending deletion**. The KMS key remains in the **Pending deletion** state throughout the waiting period, even if the KMS key becomes unavailable because you have disconnected the custom key store. This allows you to cancel the deletion of the KMS key at any time during the waiting period.

When the waiting period expires, AWS KMS deletes the KMS key from AWS KMS. Then AWS KMS makes a best effort to delete the key material from the associated AWS CloudHSM cluster. If AWS KMS cannot delete the key material, such as when the key store is disconnected from AWS KMS, you might need to manually delete the orphaned key material from the cluster.

AWS KMS does not delete the key material from cluster backups. Even if you delete the KMS key from AWS KMS and delete its key material from your AWS CloudHSM cluster, clusters created from backups might contain the deleted key material. To permanently delete the key material view the creation date of the KMS key. Then delete all cluster backups that might contain the key material.

When you schedule the deletion of a KMS key from an AWS CloudHSM key store, the KMS key becomes unusable right away (subject to eventual consistency). However, resources encrypted with data keys protected by the KMS key are not affected until the KMS key is used again, such to decrypt the data key. This issue affects AWS services, many of which use data keys to protect your resources. For details, see How unusable KMS keys affect data keys.

Troubleshooting a custom key store

AWS CloudHSM key stores are designed to be available and resilient. However, there are some error conditions that you might have to repair to keep your AWS CloudHSM key store operational.

Topics

How to fix unavailable KMS keys

The key state of AWS KMS keys in an AWS CloudHSM key store is typically `Enabled`. Like all KMS keys, the key state changes when you disable the KMS keys in an AWS CloudHSM key store or schedule them for deletion. However, unlike other KMS keys, the KMS keys in a custom key store can also have a key state of `Unavailable`.

A key state of `Unavailable` indicates that the KMS key is in a custom key store that was intentionally disconnected and attempts to reconnect it, if any, failed. While a KMS key is unavailable, you can view and manage the KMS key, but you cannot use it for cryptographic operations.

To find the key state of a KMS key, on the **Customer managed keys** page, view the **Status** field of the KMS key. Or, use the DescribeKey operation and view the `KeyState` element in the response. For details, see Viewing keys.

The KMS keys in a disconnected custom key store will have a key state of `Unavailable` or `PendingDeletion`. KMS keys that are scheduled for deletion from a custom key store have a `Pending Deletion` key state, even when the custom key store is disconnected. This allows you to cancel the scheduled key deletion without reconnecting the custom key store.

To fix an unavailable KMS key, reconnect the custom key store. After the custom key store is reconnected, the key state of the KMS keys in the custom key store is automatically restored to its previous state, such as `Enabled` or `Disabled`. KMS keys that are pending deletion remain in the `PendingDeletion` state. However, while the problem persists, enabling and disabling an unavailable KMS key does not change its key state. The enable or disable action takes effect only when the key becomes available.

For help with failed connections, see How to fix a connection failure.

How to fix a failing KMS key

Problems with creating and using KMS keys in AWS CloudHSM key stores can be caused by a problem with your AWS CloudHSM key store, its associated AWS CloudHSM cluster, the KMS key, or its key material.

When an AWS CloudHSM key store is disconnected from its AWS CloudHSM cluster, the key state of KMS keys in the custom key store is `Unavailable`. All requests to create KMS keys in a disconnected AWS CloudHSM key store return a `CustomKeyStoreInvalidStateException`

exception. All requests to encrypt, decrypt, re-encrypt, or generate data keys return a `KMSInvalidStateException` exception. To fix the problem, reconnect the AWS CloudHSM key store.

However, your attempts to use a KMS key in an AWS CloudHSM key store for cryptographic operations might fail even when its key state is `Enabled` and the connection state of the AWS CloudHSM key store is `Connected`. This might be caused by any of the following conditions.

- The key material for the KMS key might have been deleted from the associated AWS CloudHSM cluster. To investigate, find the key handle of the key material for a KMS key and, if necessary, try to recover the key material.
- All HSMs were deleted from the AWS CloudHSM cluster that is associated with the AWS CloudHSM key store. To use a KMS key in an AWS CloudHSM key store in a cryptographic operation, its AWS CloudHSM cluster must contain at least one active HSM. To verify the number and state of HSMs in an AWS CloudHSM cluster, use the AWS CloudHSM console or the DescribeClusters operation. To add an HSM to the cluster, use the AWS CloudHSM console or the CreateHsm operation.
- The AWS CloudHSM cluster associated with the AWS CloudHSM key store was deleted. To fix the problem, create a cluster from a backup that is related to the original cluster, such as a backup of the original cluster, or a backup that was used to create the original cluster. Then, edit the cluster ID in the custom key store settings. For instructions, see How to recover deleted key material for a KMS key.
- The AWS CloudHSM cluster associated with the custom key store did not have any available PKCS #11 sessions. This typically occurs during periods of high burst traffic when additional sessions are needed to service the traffic. To respond to a `KMSInternalException` with an error message about PKCS #11 sessions, back off and retry the request again.

How to fix a connection failure

If you try to connect an AWS CloudHSM key store to its AWS CloudHSM cluster, but the operation fails, the connection state of the AWS CloudHSM key store changes to `FAILED`. To find the connection state of an AWS CloudHSM key store, use the AWS KMS console or the DescribeCustomKeyStores operation.

Alternatively, some connection attempts fail quickly due to easily detected cluster configuration errors. In this case, the connection state is still `DISCONNECTED`. These failures return an error message or exception that explains why the attempt failed. Review the exception description and cluster requirements, fix the problem, update the AWS CloudHSM key store, if necessary, and try to connect again.

When the connection state is FAILED, run the DescribeCustomKeyStores operation and see the ConnectionErrorCode element in the response.

> **Note**
>
> When the connection state of an AWS CloudHSM key store is FAILED, you must disconnect the AWS CloudHSM key store before attempting to reconnect it. You cannot connect an AWS CloudHSM key store with a FAILED connection state.

- CLUSTER_NOT_FOUND indicates that AWS KMS cannot find an AWS CloudHSM cluster with the specified cluster ID. This might occur because the wrong cluster ID was provided to an API operation or the cluster was deleted and not replaced. To fix this error, verify the cluster ID, such as by using the AWS CloudHSM console or the DescribeClusters operation. If the cluster was deleted, create a cluster from a recent backup of the original. Then, disconnect the AWS CloudHSM key store, edit the AWS CloudHSM key store cluster ID setting, and reconnect the AWS CloudHSM key store to the cluster.
- INSUFFICIENT_CLOUDHSM_HSMS indicates that the associated AWS CloudHSM cluster does not contain any HSMs. To connect, the cluster must have at least one HSM. To find the number of HSMs in the cluster, use the DescribeClusters operation. To resolve this error, add at least one HSM to the cluster. If you add multiple HSMs, it's best to create them in different Availability Zones.
- INSUFFICIENT_FREE_ADDRESSES_IN_SUBNET indicates that AWS KMS could not connect the AWS CloudHSM key store to its AWS CloudHSM cluster because at least one private subnet associated with the cluster doesn't have any available IP addresses. An AWS CloudHSM key store connection requires one free IP address in each of the associated private subnets, although two are preferable.

  You can't add IP addresses (CIDR blocks) to an existing subnet. If possible, move or delete other resources that are using the IP addresses in the subnet, such as unused EC2 instances or elastic network interfaces. Otherwise, you can create a cluster from a recent backup of the AWS CloudHSM cluster with new or existing private subnets that have more free address space. Then, to associate the new cluster with your AWS CloudHSM key store, disconnect the custom key store, change the cluster ID of the AWS CloudHSM key store to the ID of the new cluster, and try to connect again.

> **Tip**
>
> To avoid resetting the `kmsuser` password, use the most recent backup of the AWS CloudHSM cluster.

- `INTERNAL_ERROR` indicates that AWS KMS could not complete the request due to an internal error. Retry the request. For `ConnectCustomKeyStore` requests, disconnect the AWS CloudHSM key store before trying to connect again.
- `INVALID_CREDENTIALS` indicates that AWS KMS cannot log into the associated AWS CloudHSM cluster because it doesn't have the correct `kmsuser` account password. For help with this error, see How to fix invalid `kmsuser` credentials.
- `NETWORK_ERRORS` usually indicates transient network issues. Disconnect the AWS CloudHSM key store, wait a few minutes, and try to connect again.
- `SUBNET_NOT_FOUND` indicates that at least one subnet in the AWS CloudHSM cluster configuration was deleted. If AWS KMS cannot find all of the subnets in the cluster configuration, attempts to connect the AWS CloudHSM key store to the AWS CloudHSM cluster fail.

  To fix this error, create a cluster from a recent backup of the same AWS CloudHSM cluster. (This process creates a new cluster configuration with a VPC and private subnets.) Verify that the new cluster meets the requirements for a custom key store, and note the new cluster ID. Then, to associate the new cluster with your AWS CloudHSM key store, disconnect the custom key store, change the cluster ID of the AWS CloudHSM key store to the ID of the new cluster, and try to connect again.

  > **Tip**
  >
  > To avoid resetting the `kmsuser` password, use the most recent backup of the AWS CloudHSM cluster.

- `USER_LOCKED_OUT` indicates that the `kmsuser` crypto user (CU) account is locked out of the associated AWS CloudHSM cluster due to too many failed password attempts. For help with this error, see How to fix invalid `kmsuser` credentials.

To fix this error, disconnect the AWS CloudHSM key store and use the changePswd command in cloudhsm_mgmt_util to change the `kmsuser` account password. Then, edit the `kmsuser` password setting for the custom key store, and try to connect again. For help, use the procedure described in the How to fix invalid `kmsuser` credentials topic.

- `USER_LOGGED_IN` indicates that the `kmsuser` CU account is logged into the associated AWS CloudHSM cluster. This prevents AWS KMS from rotating the `kmsuser` account password and logging into the cluster. To fix this error, log the `kmsuser` CU out of the cluster. If you changed the `kmsuser` password to log into the cluster, you must also and update the key store password value for the AWS CloudHSM key store. For help, see How to log out and reconnect.
- `USER_NOT_FOUND` indicates that AWS KMS cannot find a `kmsuser` CU account in the associated AWS CloudHSM cluster. To fix this error, create a `kmsuser` CU account in the cluster, and then update the key store password value for the AWS CloudHSM key store. For help, see How to fix invalid `kmsuser` credentials.

How to respond to a cryptographic operation failure

A cryptographic operation that uses a KMS key in a custom key store might fail with a `KMSInvalidStateException`. The following error messages might accompany the `KMSInvalidStateException`.

> KMS cannot communicate with your CloudHSM cluster. This might be a transient network issue. If you see this error repeatedly, verify that the Network ACLs and the security group rules for the VPC of your AWS CloudHSM cluster are correct.

- Although this is an HTTPS 400 error, it might result from transient network issues. To respond, begin by retrying the request. However, if it continues to fail, examine the configuration of your networking components. This error is most likely caused by the misconfiguration of a networking component, such as a firewall rule or VPC security group rule that is blocking outgoing traffic.

> KMS cannot communicate with your AWS CloudHSM cluster because the kmsuser is locked out. If you see this error repeatedly, disconnect the AWS CloudHSM key store and reset the kmsuser account password. Update the kmsuser password for the custom key store and try the request again.

- This error message indicates that the kmsuser crypto user (CU) account is locked out of the associated AWS CloudHSM cluster due to too many failed password attempts. For help with this error, see How to disconnect and log in.

How to fix invalid kmsuser credentials

When you connect an AWS CloudHSM key store, AWS KMS logs into the associated AWS CloudHSM cluster as the kmsuser crypto user (CU). It remains logged in until the AWS CloudHSM key store is disconnected. The DescribeCustomKeyStores response shows a ConnectionState of FAILED and ConnectionErrorCode value of INVALID_CREDENTIALS, as shown in the following example.

If you disconnect the AWS CloudHSM key store and change the kmsuser password, AWS KMS cannot log into the AWS CloudHSM cluster with the credentials of the kmsuser CU account. As a result, all attempts to connect the AWS CloudHSM key store fail. The DescribeCustomKeyStores response shows a ConnectionState of FAILED and ConnectionErrorCode value of INVALID_CREDENTIALS, as shown in the following example.

```
$ aws kms describe-custom-key-stores --custom-key-store-name ExampleKeyStore
{
   "CustomKeyStores": [
      "CloudHsmClusterId": "cluster-1a23b4cdefg",
      "ConnectionErrorCode": "INVALID_CREDENTIALS"
      "CustomKeyStoreId": "cks-1234567890abcdef0",
      "CustomKeyStoreName": "ExampleKeyStore",
      "TrustAnchorCertificate": "<certificate string appears here>",
      "CreationDate": "1.499288695918E9",
      "ConnectionState": "FAILED"
   ],
}
```

Also, after five failed attempts to log into the cluster with an incorrect password, AWS CloudHSM locks the user account. To log into the cluster, you must change the account password.

If AWS KMS gets a lockout response when it tries to log into the cluster as the kmsuser CU, the request to connect the AWS CloudHSM key store fails. The DescribeCustomKeyStores response includes a ConnectionState of FAILED and ConnectionErrorCode value of USER_LOCKED_OUT, as shown in the following example.

```
$ aws kms describe-custom-key-stores --custom-key-store-name ExampleKeyStore
```

```
{
   "CustomKeyStores": [
      "CloudHsmClusterId": "cluster-1a23b4cdefg",
      "ConnectionErrorCode": "USER_LOCKED_OUT"
      "CustomKeyStoreId": "cks-1234567890abcdef0",
      "CustomKeyStoreName": "ExampleKeyStore",
      "TrustAnchorCertificate": "<certificate string appears here>",
      "CreationDate": "1.499288695918E9",
      "ConnectionState": "FAILED"
   ],
}
```

To repair any of these conditions, use the following procedure.

1. Disconnect the AWS CloudHSM key store.
2. Run the DescribeCustomKeyStores operation and view the value of the ConnectionErrorCode element in the response.
   - If the ConnectionErrorCode value is INVALID_CREDENTIALS, determine the current password for the kmsuser account. If necessary, use the changePswd command in cloudhsm_mgmt_util to set the password to a known value.
   - If the ConnectionErrorCode value is USER_LOCKED_OUT, you must use the changePswd command in cloudhsm_mgmt_util to change the kmsuser password.
3. Edit the kmsuser password setting so it matches the current kmsuser password in the cluster. This action tells AWS KMS which password to use to log into the cluster. It does not change the kmsuser password in the cluster.
4. Connect the custom key store.

How to delete orphaned key material

After scheduling deletion of a KMS key from an AWS CloudHSM key store, you might need to manually delete the corresponding key material from the associated AWS CloudHSM cluster.

When you create a KMS key in an AWS CloudHSM key store, AWS KMS creates the KMS key metadata in AWS KMS and generates the key material in the associated AWS CloudHSM cluster. When you schedule deletion of a KMS key in an AWS CloudHSM key store, after the waiting period, AWS KMS deletes the KMS key metadata. Then AWS KMS makes a best effort to delete the corresponding key material from the AWS CloudHSM cluster. The attempt might fail if AWS KMS cannot access the cluster, such as when it's disconnected from the AWS CloudHSM key store or the kmsuser password changes. AWS KMS does not attempt to delete key material from cluster backups.

AWS KMS reports the results of its attempt to delete the key material from the cluster in the DeleteKey event entry of your AWS CloudTrail logs. It appears in the backingKeysDeletionStatus element of the additionalEventData element, as shown in the following example entry. The entry also includes the KMS key ARN, the AWS CloudHSM cluster ID, and the key handle of the key material (backing-key-id).

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "accountId": "111122223333",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2021-12-10T14:23:51Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DeleteKey",
    "awsRegion": "eu-west-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": null,
    "responseElements": null,
    "additionalEventData": {
        "customKeyStoreId": "cks-1234567890abcdef0",
        "clusterId": "cluster-1a23b4cdefg",
        "backingKeys": "[{\"keyHandle\":\"01\",\"backingKeyId\":\"backing-key-id\"}]",
        "backingKeysDeletionStatus": "[{\"keyHandle\":\"16\",\"backingKeyId\":
\"backing-key-id\",\"deletionStatus\":\"FAILURE\"}]"
    },
    "eventID": "c21f1f47-f52b-4ffe-bff0-6d994403cf40",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:eu-
west-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsServiceEvent",
    "recipientAccountId": "111122223333",
    "managementEvent": true,
    "eventCategory": "Management"
}
```

To delete the key material from the associated AWS CloudHSM cluster, use a procedure like the following one. This example uses the AWS CLI and AWS CloudHSM command line tools, but you can use the AWS Management Console instead of the CLI.

1. Disconnect the AWS CloudHSM key store, if it is not already disconnected, then log into the key_mgmt_util, as explained in How to disconnect and log in.
2. Use the deleteKey command in key_mgmt_util to delete the key from the HSMs in the cluster.

   For example, this command deletes key 262162 from the HSMs in the cluster. The key handle is listed in the CloudTrail log entry.

   ```
   Command: deleteKey -k 262162

           Cfm3DeleteKey returned: 0x00 : HSM Return: SUCCESS

           Cluster Error Status
           Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
           Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
           Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
   ```
3. Log out of key_mgmt_util and reconnect the AWS CloudHSM key store as described in How to log out and reconnect.

How to recover deleted key material for a KMS key

If the key material for an AWS KMS key is deleted, the KMS key is unusable and all ciphertext that was encrypted under the KMS key cannot be decrypted. This can happen if the key material for a KMS key in an AWS CloudHSM key store is deleted from the associated AWS CloudHSM cluster. However, it might be possible to recover the key material.

When you create an AWS KMS key (KMS key) in an AWS CloudHSM key store, AWS KMS logs into the associated AWS CloudHSM cluster and creates the key material for the KMS key. It also changes the password to a value that only it knows and remains logged in as long as the AWS CloudHSM key store is connected. Because only the key owner, that is, the CU who created a key, can delete the key, it is unlikely that the key will be deleted from the HSMs accidentally.

However, if the key material for a KMS key is deleted from the HSMs in a cluster, the key state of the KMS key eventually changes to `UNAVAILABLE`. If you attempt to use the KMS key for a cryptographic operation, the operation fails with a `KMSInvalidStateException` exception. Most importantly, any data that was encrypted under the KMS key cannot be decrypted.

Under certain circumstances, you can recover deleted key material by creating a cluster from a backup that contains the key material. This strategy works only when at least one backup was created while the key existed and before it was deleted.

Use the following process to recover the key material.

1. Find a cluster backup that contains the key material. The backup must also contain all users and keys that you need to support the cluster and its encrypted data.

   Use the DescribeBackups operation to list the backups for a cluster. Then use the backup timestamp to help you select a backup. To limit the output to the cluster that is associated with the AWS CloudHSM key store, use the `Filters` parameter, as shown in the following example.

   ```
   $ aws cloudhsmv2 describe-backups --filters clusterIds=<cluster ID>
   {
       "Backups": [
           {
               "ClusterId": "cluster-1a23b4cdefg",
               "BackupId": "backup-9g87f6edcba",
               "CreateTimestamp": 1536667238.328,
               "BackupState": "READY"
           },
                ...
       ]
   }
   ```

2. Create a cluster from the selected backup. Verify that the backup contains the deleted key and other users and keys that the cluster requires.
3. Disconnect the AWS CloudHSM key store so you can edit its properties.
4. Edit the cluster ID of the AWS CloudHSM key store. Enter the cluster ID of the cluster that you created from the backup. Because the cluster shares a backup history with the original cluster, the new cluster ID should be valid.
5. Reconnect the AWS CloudHSM key store.

How to log in as kmsuser

To create and manage the key material in the AWS CloudHSM cluster for your AWS CloudHSM key store, AWS KMS uses the kmsuser crypto user (CU) account. You create the kmsuser CU account in your cluster and provide its password to AWS KMS when you create your AWS CloudHSM key store.

In general, AWS KMS manages the kmsuser account. However, for some tasks, you need to disconnect the AWS CloudHSM key store, log into the cluster as the kmsuser CU, and use the cloudhsm_mgmt_util and key_mgmt_util command line tools.

> **Note**
>
> While a custom key store is disconnected, all attempts to create KMS keys in the custom key store or to use existing KMS keys in cryptographic operations will fail. This action can prevent users from storing and accessing sensitive data.

This topic explains how to disconnect your AWS CloudHSM key store and log in as kmsuser, run the AWS CloudHSM command line tool, and log out and reconnect your AWS CloudHSM key store.

Topics

How to disconnect and log in

Use the following procedure each time to need to log into an associated cluster as the kmsuser CU.

1. Disconnect the AWS CloudHSM key store, if it is not already disconnected. You can use the AWS KMS console or AWS KMS API.

   While your AWS CloudHSM key is connected, AWS KMS is logged in as the kmsuser. This prevents you from logging in as kmsuser or changing the kmsuser password.

   For example, this command uses DisconnectCustomKeyStore to disconnect an example key store. Replace the example AWS CloudHSM key store ID with a valid one.

   ```
   $ aws kms disconnect-custom-key-store --custom-key-store-id cks-1234567890abcdef0
   ```

2. Start cloudhsm_mgmt_util. Use the procedure described in Prepare to run cloudhsm_mgmt_util section of the *AWS CloudHSM User Guide*.

3. Log into cloudhsm_mgmt_util on the AWS CloudHSM cluster as a crypto officer (CO).

   For example, this command logs in as a CO named admin. Replace the example CO user name and password with valid values.

```
aws-cloudhsm>loginHSM CO admin <password>
loginHSM success on server 0(10.0.2.9)
loginHSM success on server 1(10.0.3.11)
loginHSM success on server 2(10.0.1.12)
```

4. Use the changePswd command to change the password of the `kmsuser` account to one that you know. (AWS KMS rotates the password when you connect your AWS CloudHSM key store.) The password must consist of 7-32 alphanumeric characters. It is case-sensitive and cannot contain any special characters.

   For example, this command changes the `kmsuser` password to `tempPassword`.

```
aws-cloudhsm>changePswd CU kmsuser tempPassword

*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. Cav server does NOT synchronize these changes with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************

Do you want to continue(y/n)?y
Changing password for kmsuser(CU) on 3 nodes
```

5. Log into key_mgmt_util or cloudhsm_mgmt_util as `kmsuser` using the password that you set. For detailed instructions, see Getting Started with cloudhsm_mgmt_util and Getting Started with key_mgmt_util. The tool that you use depends on your task.

   For example, this command logs into key_mgmt_util.

```
Command: loginHSM -u CU -s kmsuser -p tempPassword
Cfm3LoginHSM returned: 0x00 : HSM Return: SUCCESS

Cluster Error Status
```

```
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

How to log out and reconnect

1. Perform the task, then log out of the command line tool. If you do not log out, attempts to reconnect your AWS CloudHSM key store will fail.

   ```
   Command:  logoutHSM
   Cfm3LogoutHSM returned: 0x00 : HSM Return: SUCCESS

   Cluster Error Status
   Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
   Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
   ```

2. Edit the `kmsuser` password setting for the custom key store.

   This tells AWS KMS the current password for `kmsuser` in the cluster. If you omit this step, AWS KMS will not be able to log into the cluster as `kmsuser`, and all attempts to reconnect your custom key store will fail. You can use the AWS KMS console or the `KeyStorePassword` parameter of the UpdateCustomKeyStore operation.

   For example, this command tells AWS KMS that the current password is `tempPassword`. Replace the example password with the actual one.

   ```
   $ aws kms update-custom-key-store --custom-key-store-id cks-1234567890abcdef0 --
   key-store-password tempPassword
   ```

3. Reconnect the AWS KMS key store to its AWS CloudHSM cluster. Replace the example AWS CloudHSM key store ID with a valid one. During the connection process, AWS KMS changes the `kmsuser` password to a value that only it knows.

   The ConnectCustomKeyStore operation returns quickly, but the connection process can take an extended period of time. The initial response does not indicate the success of the connection process.

   ```
   $ aws kms connect-custom-key-store --custom-key-store-id cks-1234567890abcdef0
   ```

4. Use the DescribeCustomKeyStores operation to verify that the AWS CloudHSM key store is connected. Replace the example AWS CloudHSM key store ID with a valid one.

   In this example, the connection state field shows that the AWS CloudHSM key store is now connected.

```
$ aws kms describe-custom-key-stores --custom-key-store-
id cks-1234567890abcdef0
{
   "CustomKeyStores": [
      "CustomKeyStoreId": "cks-1234567890abcdef0",
      "CustomKeyStoreName": "ExampleKeyStore",
      "CloudHsmClusterId": "cluster-1a23b4cdefg",
      "TrustAnchorCertificate": "<certificate string appears here>",
      "CreationDate": "1.499288695918E9",
      "ConnectionState": "CONNECTED"
   ],
}
```

External key stores

External key stores allow you to protect your AWS resources using cryptographic keys outside of AWS. This advanced feature is designed for regulated workloads that you must protect with encryption keys stored in an external key management system that you control. External key stores support the AWS digital sovereignty pledge to give you sovereign control over your data in AWS, including the ability to encrypt with key material that you own and control outside of AWS.

An *external key store* is a custom key store backed by an *external key manager* that you own and manage outside of AWS. Your external key manager can be a physical or virtual hardware security modules (HSMs), or any hardware-based or software-based system capable of generating and using cryptographic keys. Encryption and decryption operations that use a KMS key in an external key store are performed by your external key manager using your cryptographic key material, a feature known as *hold your own keys* (HYOKs).

AWS KMS never interacts directly with your external key manager, and cannot create, view, manage, or delete your keys. Instead, AWS KMS interacts only with external key store proxy (XKS proxy) software that you provide. Your external key store proxy mediates all communication between AWS KMS and your external key manager. It transmits all requests from AWS KMS to your external key manager, and transmits responses from your external key manager back to AWS KMS.

The external key store proxy also translates generic requests from AWS KMS into a vendor-specific format that your external key manager can understand, allowing you to use external key stores with key managers from a variety of vendors.

You can use KMS keys in an external key store for client-side encryption, including with the AWS Encryption SDK. But external key stores are an important resource for server-side encryption, allowing you to protect your AWS resources in multiple AWS services with your cryptographic keys outside of AWS. AWS services that support customer managed keys for symmetric encryption also support KMS keys in an external key store. For service support details, see AWS Service Integration.

External key stores allow you to use AWS KMS for regulated workloads where encryption keys must be stored and used outside of AWS. But they are a major departure from the standard shared responsibility model, and require additional operational burdens. The greater risk to availability and latency will, for most customers, exceed the perceived security benefits of external key stores.

External key stores let you control the root of trust. Data encrypted under KMS keys in your external key store can be decrypted only by using the external key manager that you control. If you temporarily revoke access to your external key manager, such as by disconnecting the external key store or disconnecting your external key manager from the external key store proxy, AWS loses all access to your cryptographic keys until you restore it. During that interval, ciphertext encrypted under your KMS keys can't be decrypted. If you permanently revoke access to your external key manager, all ciphertext encrypted under a KMS key in your external key store becomes unrecoverable. The only exceptions are AWS services that briefly cache the data keys protected by your KMS keys. These data keys continue to work until you deactivate the resource or the cache expires. For details, see How unusable KMS keys affect data keys.

External key stores unblock the few use cases for regulated workloads where encryption keys must remain solely under your control and inaccessible to AWS. But this is a major change in the way you operate cloud-based infrastructure and a significant shift in the shared responsibility model. For most workloads, the additional operational burden and greater risks to availability and performance will exceed the perceived security benefits of external key stores.

Learn more:

- Announcing AWS KMS External Key Store in the *AWS News Blog*.

Do I need an external key store?

For most users, the default AWS KMS key store, which is protected by FIPS 140-2 Security Level 3 validated hardware security modules, fulfills their security, control, and regulatory requirements. External key store users incur substantial cost, maintenance, and troubleshooting burden, and risks to latency, availability and reliability.

When considering an external key store, take some time to understand the alternatives, including an AWS CloudHSM key store backed by an AWS CloudHSM cluster that you own and manage, and KMS keys with imported key material that you generate in your own HSMs and can delete from KMS keys on demand. In particular, importing key material with a very brief expiration interval might provide a similar level of control without the performance or availability risks.

An external key store might be the right solution for your organization if you have the following requirements:

- You are required to use cryptographic keys in your on-premises key manager or a key manager outside of AWS that you control.
- You must demonstrate that your cryptographic keys are retained solely under your control outside of the cloud.
- You must encrypt and decrypt using cryptographic keys with independent authorization.
- Key material must be subject to a secondary, independent audit path.

If you choose an external key store, limit its use to workloads that require protection with cryptographic keys outside of AWS.

Shared responsibility model

Standard KMS keys use key material that is generated and used in HSMs that AWS KMS owns and manages. You establish the access control policies on your KMS keys and configure AWS services that use KMS keys to protect your resources. AWS KMS assumes responsibility for the security, availability, latency, and durability of the key material in your KMS keys.

KMS keys in external key stores rely on key material and operations in your external key manager. As such, the balance of responsibility shifts in your direction. You are responsible for the security, reliability, durability, and performance of the cryptographic keys in your external key manager. AWS KMS is responsible for responding promptly to requests and communicating with your external key store proxy, and for maintaining our security standards. To ensure that every external key store ciphertext at least as strong than standard AWS KMS ciphertext, AWS KMS first encrypts all plaintext with AWS KMS key material specific to your KMS key, and then sends it to your external key manager for encryption with your external key, a procedure known as *double encryption*. As a result, neither AWS KMS nor the owner of the external key material can decrypt double-encrypted ciphertext alone.

You are responsible for maintaining an external key manager that meet your regulatory and performance standards, for supplying and maintaining an external key store proxy that conforms to the AWS KMS External Key Store Proxy API Specification, and for ensuring the availability and durability of your key material. You must also create, configure, and maintain an external key store. When errors arise that are caused by components that you maintain, you must be prepared to identify and resolve the errors so that AWS services can access your resources without undue disruption. AWS KMS provides troubleshooting guidance to help you determine the cause of problems and the most likely resolutions.

Review the Amazon CloudWatch metrics and dimensions that AWS KMS records for external key stores. AWS KMS strongly recommends that you create CloudWatch alarms to monitor your external key store so you can detect the early signs of performance and operational problems before they occur.

What is changing?

External key stores support only symmetric encryption KMS keys. Within AWS KMS, you use and manage KMS keys in an external key store in much the same way that you manage other customer managed keys, including setting access control policies and monitoring key use. You use the same APIs with the same parameters to request a cryptographic operation with a KMS key in an external key store that you use for any KMS key. Pricing is also the same as for standard KMS keys. For details, see Managing KMS keys in an external key store, Using KMS keys in an external key store, and AWS Key Management Service Pricing.

However, with external key stores the following principles change:

- You are responsible for the availability, durability, and latency of key operations.
- You are responsible for all costs for developing, purchasing, operating, and licensing your external key manager system.
- You can implement independent authorization of all requests from AWS KMS to your external key store proxy.
- You can monitor, audit, and log all operations of your external key store proxy, and all operations of your external key manager related to AWS KMS requests.

Where do I start?

To create and manage an external key store, you need to choose your external key store proxy connectivity option, assemble the prerequisites, and create and configure your external key store. To begin, see Planning an external key store.

Quotas

AWS KMS allows up to 10 custom key stores in each AWS account and Region, including both AWS CloudHSM key stores and external key stores, regardless of their connection state. In addition, there are AWS KMS request quotas on the use of KMS keys in an external key store.

If you choose VPC proxy connectivity for your external key store proxy, there might also be quotas on the required components, such as VPCs, subnets, and network load balancers. For information about these quotas, use the Service Quotas console.

Regions

To minimize network latency, create your external key store components in the AWS Region closest to your external key manager. If possible, choose a Region with a network round-trip time (RTT) of 35 milliseconds or less.

External key stores are supported in all AWS Regions in which AWS KMS is supported except for China (Beijing) and China (Ningxia).

Unsupported features

AWS KMS does not support the following features in custom key stores.

- Asymmetric KMS keys
- Asymmetric data key pairs
- HMAC KMS keys
- KMS keys with imported key material
- Automatic key rotation
- Multi-Region keys

Topics

- How external key stores work
- Controlling access to your external key store
- Planning an external key store
- Managing an external key store
- Managing KMS keys in an external key store
- Troubleshooting external key stores

External key store concepts

This topic explains some of the concepts used in external key stores.

Topics

- External key store
- External key manager
- External key
- External key store proxy
- External key store proxy connectivity
- External key store proxy authentication credential
- Proxy APIs
- Double encryption

External key store

An *external key store* is an AWS KMS custom key store backed by an external key manager outside of AWS that you own and manage. Each KMS key in an external key store is associated with an external key in your external key manager. When you use a KMS key in an external key store for encryption or decryption, the operation is performed in your external key manager using your external key, an arrangement known as *Hold your Own Keys* (HYOK). This feature is designed for organizations that are required to maintain their cryptographic keys in their own external key manager.

External key stores ensure that the cryptographic keys and operations that protect your AWS resources remain in your external key manager under your control. AWS KMS sends requests to your external key manager to encrypt and decrypt data, but AWS KMS cannot create, delete, or manage any external keys. All requests from AWS KMS to your external key manager are mediated by an external key store proxy software component that you supply, own, and manage.

AWS services that support AWS KMS customer managed keys can use the KMS keys in your external key store to protect your data. As a result, your data is ultimately protected by your keys using your encryption operations in your external key manager.

The KMS keys in an external key store have fundamentally different trust models, shared responsibility arrangements, and performance expectations than standard KMS keys. With external key stores, you are responsible for the security and integrity of the key material and the cryptographic operations. The availability and latency of KMS keys in an external key store are affected by the hardware, software, networking components, and the distance between AWS KMS and your external key manager. You are also likely to incur additional costs for your external key manager and for the networking and load balancing infrastructure you need for your external key manager to communicate with AWS KMS

You can use your external key store as part of your broader data protection strategy. For each AWS resource that you protect, you can decide which require a KMS key in an external key store and which can be protected by a standard KMS key. This gives you the flexibility to chose KMS keys for specific data classifications, applications, or projects.

External key manager

An *external key manager* is a component outside of AWS that can generate 256-bit AES symmetric keys and perform symmetric encryption and decryption. The external key manager for an external key store can be a physical hardware security module (HSM), a virtual HSM, or a software key manager with or without an HSM component. It can be located anywhere outside of AWS, including on your premises, in a local or remote data center, or in any cloud. Your external key store can be backed by a single external key manager or multiple related key manager instances that share cryptographic keys, such as an HSM cluster. External key stores are designed to support a variety of external managers from different vendors. For details about the requirements for your external key manager, see Planning an external key store.

External key

Each KMS key in an external key store is associated with a cryptographic key in your external key manager known as an *external key*. When you encrypt or decrypt with a KMS key in your external key store, the cryptographic operation is performed in your external key manager using your external key.

⚠ **Warning**

The external key is essential to the operation of the KMS key. If the external key is lost or deleted, ciphertext encrypted under the associated KMS key is unrecoverable.

For external key stores, an external key must be a 256-bit AES key that is enabled and can perform encryption and decryption. For detailed external key requirements, see Requirements for a KMS key in an external key store.

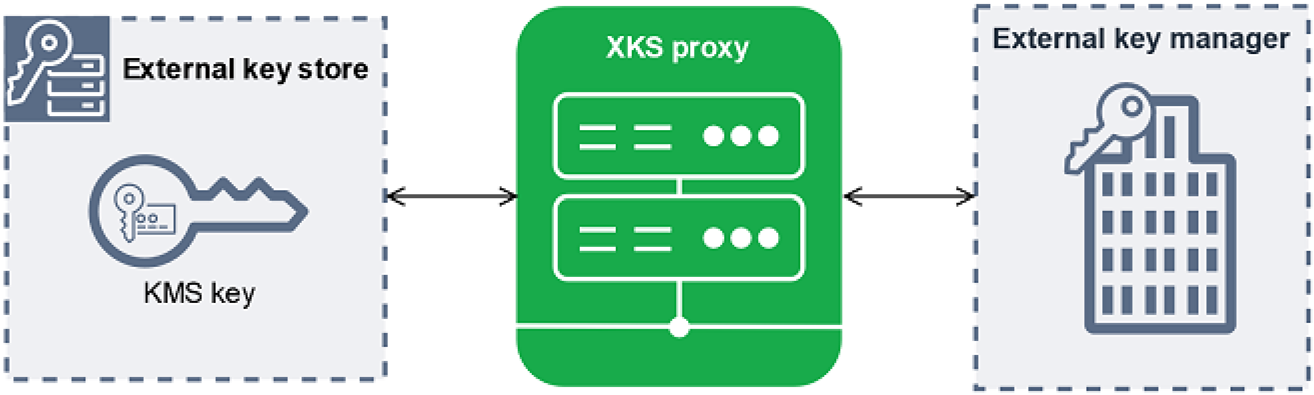
AWS KMS cannot create, delete, or manage any external keys. Your cryptographic key material never leaves your external key manager.When you create a KMS key in an external key store, you provide the ID of an external key (`XksKeyId`). You cannot change the external key ID associated with a KMS key, although your external key manager can rotate the key material associated with the external key ID.

In addition to your external key, a KMS key in an external key store also has AWS KMS key material. Data protected by the KMS key is encrypted first by AWS KMS using the AWS KMS key material, and then by your external key manager using your external key. This double encryption process ensures that ciphertext protected by your KMS key is always at least as strong as ciphertext protected only by AWS KMS.

Many cryptographic keys have different types of identifiers. When creating a KMS key in an external key store, provide the ID of the external key that the external key store proxy uses to refer to the external key. If you use the wrong identifier, your attempt to create a KMS key in your external key store fails.

External key store proxy

The *external key store proxy* ("XKS proxy") is a customer-owned and customer-managed software application that mediates all communication between AWS KMS and your external key manager. It also translates generic AWS KMS requests into a format that your vendor-specific external key manager understand. An external key store proxy is required for an external key store. Each external key store is associated with one external key store proxy.



AWS KMS cannot create, delete, or manage any external keys. Your cryptographic key material never leaves your external key manager. All communication between AWS KMS and your external key manager is mediated by your external key store proxy. AWS KMS sends requests to the external key store proxy and receives responses from the external key store proxy. The external key store proxy is responsible for transmitting requests from AWS KMS to your external key manager and transmitting responses from your external key manager back to AWS KMS

You own and manage the external key store proxy for your external key store, and you are responsible for its maintenance and operation. You can develop your external key store proxy based on the open-source external key store proxy API specification that AWS KMS publishes or purchase a proxy application from a vendor. Your external key store proxy might be included in your external key manager. To support proxy development, AWS KMS also provides a sample external key store proxy (aws-kms-xks-proxy) and a test client (xks-kms-xksproxy-test-client) that verifies that your external key store proxy conforms to the specification.

To authenticate to AWS KMS, the proxy uses server-side TLS certificates. To authenticate to your proxy, AWS KMS signs all requests to your external key store proxy with a SigV4 proxy authentication credential. Optionally, your proxy can enable mutual TLS (mTLS) for additional assurance that it only accepts requests from AWS KMS.

Your external key store proxy must support HTTP/1.1 or later and TLS 1.2 or later with at least one of the following cipher suites:

- TLS_AES_256_GCM_SHA384 (TLS 1.3)
- TLS_CHACHA20_POLY1305_SHA256 (TLS 1.3)

> **Note**
>
> The AWS GovCloud (US) Region does not support TLS_CHACHA20_POLY1305_SHA256.

- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (TLS 1.2)
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (TLS 1.2)

To create and use the KMS keys in your external key store, you must first connect the external key store to its external key store proxy. You can also disconnect your external key store from its proxy on demand. When you do, all KMS keys in the external key store become unavailable; they cannot be used in any cryptographic operation.

External key store proxy connectivity

The external key store proxy connectivity ("XKS proxy connectivity") describes the method that AWS KMS uses to communicate with your external key store proxy.

You specify your proxy connectivity option when you create your external key store, and it becomes a property of the external key store. You can change your proxy connectivity option by updating the custom key store property, but you must be certain that your external key store proxy can still access the same external keys.

AWS KMS supports the following connectivity options:

- Public endpoint connectivity — AWS KMS sends requests for your external key store proxy over the internet to a public endpoint that you control. This option is simple to create and maintain, but it might not fulfill the security requirements for every installation.
- VPC endpoint service connectivity — AWS KMS sends requests to a Amazon Virtual Private Cloud (Amazon VPC) endpoint service that you create and maintain. You can host your external key store proxy inside your Amazon VPC, or host your external key store proxy outside of AWS and use the Amazon VPC only for communication.

For details about the external key store proxy connectivity options, see Choosing a proxy connectivity option.

External key store proxy authentication credential

To authenticate to your external key store proxy, AWS KMS signs all requests to your external key store proxy with a Signature V4 (SigV4) authentication credential. You establish and maintain the authentication credential on your proxy, then provide this credential to AWS KMS when you create your external store.

> **Note**
>
> The SigV4 credential that AWS KMS uses to sign requests to the XKS proxy is unrelated to any SigV4 credentials associated with AWS Identity and Access Management principals in your AWS accounts. Do not reuse any IAM SigV4 credentials for your external key store proxy.

Each proxy authentication credential has two parts. You must provide both parts when creating an external key store or updating the authentication credential for your external key store.

- Access key ID: Identifies the secret access key. You can provide this ID in plaintext.
- Secret access key: The secret part of the credential. AWS KMS encrypts the secret access key in the credential before storing it.

You can edit the credential setting at any time, such as when you enter incorrect values, when you change your credential on the proxy, or when your proxy rotates the credential. For technical details about AWS KMS authentication to the external key store proxy, see Authentication in the AWS KMS External Key Store Proxy API Specification.

To allow you to rotate your credential without disrupting the AWS services that use KMS keys in your external key store, we recommend that the external key store proxy support at least two valid authentication credentials for AWS KMS. This ensures that your previous credential continues to work while you provide your new credential to AWS KMS.

To help you track the age of your proxy authentication credential, AWS KMS defines an Amazon CloudWatch metric, XksProxyCredentialAge. You can use this metric to create a CloudWatch alarm that notifies you when the age of your credential reaches a threshold you establish.

To provide additional assurance that your external key store proxy responds only to AWS KMS, some external key proxies support mutual Transport Layer Security (mTLS). For details, see mTLS authentication (optional).

Proxy APIs

To support an AWS KMS external key store, an external key store proxy must implement the required proxy APIs as described in the AWS KMS External Key Store Proxy API Specification. These proxy API requests are the only requests that AWS KMS sends to the proxy. Although you never send these requests directly, knowing about them might help you fix any issues that might arise with your external key store or its proxy. For example, AWS KMS includes information about the latency and success rates of these API calls in its Amazon CloudWatch metrics for external key stores. For details, see Monitoring an external key store.

The following table lists and describes each of the proxy APIs. It also includes the AWS KMS operations that trigger a call to the proxy API and any AWS KMS operation exceptions related to the proxy API.

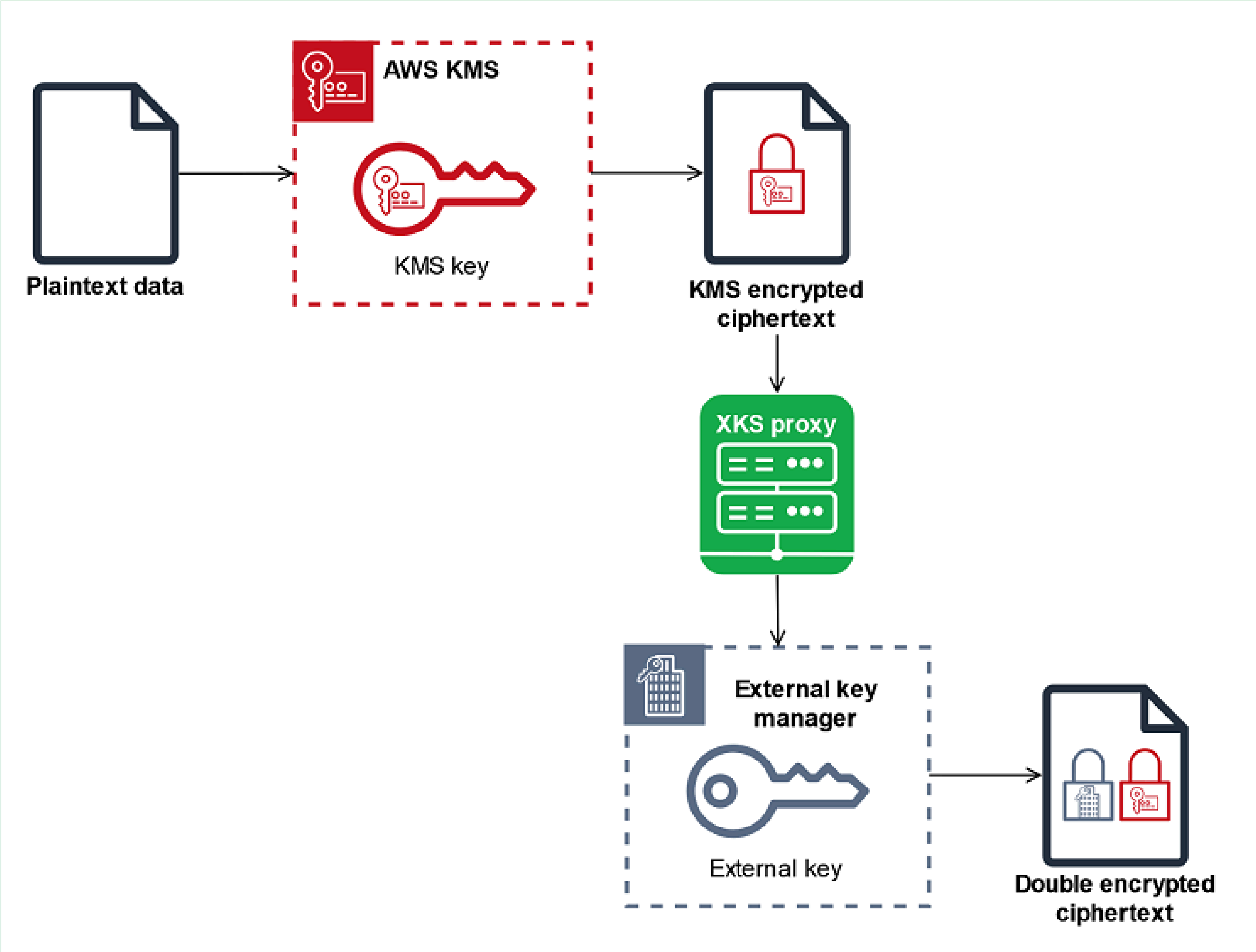
Proxy API	Description	Related AWS KMS operations
Decrypt	AWS KMS sends the ciphertext to be decrypted, and the ID of the external key to use. The required encryption algorithm is AES_GCM.	Decrypt, ReEncrypt
Encrypt	AWS KMS sends data to be encrypted , and the ID of the external key to use. The required encryption algorithm is AES_GCM.	Encrypt, GenerateDataKey, GenerateDataKeyWithoutPlaintext, ReEncrypt
GetHealth Status	AWS KMS requests information about the status of the proxy and your external key manager. The status of each external key manager can be one of the following. • `Active`: Healthy; can serve traffic • `Degraded`: Unhealthy, but can serve traffic • `Unavailable` : Unhealthy; cannot serve traffic	CreateCustomKeyStore (for public endpoint connectivity), ConnectCustomKeyStore (for VPC endpoint service connectivity) If all external key manager instances are `Unavailable` , attempts to create or connect the key store fail with `XksProxyUriUnreachableException` .

Proxy API	Description	Related AWS KMS operations
GetKeyMetadata	AWS KMS requests information about the external key associated with a KMS key in your external key store. The response includes the key spec (AES_256), the key usage ([ENCRYPT, DECRYPT]), and the whether the external key is ENABLED or DISABLED.	CreateKey If the key spec is not AES_256, or the key usage is not [ENCRYPT, DECRYPT], or the status is DISABLED, the CreateKey operation fails with XksKeyInvalidConfigurationException .

Double encryption

Data encrypted by a KMS key in an external key store is encrypted twice. First, AWS KMS encrypts the data with AWS KMS key material specific to the KMS key. Then the AWS KMS-encrypted ciphertext is encrypted by your external key manager using your external key. This process is known as *double encryption*.

Double encryption ensures that data encrypted by a KMS key in an external key store is at least as strong as ciphertext encrypted by a standard KMS key. It also protects your plaintext in transit from AWS KMS to your external key store proxy. With double encryption, you retain full control of your ciphertexts. If you permanently revoke AWS access to your external key through your external proxy, any ciphertext remaining in AWS is effectively crypto-shredded.



To enable double encryption, each KMS key in an external key store has *two* cryptographic backing keys:

- An AWS KMS key material unique to the KMS key. This key material is generated and only used in AWS KMS FIPS 140-2 Security Level 3 certified hardware security modules (HSMs).
- An external key in your external key manager.

Double encryption has the following effects:

- AWS KMS cannot decrypt any ciphertext encrypted by a KMS key in an external key store without access to your external keys via your external key store proxy.
- You cannot decrypt any ciphertext encrypted by a KMS key in an external key store outside of AWS, even if you have its external key material.

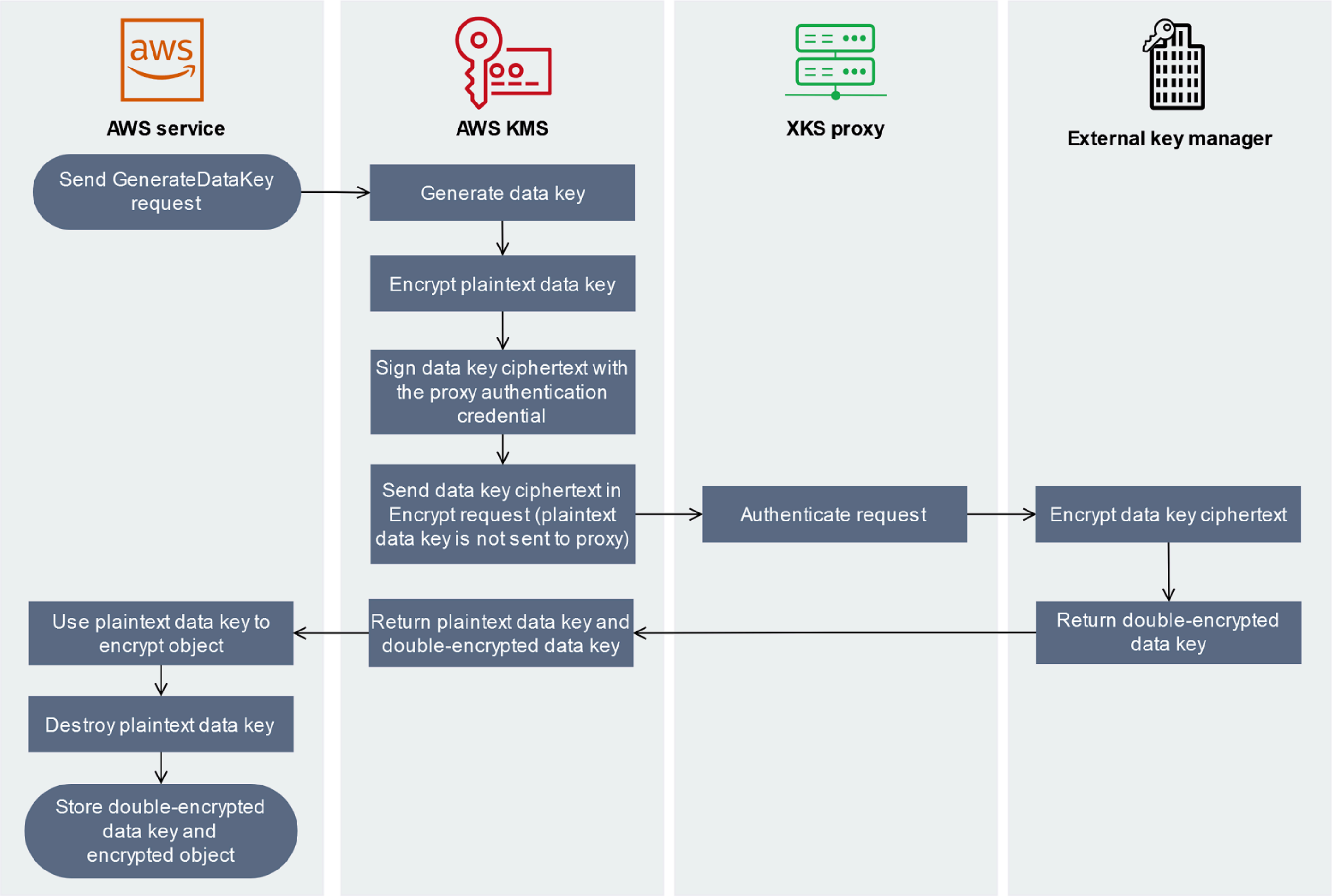
- You cannot recreate a KMS key that was deleted from an external key store, even if you have its external key material. Each KMS key has unique metadata that it includes in the symmetric ciphertext. A new KMS key would not be able to decrypt ciphertext encrypted by the original key, even if it used the same external key material.

For an example of double encryption in practice, see How external key stores work.

How external key stores work

Your external key store, external key store proxy, and external key manager work together to protect your AWS resources. The following procedure depicts the encryption workflow of a typical AWS service that encrypts each object under a unique data key protected by a KMS key. In this case, you've chosen a KMS key in an external key store to protect the object. The example shows how AWS KMS uses double encryption to protect the data key in transit and ensure that ciphertext generated by a KMS key in an external key store is always at least as strong as ciphertext encrypted by a standard symmetric KMS key with key material in AWS KMS.

The encryption methods used by each actual AWS service that integrates with AWS KMS vary. For details, see the "Data protection" topic in the Security chapter of the AWS service documentation.


AWS service
AWS KMS
XKS proxy
External key manager
Send GenerateDataKey request
Generate data key
Encrypt plaintext data key
Sign data key ciphertext with the proxy authentication credential
Send data key ciphertext in Encrypt request (plaintext data key is not sent to proxy)
Authenticate request
Encrypt data key ciphertext
Return double-encrypted data key
Return plaintext data key and double-encrypted data key
Use plaintext data key to encrypt object
Destroy plaintext data key
Store double-encrypted data key and encrypted object

1. You add a new object to your AWS service resource. To encrypt the object, the AWS service sends a GenerateDataKey request to AWS KMS using a KMS key in your external key store.
2. AWS KMS generates a 256-bit symmetric data key and prepares to send a copy of the plaintext data key to your external key manager via your external key store proxy. AWS KMS begins the double encryption process by encrypting the plaintext data key with the AWS KMS key material associated with the KMS key in the external key store.
3. AWS KMS sends an encrypt request to the external key store proxy associated with the external key store. The request includes the data key ciphertext to be encrypted and the ID of the external key that is associated with the KMS key. AWS KMS signs the request using the proxy authentication credential for your external key store proxy.

   The plaintext copy of the data key is not sent to the external key store proxy.
4. The external key store proxy authenticates the request, and then passes the encrypt request to your external key manager.

Some external key store proxies also implement an optional authorization policy that allows only selected principals to perform operations under specific conditions.

5. Your external key manager encrypts the data key ciphertext using the specified external key. The external key manager returns the double-encrypted data key to your external key store proxy, which returns it to AWS KMS.
6. AWS KMS returns the plaintext data key and the double-encrypted copy of that data key to the AWS service.
7. The AWS service uses the plaintext data key to encrypt the resource object, destroys the plaintext data key, and stores the encrypted data key with the encrypted object.

   Some AWS services might cache the plaintext data key to use for multiple objects, or to reuse while the resource is in use. For details, see How unusable KMS keys affect data keys.

To decrypt the encrypted object, the AWS service must send the encrypted data key back to AWS KMS in a Decrypt request. To decrypt the encrypted data key, AWS KMS must send the encrypted data key back to your external key store proxy with the ID of the external key. If the decrypt request to the external key store proxy fails for any reason, AWS KMS cannot decrypt the encrypted data key, and the AWS service cannot decrypt the encrypted object.

Controlling access to your external key store

All AWS KMS access control features — key policies, IAM policies, and grants — that you use with standard KMS keys, work the same way for KMS keys in an external key store. You can use IAM policies to control access to the API operations that create and manage external key stores. You use IAM policies and key policies to control access to the AWS KMS keys in your external key store. You can also use service control policies for your AWS organization and VPC endpoint policies to control access to KMS keys in your external key store.

We recommend that you provide users and roles only the permissions that they require for the tasks that they are likely to perform.

Topics

- mTLS authentication (optional)

Authorizing external key store managers

Principals who create and manage an external key store need permissions to the custom key store operations. The following list describes the minimum permissions required for external key store managers. Because a custom key store is not an AWS resource, you cannot provide permission to an external key store for principals in other AWS accounts.

- `kms:CreateCustomKeyStore`
- `kms:DescribeCustomKeyStores`
- `kms:ConnectCustomKeyStore`
- `kms:DisconnectCustomKeyStore`
- `kms:UpdateCustomKeyStore`
- `kms:DeleteCustomKeyStore`

Principals who create an external key store need permission to create and configure the external key store components. Principals can create external key stores only in their own accounts.
To create an external key store with VPC endpoint service connectivity, principals must have permission to create the following components:

- An Amazon VPC
- Public and private subnets
- A network load balancer and target group
- An Amazon VPC endpoint service

For details, see Identity and access management for Amazon VPC, Identity and access management for VPC endpoints and VPC endpoint services and Elastic Load Balancing API permissions.

Authorizing users of KMS keys in external key stores

Principals who create and manage AWS KMS keys in your external key store require the same permissions as those who create and manage any KMS key in AWS KMS. The default key policy for KMS key in an external key store is identical to the default key policy for KMS keys in AWS KMS. Attribute-based access control (ABAC), which uses tags and aliases to control access to KMS keys, is also effective on KMS keys in an external key stores.

Principals who use the KMS keys in your custom key store for cryptographic operations need permission to perform the cryptographic operation with the KMS key, such as kms:Decrypt. You can provide these permissions in an IAM or key policy. But, they do not need any additional permissions to use a KMS key in a custom key store.

To set a permission that applies only to KMS keys in an external key store, use the `kms:KeyOrigin` policy condition with a value of `EXTERNAL_KEY_STORE`. You can use this condition to limit the kms:CreateKey permission or any permission that is specific to a KMS key resource. For example, the following IAM policy allows the identity to which it is attached to call the specified operations on all KMS keys in the account, provided that the KMS keys are in an external key store. Notice that you can limit the permission to KMS keys in an external key store, and KMS keys in an AWS account, but not to any particular external key store in the account.

```
{
  "Sid": "AllowKeysInExternalKeyStores",
  "Effect": "Allow",
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:DescribeKey"
  ],
  "Resource": "arn:aws:kms:us-west-2:111122223333:key/*",
  "Condition": {
    "StringEquals": {
      "kms:KeyOrigin": "EXTERNAL_KEY_STORE"
    }
  }
}
```

Authorizing AWS KMS to communicate with your external key store proxy

AWS KMS communicates with your external key manager only through the external key store proxy that you provide. AWS KMS authenticates to your proxy by signing its requests using the Signature Version 4 (SigV4) process with the external key store proxy authentication credential that you specify. If you are using public endpoint connectivity for your external key store proxy, AWS KMS does not require any additional permissions.

However, if you are using VPC endpoint service connectivity, you must give AWS KMS permission to create an interface endpoint to your Amazon VPC endpoint service. This permission is required regardless of whether the external key store proxy is in your VPC or the external key store proxy is located elsewhere, but uses the VPC endpoint service to communicate with AWS KMS.

To allow AWS KMS to create an interface endpoint, use the Amazon VPC console or the ModifyVpcEndpointServicePermissions operation. Allow permissions for the following principal: `cks.kms.<region>.amazonaws.com`.

For example, the following AWS CLI command allows AWS KMS to connect to the specified VPC endpoint service in the US West (Oregon) (us-west-2) Region. Before using this command, replace the Amazon VPC service ID and AWS Region with valid values for your configuration.

```
modify-vpc-endpoint-service-permissions
--service-id vpce-svc-12abc34567def0987
--add-allowed-principals '["cks.kms.us-west-2.amazonaws.com"]'
```

To remove this permission, use the Amazon VPC console or the ModifyVpcEndpointServicePermissions with the `RemoveAllowedPrincipals` parameter.

External key store proxy authorization (optional)

Some external key store proxies implement authorization requirements for the use of its external keys. An external key store proxy is permitted, but not required, to design and implement an authorization scheme that allows particular users to request particular operations only under certain conditions. For example, a proxy might be configured to allow user A to encrypt with a particular external key, but not to decrypt with it.

Proxy authorization is independent of the SigV4-based proxy authentication that AWS KMS requires for all external key store proxies. It is also independent of the key policies, IAM policies, and grants that authorize access to operations affecting the external key store or its KMS keys.

To enable authorization by the external key store proxy, AWS KMS includes metadata in each proxy API request, including the caller, the KMS key, the AWS KMS operation, the AWS service (if any). The request metadata for version 1 (v1) of the external key proxy API is as follows.

```
"requestMetadata": {
    "awsPrincipalArn": string,
    "awsSourceVpc": string, // optional
    "awsSourceVpce": string, // optional
    "kmsKeyArn": string,
    "kmsOperation": string,
    "kmsRequestId": string,
    "kmsViaService": string // optional
}
```

For example, you might configure your proxy to allow requests from a particular principal (`awsPrincipalArn`), but only when the request is made on the principal's behalf by a particular AWS service (`kmsViaService`).

If proxy authorization fails, the related AWS KMS operation fails with a message that explains the error. For details , see Proxy authorization issues.

mTLS authentication (optional)

To enable your external key store proxy to authenticate requests from AWS KMS, AWS KMS signs all requests to your external key store proxy with the Signature V4 (SigV4) proxy authentication credential for your external key store.

To provide additional assurance that your external key store proxy responds only to AWS KMS requests, some external key proxies support *mutual Transport Layer Security* (mTLS), in which both parties to a transaction use certificates to authenticate to each other. mTLS adds client-side authentication — where the external key store proxy server authenticates the AWS KMS client — to the server-side authentication that standard TLS provides. In the rare case that your proxy authentication credential is compromised, mTLS prevents a third party from making successful API requests to the external key store proxy.

To implement mTLS, configure your external key store proxy to accept only client-side TLS certificates with the following properties:

- The subject common name on the TLS certificate must be `cks.kms.`*`<Region>`*`.amazonaws.com`, for example, `cks.kms.`*`eu-west-3`*`.amazonaws.com`.
- The certificate must be chained to a certificate authority associated with Amazon Trust Services.

Planning an external key store

Before creating your external key store, choose the connectivity option that determines how AWS KMS communicates with your external key store components. The connectivity option that you choose determines the remainder of the planning process.

Learn more:

- Review the process for creating an external key store, including assembling the prerequisites. It will help you to ensure that you have all of the components you need when you create your external key store.
- Learn how to control access to your external key store, including the permissions that external key store administrators and users require.
- Learn about the Amazon CloudWatch metrics and dimensions that AWS KMS records for external key stores. We strongly recommend that you create alarms to monitor your external key store so you can detect the early signs of performance and operational problems.

Choosing a proxy connectivity option

If you are creating an external key store, you need to determine how AWS KMS communicates with your external key store proxy. This choice will determine which components you need and how you configure them. AWS KMS supports the following connectivity options. Choose the option that meets your performance and security goals.

Before you begin, confirm that you need an external key store. Most customer can use KMS keys backed by AWS KMS key material.

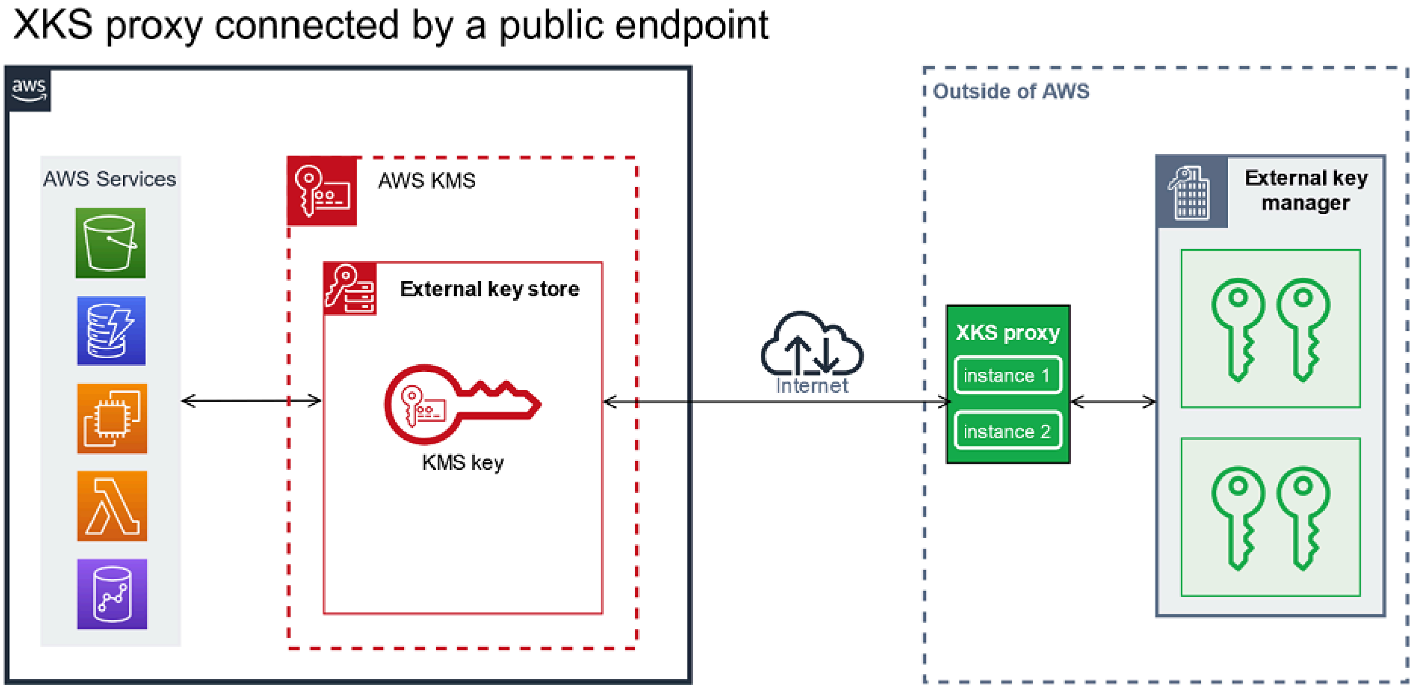
> **Note**
>
> If your external key store proxy is built into your external key manager, your connectivity might be predetermined. For guidance, consult the documentation for your external key manager or external key store proxy.

You can change your external key store proxy connectivity option even on an operating external key store. However, the process must be carefully planned and executed to minimize disruption, avoid errors, and ensure continued access to the cryptographic keys that encrypt your data.

Public endpoint connectivity

AWS KMS connects to the external key store proxy (XKS proxy) over the internet using a public endpoint.

This connectivity option is easier to set up and maintain, and it aligns well with some models of key management. However, it might not fulfill the security requirements of some organizations.



Requirements

If you choose public endpoint connectivity, the following are required.

- Your external key store proxy must be reachable at a publicly routable endpoint.
- You can use the same public endpoint for multiple external key stores provided that they use different proxy URI path values.
- You cannot use the same endpoint for an external key store with public endpoint connectivity and any external key store with VPC endpoint services connectivity in the same AWS Region, even if the key stores are in different AWS accounts.
- You must obtain a TLS certificate issued by a public certificate authority supported for external key stores. For a list, see Trusted Certificate Authorities.

  The subject common name (CN) on the TLS certificate must match the domain name in the proxy URI endpoint for the external key store proxy. For example, if the public endpoint is `https://myproxy.xks.example.com`, the TLS, the CN on the TLS certificate must be `myproxy.xks.example.com` or `*.xks.example.com`.
- Ensure that any firewalls between AWS KMS and the external key store proxy allow traffic to and from port 443 on the proxy. AWS KMS communicates on port 443. This value is not configurable.

For all requirements for an external key store, see the Assemble the prerequisites.

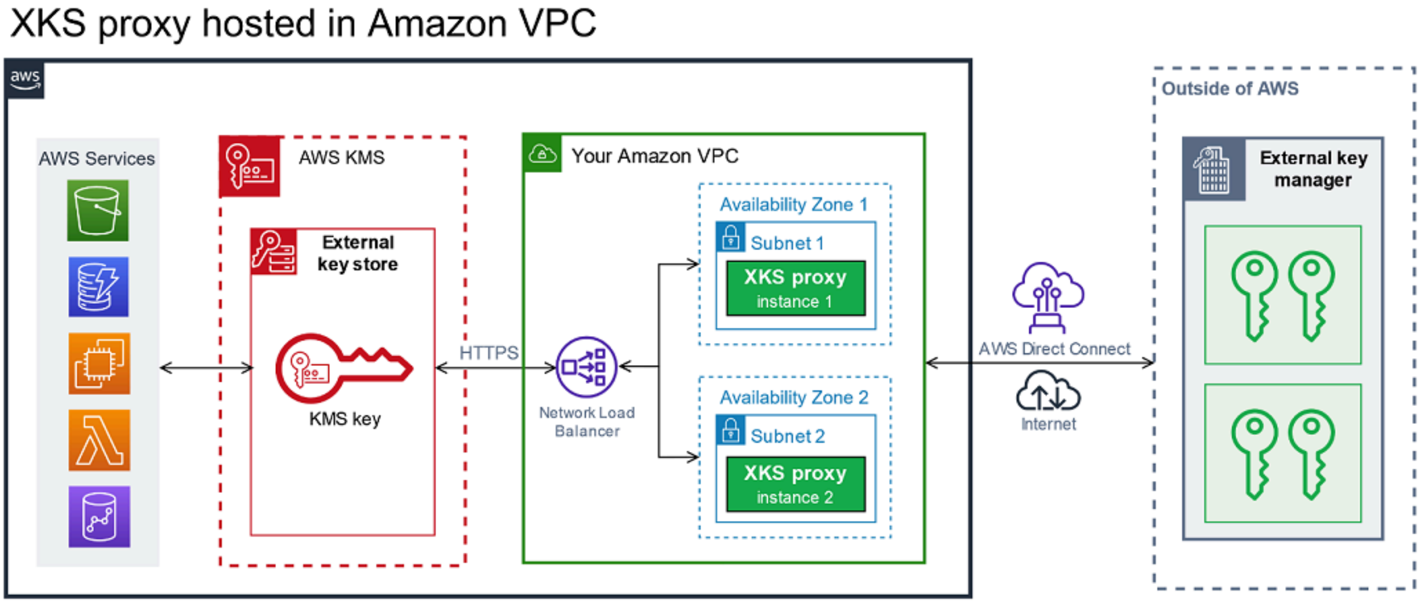
VPC endpoint service connectivity

AWS KMS connects to the external key store proxy (XKS proxy) by creating an interface endpoint to an Amazon VPC endpoint service that you create and configure. You are responsible for creating the VPC endpoint service and for connecting your VPC to your external key manager.

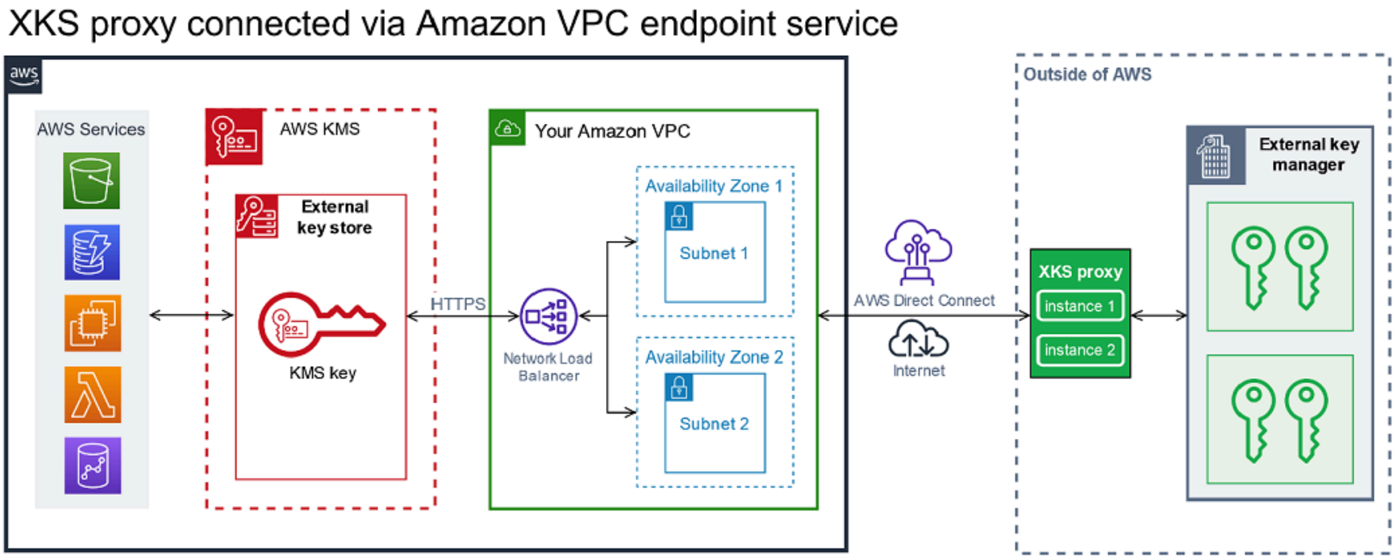
Your endpoint service can use any of the supported network-to-Amazon VPC options for communications, including AWS Direct Connect.

This connectivity option is more complicated to set up and maintain. But it uses AWS PrivateLink, which enables AWS KMS to privately connect to your Amazon VPC and your external key store proxy without using the public internet.

You can locate your external key store proxy in your Amazon VPC.



Or, locate your external key store proxy outside of AWS and use your Amazon VPC endpoint service only for secure communication with AWS KMS.



Configuring VPC endpoint service connectivity

Use the guidance in this section to create and configure the AWS resources and related components that are required for an external key store that uses VPC endpoint service connectivity. The resources listed for this connectivity option are a supplement to the resources required for all external key stores. After you create and configure the required resources, you can create your external key store.

You can locate your external key store proxy in your Amazon VPC or locate the proxy outside of AWS and use your VPC endpoint service for communication.

Before you begin, confirm that you need an external key store. Most customer can use KMS keys backed by AWS KMS key material.

> **Note**
>
> Some of the elements required for VPC endpoint service connectivity might be included in your external key manager. Also, your software might have additional configuration requirements. Before creating and configuring the AWS resources in this section, consult your proxy and key manager documentation.

Topics

- Creating a target group
- Creating a network load balancer
- Creating a VPC endpoint service
- Verifying your private DNS name domain
- Authorizing AWS KMS to connect to the VPC endpoint service

Requirements for VPC endpoint service connectivity

If you choose VPC endpoint service connectivity for your external key store, the following resources are required.

To minimize network latency, create your AWS components in the supported AWS Region that is closest to your external key manager. If possible, choose a Region with a network round-trip time (RTT) of 35 milliseconds or less.

- An Amazon VPC that is connected to your external key manager. It must have at least two private subnets in two different Availability Zones.

  You can use an existing Amazon VPC for your external key store, provided that it fulfills the requirements for use with an external key store. Multiple external key stores can share an Amazon VPC, but each external key store must have its own VPC endpoint service and private DNS name.
- An Amazon VPC endpoint service powered by AWS PrivateLink with a network load balancer and target group.

  The endpoint service cannot require acceptance. Also, you must add AWS KMS as an allowed principal. This allows AWS KMS to create interface endpoints so it can communicate with your external key store proxy.
- A private DNS name for the VPC endpoint service that is unique in its AWS Region.

  The private DNS name must be a subdomain of a higher-level public domain. For example, if the private DNS name is `myproxy-private.xks.example.com`, it must be a subdomain of a public domain such as `xks.example.com` or `example.com`.

  You must verify ownership of the DNS domain for private DNS name.
- A TLS certificate issued by a supported public certificate authority for your external key store proxy.

The subject common name (CN) on the TLS certificate must match the private DNS name. For example, if the private DNS name is `myproxy-private.xks.example.com`, the CN on the TLS certificate must be `myproxy-private.xks.example.com` or `*.xks.example.com`.

For all requirements for an external key store, see the Assemble the prerequisites.

Creating an Amazon VPC and subnets

VPC endpoint service connectivity requires an Amazon VPC that is connected to your external key manager with at least two private subnets. You can create an Amazon VPC or use an existing Amazon VPC that fulfills the requirements for external key stores. For help with creating a new Amazon VPC, see Create a VPC in the *Amazon Virtual Private Cloud User Guide*.

Requirements for your Amazon VPC

To work with external key stores using VPC endpoint service connectivity, the Amazon VPC must have the following properties:

- Must be in the same AWS account and supported Region as your external key store.
- Requires at least two private subnets, each in a different Availability Zone.
- The private IP address range of your Amazon VPC must not overlap with the private IP address range of the data center hosting your external key manager.
- All components must use IPv4.

You have many options for connecting the Amazon VPC to your external key store proxy. Choose an option that meets your performance and security needs. For a list, see Connect your VPC to other networks and Network-to-Amazon VPC connectivity options. For more details, see AWS Direct Connect, and the AWS Site-to-Site VPN User Guide.

Creating an Amazon VPC for your external key store

Use the following instructions to create the Amazon VPC for your external key store. An Amazon VPC is required only if you choose the VPC endpoint service connectivity option. You can use an existing Amazon VPC that fulfills the requirements for an external key store.

Follow the instructions in the Create a VPC, subnets, and other VPC resources topic using the following required values. For other fields, accept the default values and provide names as requested.

Field	Value
IPv4 CIDR block	Enter the IP addresses for your VPC. The private IP address range of your Amazon VPC must not overlap with the private IP address range of the data center hosting your external key manager.
Number of Availability Zones (AZs)	2 or more
Number of public subnets	None are required (0)
Number of private subnets	One for each AZ
NAT gateways	None are required.
VPC endpoints	None are required.
Enable DNS hostnames	Yes
Enable DNS resolution	Yes

Be sure to test your VPC communication. For example, if your external key store proxy is not located in your Amazon VPC, create an Amazon EC2 instance in your Amazon VPC, verify that the Amazon VPC can communicate with your external key store proxy.

Connecting the VPC to the external key manager

Connect the VPC to the data center that hosts your external key manager using any of the network connectivity options that Amazon VPC supports. Ensure that the Amazon EC2 instance in the VPC (or the external key store proxy, if it is in the VPC), can communicate with the data center and the external key manager.

Creating a target group

Before you create the required VPC endpoint service, create its required components, a network load balancer (NLB) and a target group. The network load balancer (NLB) distributes requests among multiple healthy targets, any of which can service the request. In this step, you create a target group with at least two hosts for your external key store proxy, and register your IP addresses with the target group.

Follow the instructions in the Configure a target group topic using the following required values. For other fields, accept the default values and provide names as requested.

Field	Value
Target type	IP addresses
Protocol	TCP
Port	443
IP address type	IPv4
VPC	Choose the VPC where you will create the VPC endpoint service for your external key store.
Health check protocol and path	Your health check protocol and path will differ with your external key store proxy configuration. Consult the documentation for your external key manager or external key store proxy. For general information about configuring health checks for your target groups, see Health checks for your target groups in the *Elastic Load Balancing User Guide for Network Load Balancers*.
Network	Other private IP address
IPv4 address	The private addresses of your external key store proxy
Ports	443

Creating a network load balancer

The network load balancer distributes the network traffic, including requests from AWS KMS to your external key store proxy, to the configured targets.

Follow the instructions in the Configure a load balancer and a listener topic to configure and add a listener and create a load balancer using the following required values. For other fields, accept the default values and provide names as requested.

Field	Value
Scheme	Internal
IP address type	IPv4
Network mapping	Choose the VPC where you will create the VPC endpoint service for your external key store.
Mapping	Choose both of the availability zones (at least two) that you configured for your VPC subnets. Verify the subnet names and private IP address.
Protocol	TCP
Port	443
Default action: Forward to	Choose the target group for your network load balancer.

Creating a VPC endpoint service

Typically, you create an endpoint to a service. However, when you create a VPC endpoint service, you are the provider, and AWS KMS creates an endpoint to your service. For an external key store, create a VPC endpoint service with the network load balancer that you created in the previous step. The VPC endpoint service must must be in the same AWS account and supported Region as your external key store.

Multiple external key stores can share an Amazon VPC, but each external key store must have its own VPC endpoint service and private DNS name.

Follow the instructions in the Create an endpoint service topic to create your VPC endpoint service with the following required values. For other fields, accept the default values and provide names as requested.

Field	Value
Load balancer type	Network
Available load balancers	Choose the network load balancer that you created in the previous step. If your new load balancer does not appear in the list, verify that its state is active. It might take a few minutes for the load balancer state to change from provisioning to active.
Acceptance required	False. Uncheck the check box. *Do not require acceptance*. AWS KMS cannot connect to the VPC endpoint service without a manual acceptance. If acceptance is required, attempts to create the external key store fail with an `XksProxyInvalidConfigurationException` exception.
Enable private DNS name	Associate a private DNS name with the service
Private DNS name	Enter a private DNS name that is unique in its AWS Region. The private DNS name must be a subdomain of a higher level public domain. For example, if the private DNS name is `myproxy-private.xks.example.com`, it must be a subdomain of a public domain such as `xks.example.com` or `example.com`. This private DNS name must match the subject common name (CN) in the TLS certificate configured on your external key store proxy. For example, if the private DNS name is `myproxy-private.xks.example.com`, the CN on the TLS certificate must be `myproxy-private.xks.example.com` or `*.xks.example.com`.

Field	Value
	If the certificate and private DNS name do not match, attempts to connect an external key store to its external key store proxy fail with a connection error code of XKS_PROXY_INVALID_TLS_CONFIGURATION . For details, see General configuration errors.
Supported IP address types	IPv4

Verifying your private DNS name domain

When you create your VPC endpoint service, its domain verification status is `pendingVerification`. Before using the VPC endpoint service to create an external key store, this status must be `verified`. To verify that you own the domain associated with your private DNS name, you must create a TXT record in a public DNS server.

For example, if the private DNS name for your VPC endpoint service is `myproxy-private.xks.example.com`, you must create a TXT record in a public domain, such as `xks.example.com` or `example.com`, whichever is public. AWS PrivateLink looks for the TXT record first on `xks.example.com` and then on `example.com`.

> **Tip**
>
> After you add a TXT record, it might take a few minutes for the **Domain verification status** value to change from `pendingVerification` to `verify`.

To begin, find the verification status of your domain using either of the following methods. Valid values are `verified`, `pendingVerification`, and `failed`.

- In the Amazon VPC console, choose **Endpoint services**, and choose your endpoint service. In the detail pane, see **Domain verification status**.
- Use the DescribeVpcEndpointServiceConfigurations operation. The `State` value is in the `ServiceConfigurations.PrivateDnsNameConfiguration.State` field.

If the verification status is not `verified`, follow the instructions in the Domain ownership verification topic to add a TXT record to your domain's DNS server and verify that the TXT record is published. Then check your verification status again.

You are not required to create an A record for the private DNS domain name. When AWS KMS creates an interface endpoint to your VPC endpoint service, AWS PrivateLink automatically creates a hosted zone with the required A record for the private domain name in the AWS KMS VPC. For external key stores with VPC endpoint service connectivity, this happens when you connect your external key store to its external key store proxy.

Authorizing AWS KMS to connect to the VPC endpoint service

You must add AWS KMS to the **Allow principals** list for your VPC endpoint service. This allows AWS KMS to create interface endpoints to your VPC endpoint service. If AWS KMS is not an allowed principal, attempts to create an external key store will fail with an `XksProxyVpcEndpointServiceNotFoundException` exception.

Follow the instructions in the Manage permissions topic in the *AWS PrivateLink Guide*. Use the following required value.

Field	Value
ARN	`cks.kms.<region>.amazonaws.com` For example, `cks.kms.us-east-1.amazonaws.com`

Next: Creating an external key store

Managing an external key store

You can manage an external key store by using the AWS KMS console or AWS KMS API. You can create an external key store, view and edit its properties, monitor its performance, and connect and disconnect it from its external key store proxy, and delete the external key store.

Topics

- Monitoring an external key store
- Connecting and disconnecting an external key store
- Deleting an external key store

Creating an external key store

You can create one or many external key stores in each AWS account and Region. Each external key store must be associated with an external key manager outside of AWS, and an external key store proxy (XKS proxy) that mediates communication between AWS KMS and your external key manager. For details, see Planning an external key store. Before you begin, confirm that you need an external key store. Most customer can use KMS keys backed by AWS KMS key material.

> **Tip**
>
> Some external key managers provide a simpler method for creating an external key store. For details, see your external key manager documentation.

Before you create your external key store, you need to assemble the prerequisites. During the creation process, you specify the properties of your external key store. Most importantly, you indicate whether your external key store in AWS KMS uses a public endpoint or a VPC endpoint service to connect to its external key store proxy. You also specify the connection details, including the URI endpoint of the proxy and the path within that proxy endpoint where AWS KMS sends API requests to the proxy.

- If you use public endpoint connectivity, make sure that AWS KMS can communicate with your proxy over the internet using an HTTPS connection. This includes configuring TLS on the external key store proxy and ensuring that any firewalls between AWS KMS and the proxy allow traffic to and from port 443 on the proxy. While creating an external key store with public endpoint connectivity, AWS KMS tests the connection by sending a status request to the external key store proxy. This test verifies that the endpoint is reachable and that your external key store proxy will accept a request signed with your external key store proxy authentication credential. If this test request fails, the operation to create the external key store fails.
- If you use VPC endpoint service connectivity, make sure that the network load balancer, private DNS name, and VPC endpoint service are configured correctly and operational. If the external key store proxy isn't in the VPC, you need to ensure that the VPC endpoint service can communicate

with the external key store proxy. (AWS KMS tests VPC endpoint service connectivity when you connect the external key store to its external key store proxy.)

Additional considerations:

- AWS KMS records Amazon CloudWatch metrics and dimensions especially for external key stores. Monitoring graphs based on some of these metrics appear in the AWS KMS console for each external key store. We strongly recommend that you use these metrics to create alarms that monitor yourexternal key store. These alarms alert you to early signs of performance and operational problems before they occur. For instructions, see Monitoring an external key store.
- External key stores are subject to resource quotas. Use of KMS keys in an external key store are subject to request quotas. Review these quotas before designing your external key store implementation.

Note

Review your configuration for circular dependencies that might prevent it from working. For example, if you create your external key store proxy using AWS resources, make sure that operating the proxy does not require the availability of a KMS key in an external key store that is accessed via that proxy.

All new external key stores are created in a disconnected state. Before you can create KMS keys your external key store, you must connect it to its external key store proxy. To change the properties of your external key store, edit your external key store settings.

Topics

Assemble the prerequisites

Before you create an external key store, you need to assemble the required components, including the external key manager that you will use to support the external key store and the external key store proxy that translates AWS KMS requests into a format that your external key manager can understand.

The following components are required for all external key stores. In addition to these components, you need to provide the components to support the external key store proxy connectivity option that you choose.

> ⓘ **Tip**
>
> Your external key manager might include some of these components, or they might be configured for you. For details, see your external key manager documentation.
> If you are creating your external key store in the AWS KMS console, you have the option to upload a JSON-based proxy configuration file that specifies the proxy URI path and proxy authentication credential. Some external key store proxies generate this file for you. For details, see the documentation for your external key store proxy or external key manager.

External key manager

Each external key store requires at least one external key manager instance. This can be a physical or virtual hardware security module (HSM), or key management software.

You can use a single key manager, but we recommend at least two related key manager instances that share cryptographic keys for redundancy. The external key store does not require exclusive use of the external key manager. However, the external key manager must have the capacity to handle the expected frequency of encryption and decryption requests from the AWS services that use KMS keys in the external key store to protect your resources. Your external key manager should be configured to handle up to 1800 requests per second and to respond within the 250 millisecond timeout for each request. We recommend that you locate the external key manager close to an AWS Region so that the network round-trip time (RTT) is 35 milliseconds or less.

If your external key store proxy allows it, you can change the external key manager that you associate with your external key store proxy, but the new external key manager must be a backup or snapshot with the same key material. If the external key that you associate with a KMS key is no longer available to your external key store proxy, AWS KMS cannot decrypt the ciphertext encrypted with the KMS key.

The external key manager must be accessible to the external key store proxy. If the GetHealthStatus response from the proxy reports that all external key manager instances are `Unavailable`, all attempts to create an external key store fail with an `XksProxyUriUnreachableException`.

External key store proxy

You must specify an external key store proxy (XKS proxy) that conforms to the design requirements in the AWS KMS External Key Store Proxy API Specification. You can develop or buy an external key store proxy, or use an external key store proxy provided by or built into your external key manager. AWS KMS recommends that your external key store proxy be configured to handle up to 1800 requests per second and respond within the 250 millisecond timeout for each request. We recommend that you locate the external key manager close to an AWS Region so that the network round-trip time (RTT) is 35 milliseconds or less.

You can use an external key store proxy for more than one external key store, but each external key store must have a unique URI endpoint and path within the external key store proxy for its requests.

If you are using VPC endpoint service connectivity, you can locate your external key store proxy in your Amazon VPC, but that is not required. You can locate your proxy outside of AWS, such as in your private data center, and use the VPC endpoint service only to communicate with the proxy.

Proxy authentication credential

To create an external key store, you must specify your external key store proxy authentication credential (`XksProxyAuthenticationCredential`).

You must establish an authentication credential (`XksProxyAuthenticationCredential`) for AWS KMS on your external key store proxy. AWS KMS authenticates to your proxy by signing its requests using the Signature Version 4 (SigV4) process with the external key store proxy authentication credential. You specify the authentication credential when you create your external key store and you can change it at any time. If your proxy rotates your credential, be sure to update the credential values for your external key store.

The proxy authentication credential has two parts. You must provide both parts for your external key store.

- Access key ID: Identifies the secret access key. You can provide this ID in plain text.
- Secret access key: The secret part of the credential. AWS KMS encrypts the secret access key in the credential before storing it.

The SigV4 credential that AWS KMS uses to sign requests to the external key store proxy are unrelated to any SigV4 credentials associated with any AWS Identity and Access Management principals in your AWS accounts. Do not reuse any IAM SigV4 credentials for your external key store proxy.

Proxy connectivity

To create an external key store, you must specify your external key store proxy connectivity option (`XksProxyConnectivity`).

AWS KMS can communicate with your external key store proxy by using a public endpoint or an Amazon Virtual Private Cloud (Amazon VPC) endpoint service. While a public endpoint is simpler to configure and maintain, it might not meet the security requirements for every installation. If you choose the Amazon VPC endpoint service connectivity option, you must create and maintain the required components, including an Amazon VPC with at least two subnets in two different Availability Zones, a VPC endpoint service with a network load balancer and target group, and a private DNS name for the VPC endpoint service.

You can change the proxy connectivity option for your external key store. However, you must ensure that the continued availability of the key material associated with the KMS keys in your external key store. Otherwise, AWS KMS cannot decrypt any ciphertext encrypted with those KMS keys.

For help deciding which proxy connectivity option is best for your external key store, see Choosing a proxy connectivity option. For help creating an configuring VPC endpoint service connectivity, see Configuring VPC endpoint service connectivity.

Proxy URI endpoint

To create an external key store, you must specify the endpoint (`XksProxyUriEndpoint`) that AWS KMS uses to send requests to the external key store proxy.

The protocol must be HTTPS. AWS KMS communicates on port 443. Do not specify the port in the proxy URI endpoint value.

- Public endpoint connectivity — Specify the publicly available endpoint for your external key store proxy. This endpoint must be reachable before you create your external key store.
- VPC endpoint service connectivity — Specify `https://` followed by the private DNS name of the VPC endpoint service.

The TLS server certificate configured on the external key store proxy must match the domain name in the external key store proxy URI endpoint and be issued by a certificate authority supported for external key stores. For a list, see Trusted Certificate Authorities. Your certificate authority will require proof of domain ownership before issuing the TLS certificate.

The subject common name (CN) on the TLS certificate must match the private DNS name. For example, if the private DNS name is `myproxy-private.xks.example.com`, the CN on the TLS certificate must be `myproxy-private.xks.example.com` or `*.xks.example.com`.

You can change your proxy URI endpoint, but be sure that the external key store proxy has access to the key material associated with the KMS keys in your external key store. Otherwise, AWS KMS cannot decrypt any ciphertext encrypted with those KMS keys.

Uniqueness requirements

- The combined proxy URI endpoint (`XksProxyUriEndpoint`) and proxy URI path (`XksProxyUriPath`) value must be unique in the AWS account and Region.
- External key stores with public endpoint connectivity can share the same proxy URI endpoint, provided that they have different proxy URI path values.
- An external key store with public endpoint connectivity cannot use the same proxy URI endpoint value as any external key store with VPC endpoint services connectivity in the same AWS Region, even if the key stores are in different AWS accounts.
- Each external key store with VPC endpoint connectivity must have its own private DNS name. The proxy URI endpoint (private DNS name) must be unique in the AWS account and Region.

Proxy URI path

To create an external key store, you must specify the base path in your external key store proxy to the required proxy APIs. The value must start with / and must end with /kms/xks/v1 where `v1` represents the version of the AWS KMS API for the external key store proxy. This path can include an optional prefix between the required elements such as `/example-prefix/kms/xks/v1`. To find this value, see the documentation for your external key store proxy.

AWS KMS sends proxy requests to the address specified by the concatenation of the proxy URI endpoint and proxy URI path. For example, if the proxy URI endpoint is `https://myproxy.xks.example.com` and the proxy URI path is `/kms/xks/v1`, AWS KMS sends its proxy API requests to `https://myproxy.xks.example.com/kms/xks/v1`.

You can change your proxy URI path, but be sure that the external key store proxy has access to the key material associated with the KMS keys in your external key store. Otherwise, AWS KMS cannot decrypt any ciphertext encrypted with those KMS keys.

Uniqueness requirements

- The combined proxy URI endpoint (`XksProxyUriEndpoint`) and proxy URI path (`XksProxyUriPath`) value must be unique in the AWS account and Region.

VPC endpoint service

Specifies the name of the Amazon VPC endpoint service that is used to communicate with your external key store proxy. This component is required only for external key stores that use VPC endpoint service connectivity. For help setting up and configuring your VPC endpoint service for an external key store, see Configuring VPC endpoint service connectivity.

The VPC endpoint service must have the following properties:

- The VPC endpoint service must be in the same AWS account and Region as the external key store.
- It must have a network load balancer (NLB) connected to at least two subnets, each in a different Availability Zone.
- The *allow principals list* for the VPC endpoint service must include the AWS KMS service principal for the Region: `cks.kms.<region>.amazonaws.com`, such as `cks.kms.us-east-1.amazonaws.com`.
- It must not require acceptance of connection requests.
- It must have a private DNS name within a higher level public domain. For example, you could have a private DNS name of myproxy-private.xks.example.com in the public `xks.example.com` domain.

  The private DNS name for an external key store with VPC endpoint service connectivity must be unique in its AWS Region.
- The domain verification status of the private DNS name domain must be `verified`.
- The TLS server certificate configured on the external key store proxy must specify the private DNS hostname at which the endpoint is reachable.

Uniqueness requirements

- External key stores with VPC endpoint connectivity can share an Amazon VPC, but each external key store must have its own VPC endpoint service and private DNS name.

Proxy configuration file

A *proxy configuration file* is an optional JSON-based file that contains values for the proxy URI path and proxy authentication credential properties of your external key store. When creating or editing an external key store in the AWS KMS console, you can upload a proxy configuration file to supply configuration values for your external key store. Using this file avoids typing and pasting errors, and ensures that the values in your external key store match the values in your external key store proxy.

Proxy configuration files are generated by the external key store proxy. To learn whether your external key store proxy offers a proxy configuration file, see your external key store proxy documentation.

The following is an example of a well-formed proxy configuration file with fictitious values.

```
{
  "XksProxyUriPath": "/example-prefix/kms/xks/v1",
  "XksProxyAuthenticationCredential": {
    "AccessKeyId": "ABCDE12345670EXAMPLE",
    "RawSecretAccessKey": "0000EXAMPLEFA5FT0mCc3DrGUe2sti527BitkQ0Zr9MO9+vE="
  }
}
```

You can upload a proxy configuration file only when creating or editing an external key store in the AWS KMS console. You cannot use it with the CreateCustomKeyStore or UpdateCustomKeyStore operations, but you can use the values in the proxy configuration file to ensure that your parameter values are correct.

Create an external key store (console)

Before creating an external key store, review the Planning an external key store, choose your proxy connectivity type, and ensure that you have created and configured all of the required components. If you need help finding any of the required values, consult the documentation for your external key store proxy or key management software.

Note

When you create an external key store in the AWS Management Console, you can upload a JSON-based *proxy configuration file* with values for the proxy URI path and proxy authentication credential. Some proxies generate this file for you. It is not required.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Custom key stores**, **External key stores**.
4. Choose **Create external key store**.
5. Enter a friendly name for the external key store. The name must be unique among all external key stores in your account.

   Important

   Do not include confidential or sensitive information in this field. This field may be displayed in plaintext in CloudTrail logs and other output.
6. Choose your proxy connectivity type.

   Your proxy connectivity choice determines the components required for your external key store proxy. For help making this choice, see Choosing a proxy connectivity option.
7. Choose or enter the name of the VPC endpoint service for this external key store. This step appears only when your external key store proxy connectivity type is **VPC endpoint service**.

   The VPC endpoint service and its VPCs must fulfill the requirements for an external key store. For details, see the section called "Assemble the prerequisites".
8. Enter your proxy URI endpoint. The protocol must be HTTPS. AWS KMS communicates on port 443. Do not specify the port in the proxy URI endpoint value.

   If AWS KMS recognizes the VPC endpoint service that you specified in the previous step, it completes this field for you.

   For public endpoint connectivity, enter a publicly available endpoint URI. For VPC endpoint connectivity, enter `https://` followed by the private DNS name of the VPC endpoint service.

9. To enter the values for the proxy URI path prefix and proxy authentication credential, upload a proxy configuration file, or enter the values manually.

    - If you have an optional proxy configuration file that contains values for your proxy URI path and proxy authentication credential, choose **Upload configuration file**. Follow the steps to upload the file.

      When the file is uploaded, the console displays the values from the file in editable fields. You can change the values now or edit these values after the external key store is created.

      To display the value of the secret access key, choose **Show secret access key**.
    - If you don't have a proxy configuration file, you can enter the proxy URI path and proxy authentication credential values manually.

        a. If you don't have a proxy configuration file, you can enter your proxy URI manually. The console supplies the required **/kms/xks/v1** value.

        If your proxy URI path includes an optional prefix, such as the `example-prefix` in `/`*`example-prefix`*`/kms/xks/v1`, enter the prefix in the **Proxy URI path prefix** field. Otherwise, leave the field empty.

        b. If you don't have a proxy configuration file, you can enter your proxy authentication credential manually. Both the access key ID and secret access key are required.

        - In **Proxy credential: Access key ID**, enter the access key ID of the proxy authentication credential. The access key ID identifies the secret access key.
        - In **Proxy credential: Secret access key**, enter the secret access key of the proxy authentication credential.

        To display the value of the secret access key, choose **Show secret access key**.

        This procedure does not set or change the authentication credential you established on your external key store proxy. It just associates these values with your external key store. For information about setting, changing, and your rotating proxy authentication credential, see the documentation for your external key store proxy or key management software.

        If your proxy authentication credential changes, edit the credential setting for your external key store.

10. Choose **Create external key store**.

When the procedure is successful, the new external key store appears in the list of external key stores in the account and Region. If it is unsuccessful, an error message appears that describes the problem and provides help on how to fix it. If you need more help, see CreateKey errors for the external key.

Next: New external key stores are not automatically connected. Before you can create AWS KMS keys in your external key store, you must connect the external key store to its external key store proxy.

Create an external key store (API)

You can use the CreateCustomKeyStore operation to create a new external key store. For help finding the values for the required parameters, see the documentation for your external key store proxy or key management software.

> **Tip**
>
> You cannot upload a proxy configuration file when using the `CreateCustomKeyStore` operation. However, you can use the values in the proxy configuration file to ensure that your parameter values are correct.

To create an external key store, the `CreateCustomKeyStore` operation requires the following parameter values.

- `CustomKeyStoreName` – A friendly name for the external key store that is unique in the account.

  > **Important**
  >
  > Do not include confidential or sensitive information in this field. This field may be displayed in plaintext in CloudTrail logs and other output.

- `CustomKeyStoreType` — Specify `EXTERNAL_KEY_STORE`.
- `XksProxyConnectivity` – Specify `PUBLIC_ENDPOINT` or `VPC_ENDPOINT_SERVICE`.

- XksProxyAuthenticationCredential — Specify both the access key ID and the secret access key.
- XksProxyUriEndpoint — The endpoint that AWS KMS uses to communicate with your external key store proxy.
- XksProxyUriPath — The path within the proxy to the proxy APIs.
- XksProxyVpcEndpointServiceName — Required only when your XksProxyConnectivity value is VPC_ENDPOINT_SERVICE.

Note

If you use AWS CLI version 1.0, run the following command before specifying a parameter with an HTTP or HTTPS value, such as the XksProxyUriEndpoint parameter.

```
aws configure set cli_follow_urlparam false
```

Otherwise, AWS CLI version 1.0 replaces the parameter value with the content found at that URI address, causing the following error:

```
Error parsing parameter '--xks-proxy-uri-endpoint': Unable to retrieve
https:// : received non 200 status code of 404
```

The following examples use fictitious values. Before running the command, replace them with valid values for your external key store.

Create an external key store with public endpoint connectivity.

```
$ aws kms create-custom-key-store
        --custom-key-store-name ExampleExternalKeyStorePublic \
        --custom-key-store-type EXTERNAL_KEY_STORE \
        --xks-proxy-connectivity PUBLIC_ENDPOINT \
        --xks-proxy-uri-endpoint https://myproxy.xks.example.com \
        --xks-proxy-uri-path /kms/xks/v1 \
        --xks-proxy-authentication-credential
 AccessKeyId=<value>,RawSecretAccessKey=<value>
```

Create an external key store with VPC endpoint service connectivity.

```
$ aws kms create-custom-key-store
        --custom-key-store-name ExampleExternalKeyStoreVPC \
        --custom-key-store-type EXTERNAL_KEY_STORE \
        --xks-proxy-connectivity VPC_ENDPOINT_SERVICE \
        --xks-proxy-vpc-endpoint-service-name com.amazonaws.vpce.us-east-1.vpce-svc-
example \
        --xks-proxy-uri-endpoint https://myproxy-private.xks.example.com \
        --xks-proxy-uri-path /kms/xks/v1 \
        --xks-proxy-authentication-credential
 AccessKeyId=<value>,RawSecretAccessKey=<value>
```

When the operation is successful, `CreateCustomKeyStore` returns the custom key store ID, as shown in the following example response.

```
{
    "CustomKeyStoreId": cks-1234567890abcdef0
}
```

If the operation fails, correct the error indicated by the exception, and try again. For additional help, see Troubleshooting external key stores.

Next: To use the external key store, connect it to its external key store proxy.

Editing external key store properties

You can edit selected properties of an existing external key store.

You can edit some properties while the external key store is connected or disconnected. For other properties, you must first disconnect your external key store from its external key store proxy. The connection state of the external key store must be `DISCONNECTED`. While your external key store is disconnected, you can manage the key store and its KMS keys, but you cannot create or use KMS keys in the external key store. To find the connection state of your external key store, use the DescribeCustomKeyStores operation or see the **General configuration** section on the detail page for the external key store.

Before updating the properties your external key store, AWS KMS sends a GetHealthStatus request to the external key store proxy using the new values. If the request succeeds, it indicates that you can connect and authenticate to an external key store proxy with the updated property values. If the request fails, the edit operation fails with an exception that identifies the error.

When the edit operation completes, the updated property values for your external key store appear in the AWS KMS console and the `DescribeCustomKeyStores` response. However, it can take up to five minutes for the changes to be fully effective.

If you edit your external key store in the AWS KMS console, you have the option to upload a JSON-based proxy configuration file that specifies the proxy URI path and proxy authentication credential. Some external key store proxies generate this file for you. For details, see the documentation for your external key store proxy or external key manager.

> **Warning**
>
> The updated property values must connect your external key store to a proxy for the same external key manager as the previous values, or for a backup or snapshot of the external key manager with the same cryptographic keys. If your external key store permanently loses its access to the external keys associated with its KMS keys, ciphertext encrypted under those external keys is unrecoverable. In particular, changing the proxy connectivity of an external key store can prevent AWS KMS from accessing your external keys.

> **Tip**
>
> Some external key managers provide a simpler method for editing external key store properties. For details, see your external key manager documentation.

You can change the following properties of an external key store.

Editable external key store properties	Any connection state	Require Disconnected state
Custom key store name A required friendly name for a custom key store. **Important** Do not include confidential or sensitive information in this field. This field may be	✓	

Editable external key store properties	Any connection state	Require Disconnected state
displayed in plaintext in CloudTrail logs and other output.		
Proxy authentication credential (XksProxyAuthenticationCredential) (You must specify both the access key ID and the secret access key, even if you are changing only one element.)	✓	
Proxy URI path (XksProxyUriPath)	✓	
Proxy connectivity (XksProxyConnectivity) (You must also update the proxy URI endpoint. If you are changing to VPC endpoint service connectivity, you must specify a proxy VPC endpoint service name.)		✓
Proxy URI endpoint (XksProxyUriEndpoint) If you change the proxy endpoint URI, you might also need to change the associated TLS certificate.		✓
Proxy VPC endpoint service name (XksProxyVpcEndpointServiceName) (This field is required for VPC endpoint service connectivity)		✓

Topics

- Edit an external key store (console)
- Edit an external key store (API)

Edit an external key store (console)

When you edit an key store, you can change any or of the editable values. Some changes require that the external key store be disconnected from its external key store proxy.

If you are editing the proxy URI path or proxy authentication credential, you can enter the new values or upload an external key store proxy configuration file that includes the new values.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Custom key stores**, **External key stores**.
4. Choose the row of the external key store you want to edit.
5. If necessary, disconnect the external key store from its external key store proxy. From the **Key store actions** menu, choose **Disconnect**.
6. From the **Key store actions** menu, choose **Edit**.
7. Change one or more of the editable external key store properties. You can also upload an external key store proxy configuration file with values for the proxy URI path and proxy authentication credential.You can use a proxy configuration file even if some values specified in the file haven't changed.
8. Choose **Update external key store**.
9. Review the warning, and if you decide to continue, confirm the warning, and then choose **Update external key store**.

   When the procedure is successful, a message describes the properties that you edited. When it is unsuccessful, an error message appears that describes the problem and provides help on how to fix it.
10. If necessary, reconnect the external key store. From the **Key store actions** menu, choose **Connect**.

    You can leave the external key store disconnected. But while it is disconnected, you cannot create KMS keys in the external key store or use the KMS keys in the external key store in cryptographic operations.

Edit an external key store (API)

To change the properties of an external key store, use the UpdateCustomKeyStore operation. You can change multiple properties of an external key store in the same operation. If the operation is successful, AWS KMS returns an HTTP 200 response and a JSON object with no properties.

Use the `CustomKeyStoreId` parameter to identify the external key store. Use the other parameters to change the properties. You cannot use a proxy configuration file with the `UpdateCustomKeyStore` operation. The proxy configuration file is supported only by the AWS KMS console. However, you can use the proxy configuration file to help you determine the correct parameter values for your external key store proxy.

The examples in this section use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

Before you begin, if necessary, disconnect the external key store from its external key store proxy. After updating, if necessary, you can reconnect the external key store to its external key store proxy. You can leave the external key store in the disconnected state, but you must reconnect it before you can create new KMS keys in the key store or use existing KMS keys in the key store for cryptographic operations.

Note

If you use AWS CLI version 1.0, run the following command before specifying a parameter with an HTTP or HTTPS value, such as the `XksProxyUriEndpoint` parameter.

```
aws configure set cli_follow_urlparam false
```

Otherwise, AWS CLI version 1.0 replaces the parameter value with the content found at that URI address, causing the following error:

```
Error parsing parameter '--xks-proxy-uri-endpoint': Unable to retrieve
https:// : received non 200 status code of 404
```

Change the name of the external key store

The first example uses the UpdateCustomKeyStore operation to change the friendly name of the external key store to `XksKeyStore`. The command uses the `CustomKeyStoreId` parameter to identify the custom key store and the CustomKeyStoreName to specify the new name for the custom key store. Replace all example values with actual values for your external key store.

```
$ aws kms update-custom-key-store --custom-key-store-id cks-1234567890abcdef0 --new-custom-key-store-name XksKeyStore
```

Change the proxy authentication credential

The following example updates the proxy authentication credential that AWS KMS uses to authenticate to the external key store proxy. You can use a command like this one to update the credential if it is rotated on your proxy.

Update the credential on your external key store proxy first. Then use this feature to report the change to AWS KMS. (Your proxy will briefly support both the old and new credential so you have time to update your credential in AWS KMS.)

You must always specify both the access key ID and the secret access key in the credential, even if only one value is changed.

The first two commands set variables to hold the credential values. The UpdateCustomKeyStore operations uses the CustomKeyStoreId parameter to identify the external key store. It uses the XksProxyAuthenticationCredential parameter with its AccessKeyId and RawSecretAccessKey fields to specify the new credential. Replace all example values with actual values for your external key store.

```
$ accessKeyID=access key id
$ secretAccessKey=secret access key

$ aws kms update-custom-key-store --custom-key-store-id cks-1234567890abcdef0 \
        --xks-proxy-authentication-credential \
            AccessKeyId=$accessKeyId,RawSecretAccessKey=$secretAccessKey
```

Change the proxy URI path

The following example updates the proxy URI path (XksProxyUriPath). The combination of the proxy URI endpoint and the proxy URI path must be unique in the AWS account and Region. Replace all example values with actual values for your external key store.

```
$ aws kms update-custom-key-store --custom-key-store-id cks-1234567890abcdef0 \
            --xks-proxy-uri-path /kms/xks/v1
```

Change to VPC endpoint service connectivity

The following example uses the UpdateCustomKeyStore operation to change the external key store proxy connectivity type to VPC_ENDPOINT_SERVICE. To make this change, you must specify the required values for VPC endpoint service connectivity, including the VPC endpoint service name (XksProxyVpcEndpointServiceName) and a proxy URI endpoint (XksProxyUriEndpoint) value that includes the private DNS name for the VPC endpoint service. Replace all example values with actual values for your external key store.

```
$ aws kms update-custom-key-store --custom-key-store-id cks-1234567890abcdef0 \
            --xks-proxy-connectivity "VPC_ENDPOINT_SERVICE" \
            --xks-proxy-uri-endpoint https://myproxy-private.xks.example.com \
            --xks-proxy-vpc-endpoint-service-name com.amazonaws.vpce.us-east-1.vpce-
svc-example
```

Change to public endpoint connectivity

The following example changes the external key store proxy connectivity type to PUBLIC_ENDPOINT. When you make this change, you must update the proxy URI endpoint (XksProxyUriEndpoint) value. Replace all example values with actual values for your external key store.

> **Note**
>
> VPC endpoint connectivity provides greater security than public endpoint connectivity. Before changing to public endpoint connectivity, consider other options, including locating your external key store proxy on premises and using the VPC only for communication.

```
$ aws kms update-custom-key-store --custom-key-store-id cks-1234567890abcdef0 \
            --xks-proxy-connectivity "PUBLIC_ENDPOINT" \
            --xks-proxy-uri-endpoint https://myproxy.xks.example.com
```

Viewing an external key store

You can view external key stores in each account and Region by using the AWS KMS console or by using the DescribeCustomKeyStores operation.

When you view an external key store, you can see the following:

- Basic information about the key store, including its friendly name, ID, key store type, and creation date.
- Configuration information for the external key store proxy, including the connectivity type, proxy URI endpoint and path, and the access key ID of your current proxy authentication credential.
- If the external key store proxy uses VPC endpoint service connectivity, the console displays the name of the VPC endpoint service.
- The current connection state.

> **Note**
>
> A connection state value of **Disconnected** indicates that the external key store has never been connected, or it was intentionally disconnected from its external key store proxy. However, if your attempts to use a KMS key in a connected external key store fail, that might indicate a problem with the external key store or its proxy. For help, see External key store connection errors.

- A Monitoring section with graphs of Amazon CloudWatch metrics designed to help you detect and resolve issues with your external key store. For help interpreting the graphs, using them in your planning and troubleshooting, and creating CloudWatch alarms based on the metrics in the graphs, see Monitoring an external key store.

See also:

- Viewing KMS keys in an external key store
- Logging AWS KMS API calls with AWS CloudTrail

Topics

External key store properties

The following properties of an external key store are visible in the AWS KMS console and the DescribeCustomKeyStores response.

Custom key store properties

The following values appear in the **General configuration** section of the detail page for each custom key store.These properties apply to all custom key stores, including AWS CloudHSM key stores and external key stores.

Custom key store ID

A unique ID that AWS KMS assigns to the custom key store.

Custom key store name

A friendly name that you assign to the custom key store when you create it. You can change this value at any time.

Custom key store type

The type of custom key store. Valid values are AWS CloudHSM (`AWS_CLOUDHSM`) or External key store (`EXTERNAL_KEY_STORE`). You cannot change the type after you create the custom key store.

Creation date

The date that the custom key store was created. This date is displayed in local time for the AWS Region.

Connection state

Indicates whether the custom key store is connected to its backing key store. The connection state is `DISCONNECTED` only if the custom key store has never been connected to its backing key store, or it has been intentionally disconnected. For details, see the section called "Connection state".

External key store configuration properties

The following values appear in the **External key store proxy configuration** section of the detail page for each external key store and in the `XksProxyConfiguration` element of the DescribeCustomKeyStores response. For a detailed description of each field, including uniqueness requirements and help with determining the correct value for each field, see the section called "Assemble the prerequisites" in the *Creating an external key store* topic.

Proxy connectivity

Indicates whether the external key store uses public endpoint connectivity or VPC endpoint service connectivity.

Proxy URI endpoint

The endpoint that AWS KMS uses to connect to your external key store proxy.

Proxy URI path

The path from the proxy URI endpoint where AWS KMS sends proxy API requests.

Proxy credential: Access key ID

Part of the proxy authentication credential that you establish on your external key store proxy. The access key ID identifies the secret access key in the credential.

AWS KMS uses the SigV4 signing process and the proxy authentication credential to sign its requests to your external key store proxy. The credential in the signature allows the external key store proxy to authenticate requests on your behalf from AWS KMS.

VPC endpoint service name

The name of the Amazon VPC endpoint service that supports your external key store. This value appears only when the external key store uses VPC endpoint service connectivity. You can locate your external key store proxy in the VPC or use the VPC endpoint service to communicate securely with your external key store proxy.

View an external key store (console)

To view the external key stores in a given account and Region, use the following procedure.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Custom key stores**, **External key stores**.
4. To view detailed information about an external key store, choose the key store name.

View an external key store (API)

To view your external key stores, use the DescribeCustomKeyStores operation. By default, this operation returns all custom key stores in the account and Region. But you can use either the `CustomKeyStoreId` or `CustomKeyStoreName` parameter (but not both) to limit the output to a particular custom key store.

For custom key stores, the output consists of the custom key store ID, name, and type, and the connection state of the key store. If the connection state is `FAILED`, the output also includes a `ConnectionErrorCode` that describes the reason for the error. For help interpreting the `ConnectionErrorCode` for an external key store, see Connection error codes for external key stores.

For external key stores, the output also includes the `XksProxyConfiguration` element. This element includes the connectivity type, proxy URI endpoint, proxy URI path, and the access key ID of the proxy authentication credential.

The examples in this section use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

For example, the following command returns all custom key stores in the account and Region. You can use the `Limit` and `Marker` parameters to page through the custom key stores in the output.

```
$ aws kms describe-custom-key-stores
```

The following command uses the `CustomKeyStoreName` parameter to get only the example external key store with the `ExampleXksPublic` friendly name. This example key store uses public endpoint connectivity. It is connected to its external key store proxy.

```
$ aws kms describe-custom-key-stores --custom-key-store-name ExampleXksPublic
{
    "CustomKeyStores": [
    {
      "CustomKeyStoreId": "cks-1234567890abcdef0",
      "CustomKeyStoreName": "ExampleXksPublic",
      "ConnectionState": "CONNECTED",
      "CreationDate": "2022-12-14T20:17:36.419000+00:00",
      "CustomKeyStoreType": "EXTERNAL_KEY_STORE",
      "XksProxyConfiguration": {
        "AccessKeyId": "ABCDE12345670EXAMPLE",
    "Connectivity": "PUBLIC_ENDPOINT",
    "UriEndpoint": "https://xks.example.com:6443",
    "UriPath": "/example/prefix/kms/xks/v1"
   }
  }
 ]
}
```

The following command gets an example external key store with VPC endpoint service connectivity. In this example, the external key store is connected to its external key store proxy.

```
$ aws kms describe-custom-key-stores --custom-key-store-name ExampleXksVpc
{
  "CustomKeyStores": [
  {
   "CustomKeyStoreId": "cks-9876543210fedcba9",
   "CustomKeyStoreName": "ExampleXksVpc",
   "ConnectionState": "CONNECTED",
   "CreationDate": "2022-12-13T18:34:10.675000+00:00",
   "CustomKeyStoreType": "EXTERNAL_KEY_STORE",
   "XksProxyConfiguration": {
    "AccessKeyId": "ABCDE98765432EXAMPLE",
    "Connectivity": "VPC_ENDPOINT_SERVICE",
    "UriEndpoint": "https://example-proxy-uri-endpoint-vpc",
    "UriPath": "/example/prefix/kms/xks/v1",
    "VpcEndpointServiceName": "com.amazonaws.vpce.us-east-1.vpce-svc-example"
   }
  }
 ]
}
```

A ConnectionState of Disconnected indicates that an external key store has never been connected or it was intentionally disconnected from its external key store proxy. However, if attempts to use a KMS key in a connected external key store fail, that might indicate a problem with the external key store proxy or other external components.

If the ConnectionState of the external key store is FAILED, the DescribeCustomKeyStores response includes a ConnectionErrorCode element that explains the reason for the error.

For example, in the following output, the XKS_PROXY_TIMED_OUT value indicates AWS KMS can connect to the external key store proxy, but the connection failed because the external key store proxy did not respond to AWS KMS in the time allotted. If you see this connection error code repeatedly, notify your external key store proxy vendor. For help with this and other connection error failures, see Troubleshooting external key stores.

```
$ aws kms describe-custom-key-stores --custom-key-store-name ExampleXksVpc
{
  "CustomKeyStores": [
  {
   "CustomKeyStoreId": "cks-9876543210fedcba9",
   "CustomKeyStoreName": "ExampleXksVpc",
   "ConnectionState": "FAILED",
   "ConnectionErrorCode": "XKS_PROXY_TIMED_OUT",
   "CreationDate": "2022-12-13T18:34:10.675000+00:00",
   "CustomKeyStoreType": "EXTERNAL_KEY_STORE",
   "XksProxyConfiguration": {
    "AccessKeyId": "ABCDE98765432EXAMPLE",
    "Connectivity": "VPC_ENDPOINT_SERVICE",
    "UriEndpoint": "https://example-proxy-uri-endpoint-vpc",
    "UriPath": "/example/prefix/kms/xks/v1",
    "VpcEndpointServiceName": "com.amazonaws.vpce.us-east-1.vpce-svc-example"
   }
  }
 ]
}
```

Monitoring an external key store

AWS KMS collects metrics for each interaction with an external key store and publishes them in your CloudWatch account. These metrics are used to generate the graphs in the monitoring section of the detail page for each external key store. The following topic details how to use the graphs to identify and troubleshoot operational and configuration issues impacting your external key store. We recommend using the CloudWatch metrics to set alarms that notify you when your external key store isn't performing as expected. For more information, see Monitoring with Amazon CloudWatch.

Topics

- Viewing the graphs
- Interpreting the graphs
- Setting alarms

Viewing the graphs

You can view the graphs at different levels of detail. By default, each graph uses a three hour time range and five minute aggregation period. You can adjust the graph view within the console, but your changes will revert to the default settings when the external key store detail page is closed or the browser is refreshed. For help with Amazon CloudWatch terminology, see Amazon CloudWatch concepts.

View data point details

The data in each graph is collected by AWS KMS metrics. To view more information about a specific data point, pause the mouse over the data point on the line graph. This will display a pop-up with more information about the metric that the graph was derived from. Each list item displays the dimension value recorded at that data point. The pop-up displays a null value (**–**) if there is no metric data available for the dimension value at that data point. Some graphs record multiple dimensions and values for a single data point. Other graphs, like the reliability graph, use the data collected by the metric to calculate a unique value. Each list item is associated with a different line graph color.

Modify the time range

To modify the time range, select one of the predefined time ranges in the upper right corner of the monitoring section. The predefined time ranges span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). This adjusts the time range for all graphs. If you want to view one specific graph in a different time range, or if you want to set a custom time range, enlarge the graph or view it in the Amazon CloudWatch console.

Zoom in on graphs

You can use the mini-map zoom feature to focus on sections of line graphs and stacked portions of the graphs without changing between zoomed-in and zoomed-out views. For example, you can use the mini-map zoom feature to focus on a peak in a graph, so that you can compare the spike against other graphs in the monitoring section from the same timeline.

1. Choose and drag on the area of the graph that you want to focus on, and then release the drag.
2. To reset the zoom, choose the **Reset zoom** icon, which looks like a magnifying glass with a minus (-) symbol inside.

Enlarge a graph

To enlarge a graph, select the menu icon in the upper right corner of an individual graph and choose **Enlarge**. You can also select the enlarge icon that appears next to the menu icon when you hover over a graph.

Enlarging a graph enables you to further modify the view of a graph by specifying a different period, custom time range, or refresh interval. These changes will revert to the default settings when you close the enlarged view.

Modify the period

1. Choose the **Period options** menu. By default, this menu displays the value: **5 minutes**.
2. Choose a period, the predefined periods span from 1 second to 30 days.

   For example, you can choose a one-minute view, which can be useful when troubleshooting. Or, choose a less detailed, one-hour view. That can be useful when viewing a broader time range (for example, 3 days) so that you can see trends over time. For more information, see Periods in the *Amazon CloudWatch User Guide*.

Modify the time range or time zone

1. Select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **Custom** to set your own time range.
2. Choose **Custom**

   a. *Time range:* select the **Absolute** tab in the upper left corner of the box. Use the calendar picker or text field boxes to specify a time range.

   b. *Time zone:* choose the dropdown in the upper right corner of the box. You can change the time zone to **UTC** or **Local time zone**.
3. After you specify a time range, choose **Apply**.

Modify how often the data in your graph is refreshed

1. Choose the **Refresh options** menu in the upper-right corner.
2. Choose a refresh interval (**Off**, **10 Seconds**, **1 Minute**, **2 Minutes**, **5 Minutes**, or **15 Minutes**).

View graphs in the Amazon CloudWatch console

The graphs in the monitoring section are derived from predefined metrics that AWS KMS publishes to Amazon CloudWatch. You can open them within the CloudWatch console and save them to CloudWatch dashboards. If you have multiple external key stores, you can open their respective graphs in CloudWatch and save them to a single dashboard to compare their health and usage.

Add to CloudWatch dashboard

Select **Add to dashboard** in the upper right corner to add all of the graphs to an Amazon CloudWatch dashboard. You can either select an existing dashboard or create a new one. For information on using this dashboard to create customized views of the graphs and alarms, see Using Amazon CloudWatch dashboards in the *Amazon CloudWatch User Guide*.

View in CloudWatch metrics

Select the menu icon in the upper right corner of an individual graph and choose **View in metrics** to view this graph in the Amazon CloudWatch console. From the CloudWatch console, you can add this single graph to a dashboard and modify time ranges, periods, and refresh intervals. For more information see, Graphing metrics in the *Amazon CloudWatch User Guide*.

Interpreting the graphs

AWS KMS provides several graphs to monitor the health of your external key store within the AWS KMS console. These graphs are automatically configured and derived from AWS KMS metrics.

The graph data is collected as part of the calls you make to your external key store and external keys. You might see data populating graphs during a time range that you did not make any calls, this data comes from the periodic `GetHealthStatus` calls that AWS KMS makes on your behalf to check the status of your external key store proxy and external key manager. If your graphs display the message **No data available**, then there were no calls recorded during that time range or your external key store is in a `DISCONNECTED` state. You might be able to identify the time your external key store disconnected by adjusting your view to a broader time range.

Topics

Total requests

The total number of AWS KMS requests being received for a specific external key store during a given time range. Use this graph to determine if you are at risk of throttling.

AWS KMS recommends that your external key manager be able to handle up to 1800 requests for cryptographic operations per second. If you approach 540,000 calls in a five-minute period, you are at risk of throttling.

You can monitor the number of requests for cryptographic operations on KMS keys in your external key store that AWS KMS throttles with the ExternalKeyStoreThrottle metric.

If you are getting very frequent `KMSInvalidStateException` errors with a message that explains that the request was rejected "due to a very high request rate," it might indicate that your external key manager or external key store proxy cannot keep pace with the current request rate. If possible, lower your request rate. You might also consider requesting a decrease in your custom key store request quota value. Decreasing this quota value might increase throttling, but it indicates that AWS KMS is rejecting excess requests quickly before they are sent to your external key store proxy or external key manager. To request a quota decrease, please visit AWS Support Center and create a case.

The total requests graph is derived from the XksProxyErrors metric, which collects data on both the successful and unsuccessful responses that AWS KMS receives from your external key store proxy. When you view a specific data point, the pop-up displays the value of the `CustomKeyStoreId` dimension alongside the total number of AWS KMS requests recorded at that data point. The `CustomKeyStoreId` will always be the same.

Reliability

The percentage of AWS KMS requests for which the external key store proxy returned either a successful response or a non-retryable error. Use this graph to evaluate the operational health of your external key store proxy.

When the graph displays a value less than 100%, it indicates cases where the proxy either did not respond or responded with a retryable error. This can indicate problems with the network, slowness of the external key store proxy or external key manager, or implementation bugs.

If the request includes a bad credential and your proxy responds with an `AuthenticationFailedException`, the graph will still indicate 100% reliability because the proxy identified an incorrect value in the external key store proxy API request, and therefore the failure is expected. If the percentage of your reliability graph is 100%, then your external key store proxy is responding as expected. If the graph displays a value less than 100%, then the proxy either responded with a retryable error or timed out. For example, if the proxy responds with a `ThrottlingException` due to a very high request rate, it will display a lower reliability percentage because the proxy was unable to identify a specific problem in the request that caused it to fail. This is because retryable errors are likely transient problems that can be resolved by retrying the request.

The following error responses will lower the reliability percentage. You can use the Top 5 exceptions graph and the XksProxyErrors metric to further monitor how frequently your proxy returns each retryable error.

- `InternalException`
- `DependencyTimeoutException`
- `ThrottlingException`
- `XksProxyUnreachableException`

The reliability graph is derived from the XksProxyErrors metric, which collects data on both the successful and unsuccessful responses that AWS KMS receives from your external key store proxy. The reliability percentage will only lower if the response has an `ErrorType` value of `Retryable`. When you view a specific data point, the pop-up displays the value of the `CustomKeyStoreId` dimension alongside the reliability percentage for AWS KMS requests recorded at that data point. The `CustomKeyStoreId` will always be the same.

We recommend using the XksProxyErrors metric to create a CloudWatch alarm that notifies you of potential networking problems by alerting you when more than five retryable errors are recorded in a one minute period. For more information, see Creating an Amazon CloudWatch alarm for retryable errors.

Latency

The number of milliseconds it takes for an external key store proxy to respond to an AWS KMS request. Use this graph to evaluate the performance of your external key store proxy and external key manager.

AWS KMS expects the external key store proxy to respond to each request within 250 milliseconds. In the case of network timeouts, AWS KMS will retry the request once. If the proxy fails a second time, the recorded latency is the combined timeout limit for both request attempts and the graph will display approximately 500 milliseconds. In all other cases where the proxy doesn't respond within the 250 millisecond timeout limit, the recorded latency is 250 milliseconds. If the proxy is frequently timing out on encryption and decryption operations, consult your external proxy administrator. For help troubleshooting latency problems, see Latency and timeout errors.

Slow responses might also indicate that your external key manager cannot handle the current request traffic. AWS KMS recommends that your external key manager be able to handle up to 1800 requests for cryptographic operations per second. If your external key manager cannot handle the 1800 requests per second rate, consider requesting a decrease in your request quota for KMS keys in a custom key store. Requests for cryptographic operations using the KMS keys in your external key store will fail fast with a throttling exception, rather than being processed and later rejected by your external key store proxy or external key manager.

The latency graph is derived from the XksProxyLatency metric. When you view a specific data point, the pop-up displays the corresponding `KmsOperation` and `XksOperation` dimension values alongside the average latency recorded for the operations at that data point. The list items are ordered from highest latency to lowest.

We recommend using the XksProxyLatency metric to create a CloudWatch alarm that notifies you when your latency is approaching the timeout limit. For more information, see Creating an Amazon CloudWatch alarm for response timeout.

Top 5 exceptions

The top five exceptions for failed cryptographic and management operations during a given time range. Use this graph to track the most frequent errors, so you can prioritize your engineering effort.

This count includes exceptions that AWS KMS received from the external key store proxy and the `XksProxyUnreachableException` that AWS KMS returns internally when it cannot establish communication with the external key store proxy.

High rates of retryable errors might indicate networking errors, while high rates of non-retryable errors might indicate a problem with the configuration of your external key store. For example, a spike in `AuthenticationFailedExceptions` indicates a discrepancy between the authentication credentials configured in AWS KMS and the external key store proxy. To view your external key store configuration, see Viewing an external key store. To edit your external key store settings, see Editing external key store properties.

The exceptions that AWS KMS receives from the external key store proxy are different from the exceptions that AWS KMS returns when an operation fails. AWS KMS cryptographic operations return an KMSInvalidStateException for all failures related to the external configuration or connection state of the external key store. To identify the problem, use the accompanying error message text.

The following table shows the exceptions that can appear in the top 5 exceptions graph and the corresponding exceptions that AWS KMS returns to you.

Error type	Exception displayed in the graph	Exception that AWS KMS returned to you
Non-retryable	**AccessDeniedException** For troubleshooting help, see Proxy authorization issues.	**CustomKeyStoreInvalidStateException** in response to CreateKey operations. **KMSInvalidStateException** in response to cryptographic operations.
Non-retryable	**AuthenticationFailedException** For troubleshooting help, see Authentication credential errors.	**XksProxyIncorrectAuthenticationCredentialException** in response to CreateCustomKeyStore and UpdateCustomKeyStore operations. **CustomKeyStoreInvalidStateException** in response to CreateKey operations. **KMSInvalidStateException** in response to cryptographic operations.
Retryable	**DependencyTimeoutException**	**XksProxyUriUnreachableException** in

Error type	Exception displayed in the graph	Exception that AWS KMS returned to you
	For troubleshooting help, see Latency and timeout errors.	response to `CreateCustomKeyStore` and `UpdateCustomKeyStore` operations. **`CustomKeyStoreInvalidStateException`** in response to `CreateKey` operations. **`KMSInvalidStateException`** in response to cryptographic operations.
Retryable	**`InternalException`** The external key store proxy rejected the request because it cannot communicate with the external key manager. Verify that the external key store proxy configuration is correct and that the external key manager is available.	**`XksProxyInvalidResponseException`** in response to `CreateCustomKeyStore` and `UpdateCustomKeyStore` operations. **`CustomKeyStoreInvalidStateException`** in response to `CreateKey` operations. **`KMSInvalidStateException`** in response to cryptographic operations.
Non-retryable	**`InvalidCiphertextException`** For troubleshooting help, see Decryption errors.	**`KMSInvalidStateException`** in response to cryptographic operations.

Error type	Exception displayed in the graph	Exception that AWS KMS returned to you
Non-retryable	**InvalidKeyUsageException** For troubleshooting help, see Cryptographic operation errors for the external key.	**XksKeyInvalidConfigurationException** in response to CreateKey operations. **KMSInvalidStateException** in response to cryptographic operations.
Non-retryable	**InvalidStateException** For troubleshooting help, see Cryptographic operation errors for the external key.	**XksKeyInvalidConfigurationException** in response to CreateKey operations. **KMSInvalidStateException** in response to cryptographic operations.
Non-retryable	**InvalidUriPathException** For troubleshooting help, see General configuration errors.	**XksProxyInvalidConfigurationException** in response to CreateCustomKeyStore and UpdateCustomKeyStore operations. **CustomKeyStoreInvalidStateException** in response to CreateKey operations. **KMSInvalidStateException** in response to cryptographic operations.

Error type	Exception displayed in the graph	Exception that AWS KMS returned to you
Non-retryable	**KeyNotFoundException** For troubleshooting help, see External key errors.	**XksKeyNotFoundException** in response to CreateKey operations. **KMSInvalidStateException** in response to cryptographic operations.
Retryable	**ThrottlingException** The external key store proxy rejected the request due to a very high request rate. Reduce the frequency of your calls using KMS keys in this external key store.	**XksProxyUriUnreachableException** in response to CreateCustomKeyStore and UpdateCustomKeyStore operations. **CustomKeyStoreInvalidStateException** in response to CreateKey operations. **KMSInvalidStateException** in response to cryptographic operations.
Non-retryable	**UnsupportedOperationException** For troubleshooting help, see Cryptographic operation errors for the external key.	**XksKeyInvalidResponseException** in response to CreateKey operations. **KMSInvalidStateException** in response to cryptographic operations.

Error type	Exception displayed in the graph	Exception that AWS KMS returned to you
Non-retryable	**ValidationException** For troubleshooting help, see Proxy issues.	**XksProxyInvalidResponseException** in response to CreateCustomKeyStore and UpdateCustomKeyStore operations. **CustomKeyStoreInvalidStateException** in response to CreateKey operations. **KMSInvalidStateException** in response to cryptographic operations.
Retryable	**XksProxyUnreachableException** If you see this error repeatedly, verify that your external key store proxy is active and is connected to the network, and that its URI path and endpoint URI or VPC service name are correct in your external key store.	**XksProxyUriUnreachableException** in response to CreateCustomKeyStore and UpdateCustomKeyStore operations. **CustomKeyStoreInvalidStateException** in response to CreateKey operations. **KMSInvalidStateException** in response to cryptographic operations.

The top 5 exceptions graph is derived from the XksProxyErrors metric. When you view a specific data point, the pop-up displays the value of the ExceptionName dimension alongside the number of times that the exception was recorded at that data point. The five list items are ordered from most frequent exception to least.

We recommend using the XksProxyErrors metric to create a CloudWatch alarm that notifies you of potential configuration problems by alerting you when more than five non-retryable errors are recorded in a one minute period. For more information, see Creating an Amazon CloudWatch alarm for non-retryable errors.

Certificate days to expire

The number of days until the TLS certificate for your external key store proxy endpoint (`XksProxyUriEndpoint`) expires. Use this graph to monitor upcoming expiration of your TLS certificate.

When the certificate expires, AWS KMS cannot communicate with the external key store proxy. All data protected by KMS keys in your external key store becomes inaccessible until you renew the certificate.

The certificate days to expire graph is derived from the XksProxyCertificateDaysToExpire metric. We strongly recommend using this metric to create a CloudWatch alarm that notifies you about the upcoming expiration. Certificate expiration might prevent you from accessing your encrypted resources. Set the alarm to give your organization time to renew the certificate before it expires. For more information, see Creating an Amazon CloudWatch alarm for certificate expiration.

Setting alarms

The graphs in the monitoring section provide an overview of the health of your external key stores and KMS keys in external key stores for a given period of time. However, you can create Amazon CloudWatch alarms based on external key store metrics to notify you when a metric value exceeds a threshold you specified. The alarm can send the message to an Amazon Simple Notification Service (Amazon SNS) topic or an Amazon EC2 Auto Scaling policy. For detailed information about CloudWatch alarms, see Using Amazon CloudWatch alarms in the *Amazon CloudWatch User Guide*.

Before creating an Amazon CloudWatch alarm, you need an Amazon SNS topic. For details, see Creating an Amazon SNS topic in the *Amazon CloudWatch User Guide*.

Topics

- Creating an Amazon CloudWatch alarm for non-retryable errors

Creating an Amazon CloudWatch alarm for certificate expiration

This alarm uses the XksProxyCertificateDaysToExpire metric that AWS KMS publishes to CloudWatch to record the anticipated expiration of the TLS certificate associated with your external key store proxy endpoint. You cannot create a single alarm for all external key stores in your account or an alarm for external key stores that you might create in the future.

We recommend setting the alarm to alert you 10 days before your certificate is set to expire, but you should set the threshold that best fits your needs.

Create the alarm

Follow the instructions in Create a CloudWatch alarm based on a static threshold using the following required values. For other fields, accept the default values and provide names as requested.

Field	Value
Select metric	Choose **KMS**, then choose **XKS Proxy Certificate Metrics**. Select the check box next to the `XksProxyCertificateName` that you want to monitor. Then choose **Select metric**.
Statistic	Minimum
Period	5 minutes
Threshold type	Static
Whenever …	Whenever **XksProxyCertificateDaysToExpire** is `Lower` than `10`.

Creating an Amazon CloudWatch alarm for response timeout

This alarm uses the XksProxyLatency metric that AWS KMS publishes to CloudWatch to record the number of milliseconds it takes for an external key store proxy to respond to an AWS KMS request.

You cannot create a single alarm for all external key stores in your account or an alarm for external key stores that you might create in the future.

AWS KMS expects the external key store proxy to respond to each request within 250 milliseconds. We recommend setting an alarm to alert you when your external key store proxy takes longer than 200 milliseconds to respond, but you should set the threshold that best fits your needs.

Create the alarm

Follow the instructions in Create a CloudWatch alarm based on a static threshold using the following required values. For other fields, accept the default values and provide names as requested.

Field	Value
Select metric	Choose **KMS**, then choose **XKS Proxy Latency Metrics**. Select the check box next to the `KmsOperation` that you want to monitor. Then choose **Select metric**.
Statistic	Average
Period	5 minutes
Threshold type	Static
Whenever ...	Whenever **XksProxyLatency** is `Greater` than `200`.

Creating an Amazon CloudWatch alarm for retryable errors

This alarm uses the XksProxyErrors metric that AWS KMS publishes to CloudWatch to record the number of exceptions related to AWS KMS requests to your external key store proxy. You cannot create a single alarm for all external key stores in your account or an alarm for external key stores that you might create in the future.

Retryable errors will lower your reliability percentage and can indicate networking errors. We recommend setting an alarm to alert you when more than five retryable errors are recorded in a one minute period, but you should set the threshold that best fits your needs.

Follow the instructions in Create a CloudWatch alarm based on a static threshold using the following required values. For other fields, accept the default values and provide names as requested.

Field	Value
Select metric	Choose the **Query** tab. Choose `AWS/KMS` for **Namespace**. Enter `SUM(XksProxyErrors)` for **Metric name**. Enter `ErrorType = Retryable` for **Filter by**. Choose **Run**. Then choose **Select metric**.
Label	*`Retryable errors`*
Period	1 minute
Threshold type	Static
Whenever ...	Whenever **q1** is `Greater` than `5`.

Creating an Amazon CloudWatch alarm for non-retryable errors

This alarm uses the XksProxyErrors metric that AWS KMS publishes to CloudWatch to record the number of exceptions related to AWS KMS requests to your external key store proxy. You cannot create a single alarm for all external key stores in your account or an alarm for external key stores that you might create in the future.

Non-retryable errors can indicate a problem with the configuration of your external key store. We recommend setting an alarm to alert you when more than five non-retryable errors are recorded in a one minute period, but you should set the threshold that best fits your needs.

Follow the instructions in Create a CloudWatch alarm based on a static threshold using the following required values. For other fields, accept the default values and provide names as requested.

Field	Value
Select metric	Choose the **Query** tab. Choose `AWS/KMS` for **Namespace**. Enter `SUM(XksProxyErrors)` for **Metric name**. Enter `ErrorType = Non-retryable` for **Filter by**. Choose **Run**. Then choose **Select metric**.
Label	*`Non-retryable errors`*
Period	1 minute
Threshold type	Static
Whenever ...	Whenever **q1** is `Greater` than `5`.

Connecting and disconnecting an external key store

New external key stores are not connected. To create and use AWS KMS keys in your external key store, you need to connect your external key store to its external key store proxy. You can connect and disconnect your external key store at any time, and view its connection state.

While your external key store is disconnected, AWS KMS cannot communicate with your external key store proxy. As a result, you can view and manage your external key store and its existing KMS keys. However, you cannot create KMS keys in your external key store, or use its KMS keys in cryptographic operations. You might need to disconnect your external key store at some point, such as when editing its properties, but plan accordingly. Disconnecting the key store might disrupt the operation of AWS services that use its KMS keys.

You are not required to connect your external key store. You can leave an external key store in a disconnected state indefinitely and connect it only when you need to use it. However, you might want to test the connection periodically to verify that the settings are correct and it can be connected.

When you disconnect a custom key store, the KMS keys in the key store become unusable right away (subject to eventual consistency). However, resources encrypted with data keys protected by the KMS key are not affected until the KMS key is used again, such as to decrypt the data key. This issue affects AWS services, many of which use data keys to protect your resources. For details, see How unusable KMS keys affect data keys.

> **Note**
>
> External key stores are in a `DISCONNECTED` state only when the key store has never been connected or you explicitly disconnect it. A `CONNECTED` state does not indicate that external key store or its supporting components are operating efficiently. For information about the performance of your external key store components, see the graphs in **Monitoring** section of the detail page for each external key store. For details, see Monitoring an external key store.
>
> Your external key manager might provide additional methods of stopping and restarting communication between your AWS KMS external key store and your external key store proxy, or between your external key store proxy and external key manager. For details, see your external key manager documentation.

Topics

Connecting an external key store

When your external key store is connected to its external key store proxy, you can create KMS keys in your external key store and use its existing KMS keys in cryptographic operations.

The process that connects an external key store to its external key store proxy differs based on the connectivity of the external key store.

- When you connect an external key store with public endpoint connectivity, AWS KMS sends a GetHealthStatus request to the external key store proxy to validate the proxy URI endpoint, proxy URI path, and proxy authentication credential. A successful response from the proxy confirms that the proxy URI endpoint and proxy URI path are accurate and accessible, and that the proxy authenticated the request signed with the proxy authentication credential for the external key store.

- When you connect an external key store with VPC endpoint service connectivity to its external key store proxy, AWS KMS does the following:
  - Confirms that the domain for the private DNS name specified in the proxy URI endpoint is verified.
  - Creates an interface endpoint from an AWS KMS VPC to your VPC endpoint service.
  - Creates a private hosted zone for the private DNS name specified in the proxy URI endpoint
  - Sends a GetHealthStatus request to the external key store proxy. A successful response from the proxy confirms that the proxy URI endpoint and proxy URI path are accurate and accessible, and that the proxy authenticated the request signed with the proxy authentication credential for the external key store.

The connect operation begins the process of connecting your custom key store, but connecting an external key store it its external proxy takes approximately five minutes. A success response from the connect operation does not indicate that the external key store is connected. To confirm that the connection was successful, use the AWS KMS console or the DescribeCustomKeyStores operation to view the connection state of external your key store.

When the connection state is `FAILED`, a connection error code is displayed in the AWS KMS console and is added to the `DescribeCustomKeyStore` response. For help interpreting connection error codes, see Connection error codes for external key stores.

Disconnecting an external key store

When you disconnect an external key store with VPC endpoint service connectivity from its external key store proxy, AWS KMS deletes its interface endpoint to the VPC endpoint service and removes the network infrastructure that it created to support the connection. No equivalent process is required for external key stores with public endpoint connectivity. This action does not affect the VPC endpoint service or any of its supporting components, and it does not affect the external key store proxy or any external components.

While the external key store is disconnected, AWS KMS does not send any requests to the external key store proxy. The connection state of the external key store is `DISCONNECTED`. The KMS keys in the disconnected external key store are in an `UNAVAILABLE` key state (unless they are pending deletion), which means that they cannot be used in cryptographic operations. However, you can still view and manage your external key store and its existing KMS keys.

The disconnected state is designed to be temporary and reversible. You can reconnect your external key store at any time. Typically, no reconfiguration is necessary. However, if any properties of the associated external key store proxy have changed while it was disconnected, such as rotation of its proxy authentication credential, you must edit the external key store settings before reconnecting.

> **Note**
>
> While a custom key store is disconnected, all attempts to create KMS keys in the custom key store or to use existing KMS keys in cryptographic operations will fail. This action can prevent users from storing and accessing sensitive data.

To better estimate the effect of disconnecting your external key store, identify the KMS keys in the external key store and determine their past use.

You might disconnect an external key store for reasons such as the following:

- **To edit its properties.** You can edit the custom key store name, proxy URI path, and proxy authentication credential while the external key store is connected. However, to edit the proxy connectivity type, proxy URI endpoint, or VPC endpoint service name, you must first disconnect the external key store. For details, see Editing external key store properties.
- **To stop all communication** between AWS KMS and the external key store proxy. You can also stop communication between AWS KMS and your proxy by disabling your endpoint or VPC endpoint service. In addition, your external key store proxy or key management software might provide additional mechanisms to prevent AWS KMS from communicating with the proxy or to prevent the proxy from accessing your external key manager.
- **To disable all KMS keys** in the external key store. You can disable and re-enable KMS keys in an external key store by using the AWS KMS console or the DisableKey operation. These operations complete quickly (subject to eventual consistency), but they act on one KMS key at a time. Disconnecting the external key store changes the key state of all KMS keys in the external key store to `Unavailable`, which prevents them from being used in any cryptographic operation.
- **To repair a failed connection attempt**. If an attempt to connect an external key store fails (the connection state of the custom key store is `FAILED`), you must disconnect the external key store before you try to connect it again.

Connection state

Connecting and disconnecting changes the *connection state* of your custom key store. Connection state values are the same for AWS CloudHSM key stores and external key stores.

To view the connection state of your custom key store, use the DescribeCustomKeyStores operation or AWS KMS console. **Connection state** appears in each custom key store table, in the **General configuration** section of the detail page for each custom key store, and on the **Cryptographic configuration** tab of KMS keys in a custom key store. For details, see Viewing an AWS CloudHSM key store and Viewing an external key store.

An custom key store can have one of the following connection states:

- `CONNECTED`: The custom key store is connected to its backing key store. You can create and use KMS keys in the custom key store.

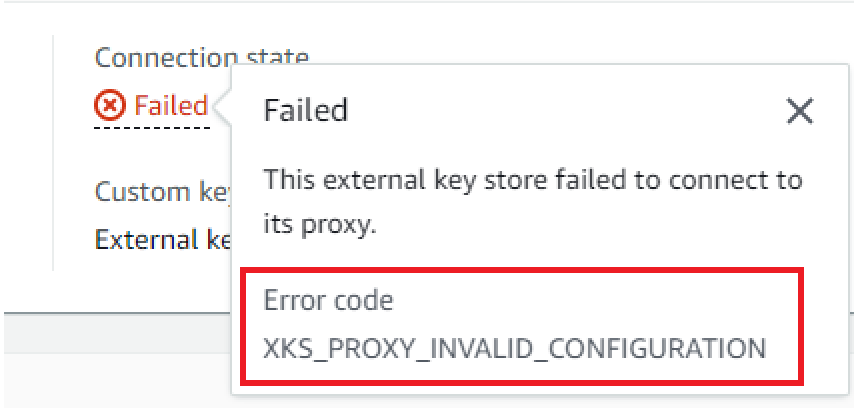
  The *backing key store* for an AWS CloudHSM key store is its associated AWS CloudHSM cluster. The *backing key store* for an external key store is external key store proxy and the external key manager that it supports.

  A CONNECTED state means that a connection succeeded and the custom key store has not been intentionally disconnected. It does not indicate that the connection is operating properly. For information about the status of the AWS CloudHSM cluster associated with your AWS CloudHSM key store, see Getting CloudWatch metrics for AWS CloudHSM in the AWS CloudHSM User Guide. For information about the status and operation of your external key store, see the graphs in the **Monitoring** section of the detail page for each external key store. For details, see Monitoring an external key store.
- `CONNECTING`: The process of connecting an custom key store is in progress. This is a transient state.
- `DISCONNECTED`: The custom key store has never been connected to its backing, or it was intentionally disconnected by using the AWS KMS console or the DisconnectCustomKeyStore operation.
- `DISCONNECTING`: The process of disconnecting an custom key store is in progress. This is a transient state.
- `FAILED`: An attempt to connect the custom key store failed. The `ConnectionErrorCode` in the DescribeCustomKeyStores response indicates the problem.

To connect an custom key store, its connection state must be DISCONNECTED. If the connection state is FAILED, use the ConnectionErrorCode to identify and resolve the problem. Then disconnect the custom key store before trying to connect it again. For help with connection failures, see External key store connection errors. For help responding to a connection error code, see Connection error codes for external key stores.

To view the connection error code:

- In the DescribeCustomKeyStores response, view the value of the ConnectionErrorCode element. This element appears in the DescribeCustomKeyStores response only when the ConnectionState is FAILED.
- To view the connection error code in the AWS KMS console, on detail page for the external key store and hover over the **Failed** value.



Failed

This external key store failed to connect to its proxy.

Error code

XKS_PROXY_INVALID_CONFIGURATION

Connect an external key store (console)

You can use the AWS KMS console to connect an external key store to its external key store proxy.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Custom key stores**, **External key stores**.
4. Choose the row of the external key store you want to connect.

   If the connection state of the external key store is **FAILED**, you must disconnect the external key store before you connect it.
5. From the **Key store actions** menu, choose **Connect**.

The connection process typically takes about five minutes to complete.When the operation completes, the connection state changes to **CONNECTED**.

If the connection state is **Failed**, hover over the connection state to see the *connection error code*, which explains the cause of the error. For help responding to a connection error code, see Connection error codes for external key stores. To connect an external key store with a **Failed** connection state, you must first disconnect the custom key store.

Connect an external key store (API)

To connect a disconnected external key store, use the ConnectCustomKeyStore operation.

Before connecting, the connection state of the external key store must be `DISCONNECTED`. If the current connection state is `FAILED`, disconnect the external key store, and then connect it.

The connection process takes about five minutes to complete. Unless it fails quickly, `ConnectCustomKeyStore` returns an HTTP 200 response and a JSON object with no properties. However, this initial response does not indicate that the connection was successful. To determine whether the external key store is connected, see the connection state in the DescribeCustomKeyStores response.

The examples in this section use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

To identify the external key store, use its custom key store ID. You can find the ID on the **Custom key stores** page in the console or by using the DescribeCustomKeyStores operation. Before running this example, replace the example ID with a valid one.

```
$ aws kms connect-custom-key-store --custom-key-store-id cks-1234567890abcdef0
```

The `ConnectCustomKeyStore` operation does not return the `ConnectionState` in its response. To verify that the external key store is connected, use the DescribeCustomKeyStores operation. By default, this operation returns all custom keys stores in your account and Region. But you can use either the `CustomKeyStoreId` or `CustomKeyStoreName` parameter (but not both) to limit the response to particular custom key stores. A `ConnectionState` value of `CONNECTED` indicates that the external key store is connected to its external key store proxy.

```
$ aws kms describe-custom-key-stores --custom-key-store-name ExampleXksVpc
{
    "CustomKeyStores": [
    {
      "CustomKeyStoreId": "cks-9876543210fedcba9",
      "CustomKeyStoreName": "ExampleXksVpc",
      "ConnectionState": "CONNECTED",
      "CreationDate": "2022-12-13T18:34:10.675000+00:00",
      "CustomKeyStoreType": "EXTERNAL_KEY_STORE",
      "XksProxyConfiguration": {
        "AccessKeyId": "ABCDE98765432EXAMPLE",
        "Connectivity": "VPC_ENDPOINT_SERVICE",
        "UriEndpoint": "https://example-proxy-uri-endpoint-vpc",
        "UriPath": "/example/prefix/kms/xks/v1",
        "VpcEndpointServiceName": "com.amazonaws.vpce.us-east-1.vpce-svc-example"
      }
    }
  ]
}
```

If the `ConnectionState` value in the `DescribeCustomKeyStores` response is `FAILED`, the `ConnectionErrorCode` element indicates the reason for the failure.

In the following example, the `XKS_VPC_ENDPOINT_SERVICE_NOT_FOUND` value for the `ConnectionErrorCode` indicates that AWS KMS can't find the VPC endpoint service that it uses to communicate with the external key store proxy. Verify that the `XksProxyVpcEndpointServiceName` is correct, the AWS KMS service principal is an allowed principal on the Amazon VPC endpoint service, and that the VPC endpoint service does not require acceptance of connection requests. For help responding to a connection error code, see Connection error codes for external key stores.

```
$ aws kms describe-custom-key-stores --custom-key-store-name ExampleXksVpc
{
    "CustomKeyStores": [
    {
      "CustomKeyStoreId": "cks-9876543210fedcba9",
      "CustomKeyStoreName": "ExampleXksVpc",
      "ConnectionState": "FAILED",
      "ConnectionErrorCode": "XKS_VPC_ENDPOINT_SERVICE_NOT_FOUND",
      "CreationDate": "2022-12-13T18:34:10.675000+00:00",
      "CustomKeyStoreType": "EXTERNAL_KEY_STORE",
      "XksProxyConfiguration": {
        "AccessKeyId": "ABCDE98765432EXAMPLE",
        "Connectivity": "VPC_ENDPOINT_SERVICE",
        "UriEndpoint": "https://example-proxy-uri-endpoint-vpc",
        "UriPath": "/example/prefix/kms/xks/v1",
        "VpcEndpointServiceName": "com.amazonaws.vpce.us-east-1.vpce-svc-example"
      }
    }
  ]
}
```

Disconnect an external key store (console)

You can use the AWS KMS console to connect an external key store to its external key store proxy. This process takes about 5 minutes to complete.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Custom key stores**, **External key stores**.
4. Choose the row of the external key store you want to disconnect.
5. From the **Key store actions** menu, choose **Disconnect**.

When the operation completes, the connection state changes from **DISCONNECTING** to **DISCONNECTED**. If the operation fails, an error message appears that describes the problem and provides help on how to fix it. If you need more help, see External key store connection errors.

Disconnect an external key store (API)

To disconnect a connected external key store, use the DisconnectCustomKeyStore operation. If the operation is successful, AWS KMS returns an HTTP 200 response and a JSON object with no properties. The process takes about five minutes to complete. To find the connection state of the external key store, use the DescribeCustomKeyStores operation.

The examples in this section use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

This example disconnects an external key store with VPC endpoint service connectivity. Before running this example, replace the example custom key store ID with a valid one.

```
$ aws kms disconnect-custom-key-store --custom-key-store-id cks-1234567890abcdef0
```

To verify that the external key store is disconnected, use the DescribeCustomKeyStores operation. By default, this operation returns all custom keys stores in your account and Region. But you can use either the CustomKeyStoreId and CustomKeyStoreName parameter (but not both) to limit the response to particular custom key stores. The ConnectionState value of DISCONNECTED indicates that this example external key store is no longer connected to its external key store proxy.

```
$ aws kms describe-custom-key-stores --custom-key-store-name ExampleXksVpc
{
    "CustomKeyStores": [
    {
      "CustomKeyStoreId": "cks-9876543210fedcba9",
      "CustomKeyStoreName": "ExampleXksVpc",
      "ConnectionState": "DISCONNECTED",
      "CreationDate": "2022-12-13T18:34:10.675000+00:00",
      "CustomKeyStoreType": "EXTERNAL_KEY_STORE",
      "XksProxyConfiguration": {
        "AccessKeyId": "ABCDE98765432EXAMPLE",
        "Connectivity": "VPC_ENDPOINT_SERVICE",
        "UriEndpoint": "https://example-proxy-uri-endpoint-vpc",
        "UriPath": "/example/prefix/kms/xks/v1",
        "VpcEndpointServiceName": "com.amazonaws.vpce.us-east-1.vpce-svc-example"
      }
    }
  ]
}
```

Deleting an external key store

When you delete an external key store, AWS KMS deletes all metadata about the external key store from AWS KMS, including information about its external key store proxy. This operation does not affect the external key store proxy, external key manager, external keys, or any AWS resources that you created to support the external key store, such as an Amazon VPC or a VPC endpoint service.

Before you delete an external key store, you must delete all of the KMS keys from the key store and disconnect the key store from its external key store proxy. Otherwise, attempts to delete the key store fail.

Deleting an external key store is irreversible, but you can create a new external key store and associate it with the same external key store proxy and external key manager. However, you cannot recreate the symmetric encryption KMS keys in the external key store, even you have access to the same external key material. AWS KMS includes metadata in the symmetric ciphertext unique to each KMS key. This security feature ensures that only the KMS key that encrypted the data can decrypt it.

Instead of deleting the external key store, consider disconnecting it. While an external key store is disconnected, you can manage the external key store and its AWS KMS keys but you cannot create or use KMS keys in the external key store. You can reconnect the external key store at any time and resume using its KMS keys to encrypt and decrypt data. There is no cost for a disconnected external key store proxy or its unavailable KMS keys.

Topics

- Delete an external key store (console)
- Delete an external key store (API)

Delete an external key store (console)

You can use the AWS KMS console to delete an external key store.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Custom key stores**, **External key stores**.
4. Find the row that represents the external key store that you want to delete. If the **Connection state** of the external key store is not **DISCONNECTED**, you must disconnect the external key store before you delete it.
5. From the **Key store actions** menu, choose **Delete**.

When the operation completes, a success message appears and the external key store no longer appears in the key store list. If the operation is unsuccessful, an error message appears that describes the problem and provides help on how to fix it. If you need more help, see Troubleshooting external key stores.

Delete an external key store (API)

To delete an external key store, use the DeleteCustomKeyStore operation. If the operation is successful, AWS KMS returns an HTTP 200 response and a JSON object with no properties.

To begin, disconnect the external key store. Before running this command, replace the example custom key store ID with a valid one.

```
$ aws kms disconnect-custom-key-store --custom-key-store-id cks-1234567890abcdef0
```

After the external key store is disconnected, you can use the DeleteCustomKeyStore operation to delete it.

```
$ aws kms delete-custom-key-store --custom-key-store-id cks-1234567890abcdef0
```

To confirm that the external key store is deleted, use the DescribeCustomKeyStores operation.

```
$ aws kms describe-custom-key-stores

{
    "CustomKeyStores": []
}
```

If you specify a custom key store name or ID that no longer exists, AWS KMS returns a `CustomKeyStoreNotFoundException` exception.

```
$ aws kms describe-custom-key-stores --custom-key-store-id cks-1234567890abcdef0

An error occurred (CustomKeyStoreNotFoundException) when calling the
 DescribeCustomKeyStore operation:
```

Managing KMS keys in an external key store

To create, view, manage, use, and schedule deletion of the KMS keys in an external key store, you use procedures that are very similar to those you use for other KMS keys. However, when you create a KMS key in an external key store, you specify an external key store and an external key. When you use a KMS key in an external key store, encryption and decryption operations are performed by your external key manager using the specified external key.

AWS KMS cannot create, view, update, or delete any cryptographic keys in your external key manager. AWS KMS never directly accesses your external key manager or any external key. All requests for cryptographic operations are mediated by your external key store proxy. To use a KMS key in an external key store, the external key store that hosts the KMS key must be connected to its external key store proxy.

Supported features

In addition to the procedures discussed in this section, you can do the following with KMS keys in an external key store:

- Use key policies, IAM policies, and grants to control access to the KMS keys.
- Enable and disable the KMS keys. These actions do not affect the external key in your external key manager.
- Assign tags and create aliases, and use attribute-based access control (ABAC) to authorize access to the KMS keys.
- Use the KMS keys with AWS services that integrate with AWS KMS and support customer managed keys.

Unsupported features

- External key stores support only symmetric encryption KMS keys. You cannot create HMAC KMS keys or asymmetric KMS keys in an external key store.
- GenerateDataKeyPair and GenerateDataKeyPairWithoutPlaintext are not supported on KMS keys in an external key store.
- You cannot use an AWS CloudFormation template to create an external key store or a KMS key in an external key store.
- Multi-Region keys are not supported in an external key store.
- KMS keys with imported key material are not supported in an external key store.
- Automatic key rotation is not supported for KMS keys in an external key store.

Topics

Creating KMS keys in an external key store

After you have created and connected your external key store, you can create AWS KMS keys in your key store. They must be symmetric encryption KMS keys with an origin value of **External key store** (`EXTERNAL_KEY_STORE`). You cannot create asymmetric KMS keys, HMAC KMS keys or KMS keys with imported key material in a custom key store. Also, you cannot use symmetric encryption KMS keys in a custom key store to generate asymmetric data key pairs.

A KMS key in an external key store might have poorer latency, durability and availability than a standard KMS key because it depends on components located outside of AWS. Before creating or using a KMS key in an external key store, verify that you require a key with external key store properties.

> **Note**
>
> Some external key managers provide a simpler method for creating KMS keys in an external key store. For details, see your external key manager documentation.

To create a KMS key in your external key store, you specify the following:

- The ID of your external key store.
- A key material origin of External key store (`EXTERNAL_KEY_STORE`).
- The ID of an existing external key in the external key manager associated with your external key store. This external key serves as key material for the KMS key. You cannot change the external key ID after you create the KMS key.

  AWS KMS provides the external key ID to your external key store proxy in requests for encryption and decryption operations. AWS KMS cannot directly access your external key manager or any of its cryptographic keys.

In addition to the external key, a KMS key in an external key store also has AWS KMS key material. All data encrypted under the KMS key is first encrypted in AWS KMS using the key's AWS KMS key material and then by your external key manager using your external key. This double encryption process ensures that ciphertext protected by a KMS key in an external key store is at least as strong as ciphertext protected only by AWS KMS. For details, see How external key stores work.

When the `CreateKey` operation succeeds, the key state of the new KMS key is `Enabled`. When you view a KMS key in an external key store you can see typical properties, like its key ID, key spec, key usage, key state, and creation date. But you can also see the ID and connection state of the external key store and the ID of the external key.

If your attempt to create a KMS key in your external key store fails, use the error message to identify the cause. It might indicate that the external key store is not connected (`CustomKeyStoreInvalidStateException`), that your external key store proxy cannot find an external key with the specified external key ID (`XksKeyNotFoundException`), or that the external key is already associated with a KMS key in the same external key store `XksKeyAlreadyInUseException`.

.

For an example of the AWS CloudTrail log of the operation that creates a KMS key in an external key store, see CreateKey.

Topics

- Requirements for a KMS key in an external key store
- Create a KMS key in an external key store (console)
- Create a KMS key in an external key store (AWS KMS API)

Requirements for a KMS key in an external key store

To create a KMS key in an external key store, the following properties are required of the external key store, the KMS key, and the external key that serves as the external cryptographic key material for the KMS key.

External key store requirements

- Must be connected to its external key store proxy.

  To view the connection state of your external key store, see Viewing an external key store. To connect your external key store, see Connecting and disconnecting an external key store.

KMS key requirements

You cannot change these properties after you create the KMS key.

- Key spec: SYMMETRIC_DEFAULT
- Key usage: ENCRYPT_DECRYPT
- Key material origin: EXTERNAL_KEY_STORE
- Multi-Region: FALSE

External key requirements

- 256-bit AES cryptographic key (256 random bits). The `KeySpec` of the external key must be `AES_256`.
- Enabled and available for use. The `Status` of the external key must be `ENABLED`.
- Configured for encryption and decryption. The `KeyUsage` of the external key must include `ENCRYPT` and `DECRYPT`.
- Used only with this KMS key. Each `KMS key` in an external key store must be associated with a different external key.

  AWS KMS also recommends that the external key be used exclusively for the external key store. This restriction makes it easier to identify and resolve problems with the key.
- Accessible by the external key store proxy for the external key store.

  If the external key store proxy can't find the key using the specified external key ID, the `CreateKey` operation fails.
- Can handle the anticipated traffic that your use of AWS services generates. AWS KMS recommends that external keys be prepared to handle up to 1800 requests per second.

Create a KMS key in an external key store (console)

There are two ways to create a KMS key in an external key store.

- Method 1 (recommended): Choose an external key store, then create a KMS key in that external key store.
- Method 2: Create a KMS key, then indicate that it's in an external key store.

If you use Method 1, where you choose your external key store before you create your key, AWS KMS chooses all required KMS key properties for you and fills in the ID of your external key store. This method avoids errors you might make when creating your KMS key.

> **Note**
>
> Do not include confidential or sensitive information in the alias, description, or tags. These fields may appear in plain text in CloudTrail logs and other output.

Method 1 (recommended): Start in your external key store

To use this method, choose your external key store, then create a KMS key. The AWS KMS console chooses all required properties for you and fills in the ID of your external key store. This method avoids many errors you might make when creating your KMS key.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Custom key stores**, **External key stores**.
4. Choose the name of your external key store.
5. In the top right corner, choose **Create a KMS key in this key store**.

   If the external key store is *not* connected, you will be prompted to connect it. If the connection attempt fails, you need to resolve the problem and connect the external key store before you can create a new KMS key in it.

   If the external key store is connected, you are redirected to the **Customer managed keys** page for creating a key. The required **Key configuration** values are already chosen for you. Also, the custom key store ID of your external key store is filled in, although you can change it.
6. Enter the key ID of an external key in your external key manager. This external key must fulfill the requirements for use with a KMS key. You cannot change this value after the key is created.

   If the external key has multiple IDs, enter the key ID that the external key store proxy uses to identify the external key.
7. Confirm that you intend to create a KMS key in the specified external key store.
8. Choose **Next**.

   The remainder of this procedure is the same as creating a standard KMS key.
9. Type an alias (required) and a description (optional) for the KMS key.
10. (Optional). On the **Add Tags** page, add tags that identify or categorize your KMS key.

    When you add tags to your AWS resources, AWS generates a cost allocation report with usage and costs aggregated by tags. Tags can also be used to control access to a KMS key. For information about tagging KMS keys, see Tagging keys and ABAC for AWS KMS.
11. Choose **Next**.
12. In the **Key Administrators** section, select the IAM users and roles who can manage the KMS key. For more information, see Allows key administrators to administer the KMS key.

> **Note**
>
> IAM policies can give other IAM users and roles permission to use the KMS key. IAM best practices discourage the use of IAM users with long-term credentials. Whenever possible, use IAM roles, which provide temporary credentials. For details, see Security best practices in IAM in the *IAM User Guide*.

13. (Optional) To prevent these key administrators from deleting this KMS key, clear **Allow key administrators to delete this key** check box.

    Deleting a KMS key is a destructive and irreversible operation that can render ciphertext unrecoverable. You cannot recreate a symmetric KMS key in an external key store, even if you have the external key material. However, deleting a KMS key has no effect on its associated external key. For information about deleting a KMS key from an external key store, see Scheduling deletion of KMS keys from an external key store.

14. Choose **Next**.

15. In the **This account** section, select the IAM users and roles in this AWS account that can use the KMS key in cryptographic operations. For more information, see Allows key users to use the KMS key.

    > **Note**
    >
    > IAM policies can give other IAM users and roles permission to use the KMS key. IAM best practices discourage the use of IAM users with long-term credentials. Whenever possible, use IAM roles, which provide temporary credentials. For details, see Security best practices in IAM in the *IAM User Guide*.

16. (Optional) You can allow other AWS accounts to use this KMS key for cryptographic operations. To do so, in the **Other AWS accounts** section at the bottom of the page, choose **Add another AWS account** and enter the AWS account ID of an external account. To add multiple external accounts, repeat this step.

> **Note**
>
> Administrators of the other AWS accounts must also allow access to the KMS key by creating IAM policies for their users. For more information, see Allowing users in other accounts to use a KMS key.

17. Choose **Next**.
18. Review the key settings that you chose. You can still go back and change all settings.
19. When you're done, choose **Finish** to create the key.

Method 2: Start in Customer managed keys

This procedure is the same as the procedure to create a symmetric encryption key with AWS KMS key material. But, in this procedure, you specify the custom key store ID of the external key store and the key ID of the external key. You must also specify the required property values for a KMS key in an external key store, such as the key spec and key usage.

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Choose **Create key**.
5. Choose **Symmetric**.
6. In **Key usage**, the **Encrypt and decrypt** option is selected for you. Do not change it.
7. Choose **Advanced options**.
8. For **Key material origin**, choose **External key store**.
9. Confirm that you intend to create a KMS key in the specified external key store.
10. Choose **Next**.
11. Choose the row that represents the external key store for your new KMS key.

    You cannot choose a disconnected external key store. To connect a key store that is disconnected, choose the key store name, and then, from **Key store actions**, choose, **Connect**. For details, see Connect an external key store (console).

12. Enter the key ID of an external key in your external key manager. This external key must fulfill the requirements for use with a KMS key. You cannot change this value after the key is created.

    If the external key has multiple IDs, enter the key ID that the external key store proxy uses to identify the external key.

13. Choose **Next**.

    The remainder of this procedure is the same as creating a standard KMS key.

14. Type an alias and an optional description for the KMS key.

15. (Optional). On the **Add Tags** page, add tags that identify or categorize your KMS key.

    When you add tags to your AWS resources, AWS generates a cost allocation report with usage and costs aggregated by tags. Tags can also be used to control access to a KMS key. For information about tagging KMS keys, see Tagging keys and ABAC for AWS KMS.

16. Choose **Next**.

17. In the **Key Administrators** section, select the IAM users and roles who can manage the KMS key. For more information, see Allows key administrators to administer the KMS key.

    > **Note**
    >
    > IAM policies can give other IAM users and roles permission to use the KMS key.

18. (Optional) To prevent these key administrators from deleting this KMS key, clear **Allow key administrators to delete this key** check box.

    Deleting a KMS key is a destructive and irreversible operation that can render ciphertext unrecoverable. You cannot recreate a symmetric KMS key in an external key store, even if you have the external key material. However, deleting a KMS key has no effect on its associated external key. For information about deleting a KMS key from an external key store, see Scheduling deletion of KMS keys from an external key store.

19. Choose **Next**.

20. In the **This account** section, select the IAM users and roles in this AWS account that can use the KMS key in cryptographic operations. For more information, see Allows key users to use the KMS key.

> **Note**
>
> IAM policies can give other IAM users and roles permission to use the KMS key.

21. (Optional) You can allow other AWS accounts to use this KMS key for cryptographic operations. To do so, in the **Other AWS accounts** section at the bottom of the page, choose **Add another AWS account** and enter the AWS account ID of an external account. To add multiple external accounts, repeat this step.

    > **Note**
    >
    > Administrators of the other AWS accounts must also allow access to the KMS key by creating IAM policies for their users. For more information, see Allowing users in other accounts to use a KMS key.

22. Choose **Next**.
23. Review the key settings that you chose. You can still go back and change all settings.
24. When you're done, choose **Finish** to create the key.

When the procedure succeeds, the display shows the new KMS key in the external key store that you chose. When you choose the name or alias of the new KMS key, the **Cryptographic configuration** tab on its detail page displays the origin of the KMS key (**External key store**), the name, ID, and type of the custom key store, and the ID, key usage, and status of the external key. If the procedure fails, an error message appears that describes the failure. For , see Troubleshooting external key stores.

> **Tip**
>
> To make it easier to identify KMS keys in a custom key store, on the **Customer managed keys** page, add the **Origin** and **Custom key store ID** column to the display. To change the table fields, choose the gear icon in the upper right corner of the page. For details, see Customizing your KMS key tables.

Create a KMS key in an external key store (AWS KMS API)

To create a new KMS key in an external key store, use the CreateKey operation. The following parameters are required:

- The `Origin` value must be `EXTERNAL_KEY_STORE`.
- The `CustomKeyStoreId` parameter identifies your external key store. The `ConnectionState` of the specified external key store must be `CONNECTED`. To find the `CustomKeyStoreId` and `ConnectionState`, use the `DescribeCustomKeyStores` operation.
- The `XksKeyId` parameter identifies the external key. This external key must fulfills the requirements for association with a KMS key.

You can also use any of the optional parameters of the `CreateKey` operation, such as using the `Policy` or Tags parameters.

> **Note**
>
> Do not include confidential or sensitive information in the `Description` or `Tags` fields. These fields may appear in plain text in CloudTrail logs and other output.

The examples in this section use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

This example command uses the CreateKey operation to create a KMS key in an external key store. The response includes the properties of the KMS keys, the ID of the external key store, and the ID, usage, and status of the external key. For detailed information about these fields, see Viewing KMS keys in an external key store.

Before running this command, replace the example custom key store ID with a valid ID.

```
$ aws kms create-key --origin EXTERNAL_KEY_STORE --custom-key-store-
id cks-1234567890abcdef0 --xks-key-id bb8562717f809024
{
  "KeyMetadata": {
    "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "AWSAccountId": "111122223333",
    "CreationDate": "2022-12-02T07:48:55-07:00",
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "CustomKeyStoreId": "cks-1234567890abcdef0",
    "Description": "",
    "Enabled": true,
    "EncryptionAlgorithms": [
      "SYMMETRIC_DEFAULT"
    ],
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "KeyManager": "CUSTOMER",
    "KeySpec": "SYMMETRIC_DEFAULT",
    "KeyState": "Enabled",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "MultiRegion": false,
    "Origin": "EXTERNAL_KEY_STORE",
    "XksKeyConfiguration": {
      "Id": "bb8562717f809024"
    }
  }
}
```

Viewing KMS keys in an external key store

To view the KMS keys in an external key store, use the AWS KMS console or the DescribeKey operation. You can use the same techniques that you would use to view any AWS KMS customer managed keys. To learn the basics, see Viewing keys.

In the AWS KMS console, the KMS keys in your external key store are displayed on the **Customer managed keys** page, along with all other customer managed keys in your AWS account and Region. To identify KMS keys in an external key store, filter by the distinctive origin value, **External key store**, and the custom key store ID.

For more information, see Viewing an external key store, Monitoring an external key store, and Logging AWS KMS API calls with AWS CloudTrail.

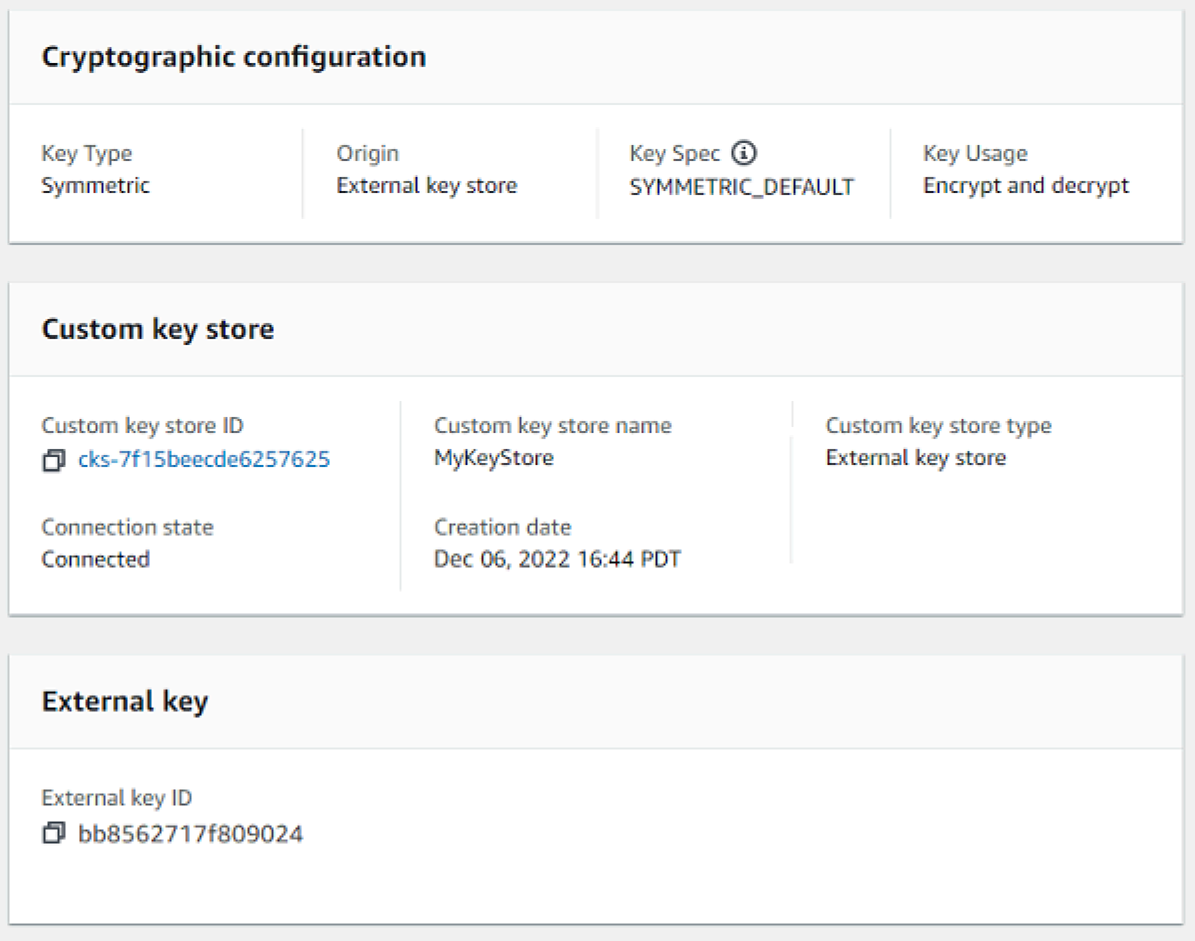
Topics

- Properties of KMS keys in an external key store
- Viewing KMS keys in an external key store (console)
- Viewing KMS keys in an external key store (AWS KMS API)

Properties of KMS keys in an external key store

Like all KMS keys, the KMS keys in an external key store, have a key ARN, key spec, and key usage values, but they also have properties and property values specific to KMS keys in an external key store. For example, the **Origin** value for all KMS keys in external key stores is **External key store**.

For a KMS key in an external key store, the **Cryptographic configuration** tab in the AWS KMS console include two additional sections, **Custom key store** and **External key**.



Cryptographic configuration

Key Type	Origin	Key Spec	Key Usage
Symmetric	External key store	SYMMETRIC_DEFAULT	Encrypt and decrypt

Custom key store

Custom key store ID	Custom key store name	Custom key store type
cks-7f15beecde6257625	MyKeyStore	External key store

Connection state	Creation date
Connected	Dec 06, 2022 16:44 PDT

External key

External key ID
bb8562717f809024

Custom key store properties

The following values appear in the **Custom key store** section of the **Cryptographic configuration** tab and in the DescribeKey response. These properties apply to all custom key stores, including AWS CloudHSM key stores and external key stores.

Custom key store ID

A unique ID that AWS KMS assigns to the custom key store.

Custom key store name

A friendly name that you assign to the custom key store when you create it. You can change this value at any time.

Custom key store type

The type of custom key store. Valid values are AWS CloudHSM (`AWS_CLOUDHSM`) or External key store (`EXTERNAL_KEY_STORE`). You cannot change the type after you create the custom key store.

Creation date

The date that the custom key store was created. This date is displayed in local time for the AWS Region.

Connection state

Indicates whether the custom key store is connected to its backing key store. The connection state is `DISCONNECTED` only if the custom key store has never been connected to its backing key store, or it has been intentionally disconnected. For details, see the section called "Connection state".

External key properties

External key properties appear in the **External key** section of the **Cryptographic configuration** tab and in the `XksKeyConfiguration` element of the DescribeKey response.

The **External key** section appears in the AWS KMS console only for KMS keys in external key stores. It provides information about the external key associated with the KMS key. The *external key* is a cryptographic key outside of AWS that serves as the key material for the KMS key in the external key store. When you encrypt or decrypt with the KMS key, the operation is performed by your external key manager using the specified external key.

The following values appear in the **External key** section.

External key ID

The identifier for the external key in its external key manager. This is the value that the external key store proxy uses to identify the external key. You specify the ID of the external key when you create the KMS key and you cannot change it. If the external key ID value that you used to create the KMS key changes or becomes invalid, you must schedule the KMS key for deletion and create a new KMS key with the correct external key ID value.

Viewing KMS keys in an external key store (console)

To view the KMS keys in an external key store (Console)

1. Open the AWS KMS console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. To identify the KMS keys in your external key store, add the **Origin** and **Custom key store ID** fields to your key table. KMS keys in any external key store have an **Origin** value of **External key store**.

   In the upper-right corner, choose the gear icon, choose **Origin** and **Custom key store ID**, then choose **Confirm**.
5. Choose the alias or key ID of a KMS key in an external key store.
6. To view the properties specific to KMS keys in an external key store, choose the **Cryptographic configuration** tab. Special values for KMS keys in an external key store appear in the **Custom key store** and **External key** sections.

Viewing KMS keys in an external key store (AWS KMS API)

To view the KMS keys in an external key store (API)

You use the same AWS KMS API operations to view the KMS keys in an external key store that you would use for any KMS key, including ListKeys, DescribeKey, and GetKeyPolicy. For example, the following `describe-key` operation in the AWS CLI shows the special fields for a KMS key in an external key store. Before running a command like this one, replace the example KMS key ID with a valid value.

```
$ aws kms describe-key --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
{
  "KeyMetadata": {
    "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "AWSAccountId": "111122223333",
    "CreationDate": "2022-12-02T07:48:55-07:00",
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "CustomKeyStoreId": "cks-1234567890abcdef0",
    "Description": "",
    "Enabled": true,
    "EncryptionAlgorithms": [
      "SYMMETRIC_DEFAULT"
    ],
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "KeyManager": "CUSTOMER",
    "KeySpec": "SYMMETRIC_DEFAULT",
    "KeyState": "Enabled",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "MultiRegion": false,
    "Origin": "EXTERNAL_KEY_STORE",
    "XksKeyConfiguration": {
      "Id": "bb8562717f809024"
    }
  }
}
```

Using KMS keys in an external key store

After you create a symmetric encryption KMS key in an external key store, you can use it for the following cryptographic operations:

- Encrypt
- Decrypt
- GenerateDataKey
- GenerateDataKeyWithoutPlaintext
- ReEncrypt

The symmetric encryption operations that generate asymmetric data key pairs, GenerateDataKeyPair and GenerateDataKeyPairWithoutPlaintext, are *not* supported in custom key stores.

An encryption context is supported for all cryptographic operations with KMS keys in an external key store. As always, using an encryption context is a security best practice that AWS KMS recommends.

When you use your KMS key in a request, identify the KMS key by its key ID, key ARN, alias, or alias ARN. You do not need to specify the external key store. The response includes the same fields that are returned for any symmetric encryption KMS key. However, when you use a KMS key in an external key store, encryption and decryption operations are performed by your external key manager using the external key that is associated with the KMS key.

To ensure that ciphertext encrypted by a KMS key in an external key store is at least as secure as any ciphertext encrypted by a standard KMS key, AWS KMS uses double encryption. Data is first encrypted in AWS KMS using AWS KMS key material. Then it is encrypted by your external key manager using the external key for the KMS key. To decrypt double-encrypted ciphertext, the ciphertext is first decrypted by your external key manager using the external key for the KMS key. Then it is decrypted in AWS KMS using the AWS KMS key material for the KMS key.

To make this possible, the following conditions are required.

- The key state of the KMS key must be `Enabled`. To find the key state, see the **Status** field for customer managed keys the AWS KMS console or the `KeyState` field in the DescribeKey response.
- The external key store that hosts the KMS key must be connected to its external key store proxy, that is, the connection state of the external key store must be `CONNECTED`.

  You can view the connection state on the **External key stores** page in the AWS KMS console or in the DescribeCustomKeyStores response. The connection state of the external key store is also displayed on the detail page for the KMS key in the AWS KMS console. On the detail page, choose the **Cryptographic configuration** tab and see the **Connection state** field in the **Custom key store** section.

  If the connection state is `DISCONNECTED`, you must first connect it. If the connection state is `FAILED`, you must resolve the problem, disconnect the external key store, and then connect it. For instructions, see Connecting and disconnecting an external key store..
- The external key store proxy must be able to find the external key.
- The external key must be enabled and it must perform encryption and decryption.

  The status of the external key is independent of and not affected by changes in the key state of the KMS key, including enabling and disabling the KMS key. Similarly, disabling or deleting the external key doesn't change the key state of the KMS key, but cryptographic operations using the associated KMS key will fail.

If these conditions are not met, the cryptographic operation fails, and AWS KMS returns a `KMSInvalidStateException` exception. You might need to reconnect the external key store or use your external key manager tools to reconfigure or repair your external key. For additional help, see the section called “Troubleshooting external key stores”.

When using the KMS keys in an external key store, be aware that the KMS keys in each external key store share a custom key store request quota for cryptographic operations. If you exceed the quota, AWS KMS returns a `ThrottlingException`. For details about the custom key store request quota, see Custom key store request quotas.

Scheduling deletion of KMS keys from an external key store

When you are certain that you will not need to use an AWS KMS key for any cryptographic operation, you can schedule the deletion of the KMS key. Use the same procedure that you would use to schedule the deletion of any KMS key from AWS KMS. Deleting a KMS key from an external key store has no effect on the external key that served as its key material.

You can cancel the scheduled deletion of a KMS key during its mandatory waiting period. However, a deleted KMS key not recoverable. You cannot recreate a symmetric encryption KMS key in an external key store, even you use the same external key. Because each symmetric KMS key in an external key store has unique AWS KMS key material and metadata, only the AWS KMS key that encrypted a symmetric ciphertext can decrypt it.

> ⚠ **Warning**
>
> Deleting a KMS key is a destructive and potentially dangerous operation that prevents you from recovering all data encrypted under the KMS key. Before scheduling deletion of the KMS key, examine past usage of the KMS key and create a Amazon CloudWatch alarm that alerts you when someone tries to use the KMS key while it is pending deletion. Whenever possible, disable the KMS key, instead of deleting it.

When you schedule deletion of a KMS key from an external key store, its key state changes to **Pending deletion**. The KMS key remains in the **Pending deletion** state throughout the waiting period, even if the KMS key becomes unavailable because you have disconnected the external key store. This allows you to cancel the deletion of the KMS key at any time during the waiting period. When the waiting period expires, AWS KMS deletes the KMS key from AWS KMS.

When you schedule the deletion of a KMS key from an external key store, the KMS key becomes unusable right away (subject to eventual consistency). However, resources encrypted with data keys protected by the KMS key are not affected until the KMS key is used again, such to decrypt the data key. This issue affects AWS services, many of which use data keys to protect your resources. For details, see How unusable KMS keys affect data keys.

You can monitor the scheduling, cancellation, and deletion of the KMS key in your AWS CloudTrail logs.

Troubleshooting external key stores

The resolution for most problems with external key stores are indicated by the error message that AWS KMS displays with each exception, or by the connection error code that AWS KMS returns when an attempt to connect the external key store to its external key store proxy fails. However, some issues are a bit more complex.

When diagnosing an issue with an external key store, first locate the cause. This will narrow the range of remedies and make your troubleshooting more efficient.

- AWS KMS — The problem might be within AWS KMS, such as an incorrect value in your external key store configuration.
- External — The problem might originate outside of AWS KMS, including problems with the configuration or operation of the external key store proxy, external key manager, external keys, or VPC endpoint service.
- Networking — It might be a problem with connectivity or networking, such as a problem with your proxy endpoint, port, or your private DNS name or domain.

> **Note**
>
> When management operations on external key stores fail, they generate several different exceptions. But AWS KMS cryptographic operations return `KMSInvalidStateException` for all failures related to the external configuration or connection state of the external key store. To identify the problem, use the accompanying error message text.
> The ConnectCustomKeyStore operation succeeds quickly before the connection process is complete. To determine whether the connection process is successful, view the connection state of the external key store. If the connection process fails, AWS KMS returns a connection error code that explains the cause and suggests a remedy.

Topics

Troubleshooting tools for external key stores

AWS KMS provides several tools to help you identify and resolve problems with your external key store and its keys. Use these tools in conjunction with the tools provided with your external key store proxy and external key manager.

> **Note**
>
> Your external key store proxy and external key manager might provide easier methods of creating and maintaining your external key store and its KMS keys. For details, see the documentation for your external tools.

AWS KMS exceptions and error messages

AWS KMS provides a detailed error message about any problem it encounters. You can find additional information about AWS KMS exceptions in the *AWS Key Management Service API Reference* and AWS SDKs. Even if you are using the AWS KMS console, you might find these references to be helpful. For example, see the Errors list for the `CreateCustomKeyStores` operation.

To optimize the performance of your external key store proxy, AWS KMS returns exceptions based on your proxy's reliability within a given aggregation period of 5 minutes. In the event of a 500 Internal Server Error, 503 Service Unavailable, or connection timeout, a proxy with high reliability returns `KMSInternalException` and triggers an automatic retry to ensure that requests eventually succeed. However, a proxy with low reliability returns `KMSInvalidStateException`. For more information, see Monitoring an external key store.

If the problem surfaces in a different AWS service, such as when you use a KMS key in your external key store to protect a resource in another AWS service, the AWS service might provide additional information to help you identify the problem. If the AWS service doesn't provide the message, you can view the error message in the CloudTrail logs that record the use of your KMS key.

CloudTrail logs

Every AWS KMS API operation, including actions in the AWS KMS console, is recorded in AWS CloudTrail logs. AWS KMS records a log entry for successful and failed operations. For failed operations, the log entry includes the AWS KMS exception name (`errorCode`) and the error message (`errorMessage`). You can use this information to help you identify and resolve the error. For an example, see Decrypt failure with a KMS key in an external key store.

The log entry also includes the request ID. If the request reached your external key store proxy, you can use the request ID in the log entry to find the corresponding request in your proxy logs, if your proxy provides them.

CloudWatch metrics

AWS KMS records detailed Amazon CloudWatch metrics about the operation and performance of your external key store, including latency, throttling, proxy errors, external key manager status, the number of days until your TLS certificate expires, and the reported age of your proxy authentication credentials. You can use these metrics to develop data models for the operation of your external key store and CloudWatch alarms that alert you to impending problems before they occur.

⚠ **Important**

AWS KMS recommends that you create CloudWatch alarms to monitor the external key store metrics. These alarms will alert you to early signs of problems before they develop.

Monitoring graphs

AWS KMS displays graphs of the external key store CloudWatch metrics on the detail page for each external key store in the AWS KMS console. You can use the data in the graphs to help locate the source of errors, detect impending problems, establish baselines, and refine your CloudWatch alarm thresholds. For details about interpreting the monitoring graphs and using their data, see Monitoring an external key store.

Displays of external key stores and KMS keys

AWS KMS displays detailed information about your external key stores and the KMS keys in the external key store in the AWS KMS console, and in the response to the DescribeCustomKeyStores and DescribeKey operations. These displays include special fields for external key stores and KMS keys with information that you can use for troubleshooting, such as the connection state of the external key store and the ID of the external key that is associated with the KMS key. For details, see Viewing an external key store and Viewing KMS keys in an external key store.

XKS Proxy Test Client

AWS KMS provides an open source test client that verifies that your external key store proxy conforms to the AWS KMS External Key Store Proxy API Specification. You can use this test client to identify and resolve problems with your external key store proxy.

Configuration errors

When you create an external key store, you specify property values that comprise the *configuration* of your external key store, such as the proxy authentication credential, proxy URI endpoint, proxy URI path, and VPC endpoint service name. When AWS KMS detects an error in a property value, the operation fails and returns an error that indicates the faulty value.

Many configuration issues can be resolved by fixing the incorrect value. You can fix an invalid proxy URI path or proxy authentication credential without disconnecting the external key store. For definitions of these values, including uniqueness requirements, see Assemble the prerequisites. For instructions about updating these values, see Editing external key store properties.

To avoid errors with your proxy URI path and proxy authentication credential values, when creating or updating your external key store, upload a proxy configuration file to the AWS KMS console. This is a JSON-based file with proxy URI path and proxy authentication credential values that is provided by your external key store proxy or external key manager. You can't use a proxy configuration file with AWS KMS API operations, but you can use the values in the file to help you provide parameter values for your API requests that match the values in your proxy.

General configuration errors

Exceptions: `CustomKeyStoreInvalidStateException` (`CreateKey`), `KMSInvalidStateException` (cryptographic operations), `XksProxyInvalidConfigurationException` (management operations, except for `CreateKey`)

Connection error codes: `XKS_PROXY_INVALID_CONFIGURATION`, `XKS_PROXY_INVALID_TLS_CONFIGURATION`

For external key stores with public endpoint connectivity, AWS KMS tests the property values when you create and update the external key store. For external key stores with VPC endpoint service connectivity, AWS KMS tests the property values when you connect and update the external key store.

> **Note**
>
> The `ConnectCustomKeyStore` operation, which is asynchronous, might succeed even though the attempt to connect the external key store to its external key store proxy fails. In that case, there is no exception, but the connection state of the external key store is Failed, and a connection error code explains the error message. For more information, see External key store connection errors.

If AWS KMS detects an error in a property value, the operation fails and returns `XksProxyInvalidConfigurationException` with one of the following error messages.

```
The external key store proxy rejected the request because of an invalid URI path. Verify the URI path for your external key store and update if necessary.
```

- The proxy URI path is the base path for AWS KMS requests to the proxy APIs. If this path is incorrect, all requests to the proxy fail. To view the current proxy URI path for your external key store, use the AWS KMS console or the `DescribeCustomKeyStores` operation. To find the correct proxy URI path, see your external key store proxy documentation. For help correcting your proxy URI path value, see Editing external key store properties.
- The proxy URI path for your external key store proxy can change with updates to your external key store proxy or external key manager. For information about these changes, see the documentation for your external key store proxy or external key manager.

```
XKS_PROXY_INVALID_TLS_CONFIGURATION
AWS KMS cannot establish a TLS connection to the external key store proxy. Verify the TLS
configuration, including its certificate.
```

- All external key store proxies require a TLS certificate. The TLS certificate must be issued by a public certificate authority (CA) that is supported for external key stores. For list of supported CAs, see Trusted Certificate Authorities in the AWS KMS External Key Store Proxy API Specification.
- For public endpoint connectivity, the subject common name (CN) on the TLS certificate must match the domain name in the proxy URI endpoint for the external key store proxy. For example, if the public endpoint is https://myproxy.xks.example.com, the TLS, the CN on the TLS certificate must be `myproxy.xks.example.com` or `*.xks.example.com`.
- For VPC endpoint service connectivity, the subject common name (CN) on the TLS certificate must match the private DNS name for your VPC endpoint service. For example, if the private DNS name is myproxy-private.xks.example.com, the CN on the TLS certificate must be `myproxy-private.xks.example.com` or `*.xks.example.com`.
- The TLS certificate cannot be expired. To get the expiration date of a TLS certificate, use SSL tools, such as OpenSSL. To monitor the expiration date of a TLS certificate associated with an external key store, use the XksProxyCertificateDaysToExpire CloudWatch metric. The number of days to your TLS certification expiration date also appears in the **Monitoring** section of the AWS KMS console.
- If you are using public endpoint connectivity, use SSL test tools to test your SSL configuration. TLS connection errors can result from incorrect certificate chaining.

VPC endpoint service connectivity configuration errors

Exceptions: `XksProxyVpcEndpointServiceNotFoundException`, `XksProxyVpcEndpointServiceInvalidConfigurationException`

In addition to general connectivity issues, you might encounter the following issues while creating, connecting, or updating an external key store with VPC endpoint service connectivity. AWS KMS tests the property values of an external key store with VPC endpoint service connectivity while creating, connecting, and updating the external key store. When management operations fail due to configuration errors, they generate the following exceptions:

```
XksProxyVpcEndpointServiceNotFoundException
```

The cause might be one of the following:

- An incorrect VPC endpoint service name. Verify that the VPC endpoint service name for the external key store is correct and matches the proxy URI endpoint value for the external key store. To find the VPC endpoint service name, use the Amazon VPC console or the DescribeVpcEndpointServices operation. To find the VPC endpoint service name and proxy URI endpoint of an existing external key store, use the AWS KMS console or the DescribeCustomKeyStores operation. For details, see Viewing an external key store.
- The VPC endpoint service might be in a different AWS Region than the external key store. Verify that the VPC endpoint service and external key store are in same Region. (The external name of the Region name, such as `us-east-1`, is part of the VPC endpoint service name, such as com.amazonaws.vpce.us-east-1.vpce-svc-example.) For a list of requirements for the VPC endpoint service for an external key store, see VPC endpoint service. You cannot move a VPC endpoint service or an external key store to a different Region. However, you can create a new external key store in the same Region as the VPC endpoint service. For details, see Configuring VPC endpoint service connectivity and Creating an external key store.
- AWS KMS is not an allowed principal for the VPC endpoint service. The **Allow principals** list for the VPC endpoint service must include the `cks.kms.<region>.amazonaws.com` value, such as `cks.kms.eu-west-3.amazonaws.com`. For instructions about adding this value, see Manage permissions in the *AWS PrivateLink Guide*.

```
XksProxyVpcEndpointServiceInvalidConfigurationException
```

This error occurs when the VPC endpoint service fails to meet one of the following requirements:

- The VPC requires at least two private subnets, each in a different Availability Zone. For help adding a subnet to your VPC, see Create a subnet in your VPC in the *Amazon VPC User Guide*.
- Your VPC endpoint service type must use a network load balancer, not a gateway load balancer.
- Acceptance must not be required for the VPC endpoint service (**Acceptance required** must be false.). If manual acceptance of each connection request is required, AWS KMS cannot use the

VPC endpoint service to connect to the external key store proxy. For details, see Accept or reject connection requests in the *AWS PrivateLink Guide*.

- The VPC endpoint service must have a private DNS name that is a subdomain of a public domain. For example, if the private DNS name is `https://myproxy-private.xks.example.com`, the `xks.example.com` or `example.com` domains must have a public DNS server. To view or change the private DNS name for your VPC endpoint service, see Manage DNS names for VPC endpoint services in the *AWS PrivateLink Guide*.
- The **Domain verification status** of the domain for your private DNS name must be `verified`. To view and update the verification status of the private DNS name domain, see Verifying your private DNS name domain. It might take a few minutes for the updated verification status to appear after you've added the required text record.

> **Note**
>
> A private DNS domain can be verified only if it is the subdomain of a public domain. Otherwise, the verification status of the private DNS domain does not change, even after you add the required TXT record.

- The private DNS name of the VPC endpoint service must match the proxy URI endpoint value for the external key store. For an external key store with VPC endpoint service connectivity, the proxy URI endpoint must be `https://` followed by the private DNS name of the VPC endpoint service. To view the proxy URI endpoint value, see Viewing an external key store. To change the proxy URI endpoint value, see Editing external key store properties.

External key store connection errors

The process of connecting an external key store to its external key store proxy takes about five minutes to complete. Unless it fails quickly, the `ConnectCustomKeyStore` operation returns an HTTP 200 response and a JSON object with no properties. However, this initial response does not indicate that the connection was successful. To determine whether the external key store is connected, see its connection state. If the connection fails, the connection state of the external key store changes to `FAILED` and AWS KMS returns a connection error code that explains the cause of the failure.

> **Note**
>
> When the connection state of a custom key store is `FAILED`, you must disconnect the custom key store before attempting to reconnect it. You cannot connect a custom key store with a `FAILED` connection status.

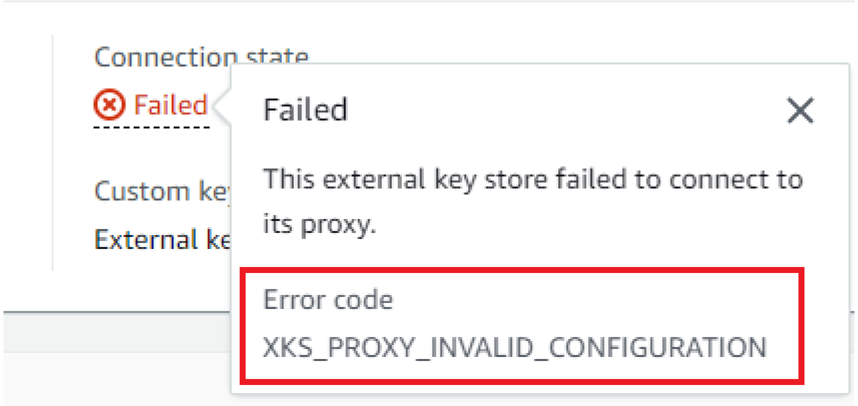
To view the connection state of an external key store:

- In the DescribeCustomKeyStores response, view the value of the `ConnectionState` element.
- In the AWS KMS console, the **Connection state** appears in the external key store table. Also, on the detail page for each external key store, the **Connection state** appears in the **General configuration** section.

When the connection state is `FAILED`, the connection error code helps to explains the error.

To view the connection error code:

- In the DescribeCustomKeyStores response, view the value of the `ConnectionErrorCode` element. This element appears in the `DescribeCustomKeyStores` response only when the `ConnectionState` is `FAILED`.
- To view the connection error code in the AWS KMS console, on detail page for the external key store and hover over the **Failed** value.



Connection state

Failed

Custom ke

External ke

Failed

This external key store failed to connect to its proxy.

Error code

XKS_PROXY_INVALID_CONFIGURATION

Connection error codes for external key stores

The following connection error codes apply to external key stores

`INTERNAL_ERROR`

AWS KMS could not complete the request due to an internal error. Retry the request. For `ConnectCustomKeyStore` requests, disconnect the custom key store before trying to connect again.

`INVALID_CREDENTIALS`

One or both of the `XksProxyAuthenticationCredential` values is not valid on the specified external key store proxy.

`NETWORK_ERRORS`

Network errors are preventing AWS KMS from connecting the custom key store to its backing key store.

`XKS_PROXY_ACCESS_DENIED`

AWS KMS requests are denied access to the external key store proxy. If the external key store proxy has authorization rules, verify that they permit AWS KMS to communicate with the proxy on your behalf.

`XKS_PROXY_INVALID_CONFIGURATION`

A configuration error is preventing the external key store from connecting to its proxy. Verify the value of the `XksProxyUriPath`.

`XKS_PROXY_INVALID_RESPONSE`

AWS KMS cannot interpret the response from the external key store proxy. If you see this connection error code repeatedly, notify your external key store proxy vendor.

`XKS_PROXY_INVALID_TLS_CONFIGURATION`

AWS KMS cannot connect to the external key store proxy because the TLS configuration is invalid. Verify that the external key store proxy supports TLS 1.2 or 1.3. Also, verify that the TLS certificate is not expired, that it matches the hostname in the `XksProxyUriEndpoint` value, and that it is signed by a trusted certificate authority included in the Trusted Certificate Authorities list.

`XKS_PROXY_NOT_REACHABLE`

AWS KMS can't communicate with your external key store proxy. Verify that the `XksProxyUriEndpoint` and `XksProxyUriPath` are correct. Use the tools for your external key store proxy to verify that the proxy is active and available on its network. Also, verify that your external key manager instances are operating properly. Connection attempts fail with this connection error code if the proxy reports that all external key manager instances are unavailable.

XKS_PROXY_TIMED_OUT

AWS KMS can connect to the external key store proxy, but the proxy does not respond to AWS KMS in the time allotted. If you see this connection error code repeatedly, notify your external key store proxy vendor.

XKS_VPC_ENDPOINT_SERVICE_INVALID_CONFIGURATION

The Amazon VPC endpoint service configuration doesn't conform to the requirements for an AWS KMS external key store.

- The VPC endpoint service must be an endpoint service for interface endpoints in the caller's AWS account.
- It must have a network load balancer (NLB) connected to at least two subnets, each in a different Availability Zone.
- The Allow principals list must include the AWS KMS service principal for the Region, cks.kms.<region>.amazonaws.com, such as cks.kms.us-east-1.amazonaws.com.
- It must *not* require acceptance of connection requests.
- It must have a private DNS name. The private DNS name for an external key store with VPC_ENDPOINT_SERVICE connectivity must be unique in its AWS Region.
- The domain of the private DNS name must have a verification status of verified.
- The TLS certificate specifies the private DNS hostname at which the endpoint is reachable.

XKS_VPC_ENDPOINT_SERVICE_NOT_FOUND

AWS KMS can't find the VPC endpoint service that it uses to communicate with the external key store proxy. Verify that the XksProxyVpcEndpointServiceName is correct and the AWS KMS service principal has service consumer permissions on the Amazon VPC endpoint service.

Latency and timeout errors

Exceptions: CustomKeyStoreInvalidStateException (CreateKey), KMSInvalidStateException (cryptographic operations), XksProxyUriUnreachableException (management operations)

Connection error codes: XKS_PROXY_NOT_REACHABLE, XKS_PROXY_TIMED_OUT

When AWS KMS can't contact the proxy within the 250 millisecond timeout interval, it returns an exception. `CreateCustomKeyStore` and `UpdateCustomKeyStore` return `XksProxyUriUnreachableException`. Cryptographic operations return the standard `KMSInvalidStateException` with an error message that describes the problem. If `ConnectCustomKeyStore` fails, AWS KMS returns a connection error code that describes the problem.

Timeout errors might be transient issues that can be resolved by retrying the request. If the problem persists, verify that your external key store proxy is active and is connected to the network, and that its proxy URI endpoint, proxy URI path, and VPC endpoint service name (if any) are correct in your external key store. Also, verify that your external key manager is close to the AWS Region for your external key store. If you need to update any of these values, see Editing external key store properties.

To track latency patterns, use the `XksProxyLatency` CloudWatch metric and the **Average latency** graph (based on that metric) in the **Monitoring** section of the AWS KMS console. Your external key store proxy might also generate logs and metrics that track latency and timeouts.

```
XksProxyUriUnreachableException
AWS KMS cannot communicate with the external key store proxy. This might be a transient
 network issue. If you see this error repeatedly, verify that your external key store proxy is active
and is connected to the network, and that its endpoint URI is correct in your external key store.
```

- The external key store proxy didn't respond to an AWS KMS proxy API request within the 250 millisecond timeout interval. This might indicate a transient network problem or an operational or performance problem with the proxy. If retrying doesn't solve the problem, notify your external key store proxy administrator.

Latency and timeout errors often manifest as connection failures. When the ConnectCustomKeyStore operation fails, the *connection state* of the external key store changes to `FAILED` and AWS KMS returns a *connection error code* that explains the error. For a list of connection error codes and suggestions for resolving the errors, see Connection error codes for external key stores. The connection codes lists for **All custom key stores** and **External key stores** apply to external key stores. The following connection errors are related to latency and timeouts.

```
XKS_PROXY_NOT_REACHABLE
```

-or-

`CustomKeyStoreInvalidStateException` , `KMSInvalidStateException` , `XksProxyUriUnreachableException`

AWS KMS cannot communicate with the external key store proxy. Verify that your external key store proxy is active and is connected to the network, and that its URI path and endpoint URI or VPC service name are correct in your external key store.

This error might occur for the following reasons:

- The external key store proxy is not active and or not connected to the network.
- There is an error in the proxy URI endpoint, proxy URI path, or VPC endpoint service name (if applicable) values in the external key store configuration. To view the external key store configuration, use the DescribeCustomKeyStores operation or view the detail page for the external key store in the AWS KMS console.
- There might be a network configuration error, such as a port error, on the network path between AWS KMS and the external key store proxy. AWS KMS communicates with the external key store proxy on port 443. This value is not configurable.
- When the external key store proxy reports (in a GetHealthStatus response) that all external key manager instances are `UNAVAILABLE`, the ConnectCustomKeyStore operation fails with a `ConnectionErrorCode` of `XKS_PROXY_NOT_REACHABLE`. For help, see your external key manager documentation.
- This error can result from a long physical distance between the external key manager and the AWS Region with the external key store. The ping latency (network round-trip time (RTT)) between the AWS Region and the external key manager must be no more than 35 milliseconds. You might have to create an external key store in an AWS Region that is closer to the external key manager, or move the external key manager to a data center that is closer to the AWS Region.

`XKS_PROXY_TIMED_OUT`

-or-

`CustomKeyStoreInvalidStateException` , `KMSInvalidStateException` , `XksProxyUriUnreachableException`

AWS KMS rejected the request because the external key store proxy did not respond in time. Retry the request. If you see this error repeatedly, report it to your external key store proxy administrator.

This error might occur for the following reasons:

- This error can result from a long physical distance between the external key manager and the external key store proxy. If possible, move the external key store proxy closer to the external key manager.
- Timeout errors can occur when the proxy is not designed to handle the volume and frequency of requests from AWS KMS. If your CloudWatch metrics indicate a persistent problem, notify your external key store proxy administrator.
- Timeout errors can occur when the connection between the external key manager and the Amazon VPC for the external key store is not operating properly. If you are using AWS Direct Connect, verify that your VPC and external key manager can communicate effectively. For help resolving any issues, see Troubleshooting AWS Direct Connect in the AWS Direct Connect User Guide.

`XKS_PROXY_TIMED_OUT`

-or-

`CustomKeyStoreInvalidStateException` , `KMSInvalidStateException` , `XksProxyUriUnreachableException`

The external key store proxy did not respond to the request in the time allotted. Retry the request. If you see this error repeatedly, report it to your external key store proxy administrator.

- This error can result from a long physical distance between the external key manager and the external key store proxy. If possible, move the external key store proxy closer to the external key manager.

Authentication credential errors

Exceptions: `CustomKeyStoreInvalidStateException` (`CreateKey`), `KMSInvalidStateException` (cryptographic operations), `XksProxyIncorrectAuthenticationCredentialException` (management operations other than `CreateKey`)

You establish and maintain an authentication credential for AWS KMS on your external key store proxy. Then you tell AWS KMS the credential values when you create an external key store. To change the authentication credential, make the change on your external key store proxy. Then update the credential for your external key store. If your proxy rotates the credential, you must update the credential for your external key store.

If the external key store proxy won't authenticate a request signed with the proxy authentication credential for your external key store, the effect depends on the request:

- `CreateCustomKeyStore` and `UpdateCustomKeyStore` fail with an `XksProxyIncorrectAuthenticationCredentialException`.
- `ConnectCustomKeyStore` succeeds, but the connection fails. The connection state is `FAILED` and the connection error code is `INVALID_CREDENTIALS`. For details, see External key store connection errors.
- Cryptographic operations return `KMSInvalidStateException` for all external configuration errors and connection state errors in an external key store. The accompanying error message describes the problem.

The external key store proxy rejected the request because it could not authenticate AWS KMS. Verify the credentials for your external key store and update if necessary.

This error might occur for the following reasons:

- The access key ID or the secret access key for the external key store doesn't match the values established on the external key store proxy.

  To fix this error, update the proxy authentication credential for your external key store. You can make this change without disconnecting your external key store.
- A reverse proxy between AWS KMS and the external key store proxy could be manipulating HTTP headers in a manner that invalidates the SigV4 signatures. To fix this error, notify your proxy administrator.

Key state errors

Exceptions: `KMSInvalidStateException`

`KMSInvalidStateException` is used for two distinct purposes for KMS keys in custom key stores.

- When a management operation, such as `CancelKeyDeletion`, fails and returns this exception, it indicates that the key state of the KMS key is not compatible with the operation.
- When a cryptographic operation on a KMS key in a custom key store fails with `KMSInvalidStateException`, it can indicate a problem with the key state of the KMS key. But AWS KMS cryptographic operation return `KMSInvalidStateException` for all external configuration errors and connection state errors in an external key store. To identify the problem, use the error message that accompanies the exception.

To find the required key state for an AWS KMS API operations, see Key states of AWS KMS keys. To find the key state of a KMS key, on the **Customer managed keys** page, view the **Status** field of the KMS key. Or, use the DescribeKey operation and view the `KeyState` element in the response. For details, see Viewing keys.

> **Note**
>
> The key state of a KMS key in an external key store does not indicate anything about the status of its associated external key. For information about the external key status, use your external key manager and external key store proxy tools.
> The `CustomKeyStoreInvalidStateException` refers to the connection state of the external key store, not the key state of a KMS key.

A cryptographic operation on a KMS key in a custom store might fail because the key state of the KMS key is `Unavailable` or `PendingDeletion`. (Disabled keys return `DisabledException`.)

- A KMS key has a `Disabled` key state only when you intentionally disable the KMS key in the AWS KMS console or by using the DisableKey operation. While a KMS key is disabled, you can view and manage the key, but you cannot use it in cryptographic operations. To fix this problem, enable the key. For details, see Enabling and disabling keys.
- A KMS key has an `Unavailable` key state when the external key store is disconnected from its external key store proxy. To fix an unavailable KMS key, reconnect the external key store. After

the external key store is reconnected, the key state of the KMS keys in the external key store is automatically restored to its previous state, such as `Enabled` or `Disabled`.

A KMS key has a `PendingDeletion` key state when it has been scheduled for deletion and is in its waiting period. A key state error on a KMS key that is pending deletion indicates that the key should not be deleted, either because it's being used for encryption, or it is required for decryption. To re-enable the KMS key, cancel the scheduled deletion, and then enable the key. For details, see Scheduling and canceling key deletion.

Decryption errors

Exceptions: `KMSInvalidStateException`

When a Decrypt operation with a KMS key in an external key store fails, AWS KMS returns the standard `KMSInvalidStateException` that cryptographic operations use for all external configuration errors and connection state errors on an external key store. The error message indicates the problem.

To decrypt a ciphertext that was encrypted using double encryption, the external key manager first uses the external key to decrypt the outer layer of ciphertext. Then AWS KMS uses the AWS KMS key material in the KMS key to decrypt the inner layer of ciphertext. An invalid or corrupt ciphertext can be rejected by the external key manager or AWS KMS.

The following error messages accompany the `KMSInvalidStateException` when decryption fails. It indicates a problem with the ciphertext or the optional encryption context in the request.

```
The external key store proxy rejected the request because the specified ciphertext or additional
authenticated data is corrupted, missing, or otherwise invalid.
```

- When the external key store proxy or external key manager report that a ciphertext or its encryption context is invalid, it typically indicates a problem with the ciphertext or encryption context in the `Decrypt` request sent to AWS KMS. For `Decrypt` operations, AWS KMS sends the proxy the same ciphertext and encryption context it receives in the `Decrypt` request.

  This error might be caused by a networking problem in transit, such as a flipped bit. Retry the `Decrypt` request. If the problem persists, verify that the ciphertext was not altered or corrupted. Also, verify that the encryption context in the `Decrypt` request to AWS KMS matches the encryption context in the request that encrypted the data.

```
The ciphertext that the external key store proxy submitted for decryption, or the encryption
context, is corrupted, missing, or otherwise invalid.
```

- When AWS KMS rejects the ciphertext that it received from the proxy, it indicates that the external key manager or proxy returned an invalid or corrupt ciphertext to AWS KMS.

  This error might be caused by a networking problem in transit, such as a flipped bit. Retry the `Decrypt` request. If the problem persists, verify that the external key manager is operating properly, and that the external key store proxy does not alter the ciphertext that it receives from the external key manager before it returns it to AWS KMS.

External key errors

An external key is a cryptographic key in the external key manager that serves as the external key material for a KMS key. AWS KMS cannot directly access the external key. It must ask the external key manager (via the external key store proxy) to use the external key to encrypt data or decrypt a ciphertext.

You specify the ID of the external key in its external key manager when you create a KMS key in your external key store. You cannot change the external key ID after the KMS key is created. To prevent problems with the KMS key, the `CreateKey` operation asks the external key store proxy to verify the ID and configuration of the external key. If the external key doesn't fulfill the requirements for use with a KMS key, the `CreateKey` operation fails with an exception and error message that identifies the problem.

However, issues can occur after the KMS key is created. If a cryptographic operation fails because of a problem with the external key, the operation fails and returns an `KMSInvalidStateException` with an error message that indicates the problem.

CreateKey errors for the external key

Exceptions: `XksKeyAlreadyInUseException`, `XksKeyNotFoundException`, `XksKeyInvalidConfigurationException`

The CreateKey operation attempts to verify the ID and properties of the external key that you provide in the **External key ID** (console) or `XksKeyId` (API) parameter. This practice is designed to detect errors early before you try to use the external key with the KMS key.

External key in use

Each KMS key in an external key store must use a different external key. When CreateKey recognizes that the external key ID (XksKeyId) for a KMS key is not unique in the external key store, it fails with an XksKeyAlreadyInUseException.

If you use multiple IDs for the same external key, CreateKey won't recognize the duplicate. However, KMS keys with the same external key are not interoperable because they have different AWS KMS key material and metadata.

External key not found

When the external key store proxy reports that it cannot find the external key using the external key ID (XksKeyId) for the KMS key, the CreateKey operation fails and returns XksKeyNotFoundException with the following error message.

```
The external key store proxy rejected the request because it could not find the external key.
```

This error might occur for the following reasons:

- The ID of the external key (XksKeyId) for the KMS key might be invalid. To find the ID for your external key proxy uses to identify the external key, see your external key store proxy or external key manager documentation.
- The external key might have been deleted from your external key manager. To investigate, use your external key manager tools. If the external key is permanently deleted, use a different external key with the KMS key. For a list or requirements for the external key, see Requirements for a KMS key in an external key store.

External key requirements not met

When the external key store proxy reports that the external key does not fulfill the requirements for use with a KMS key, the CreateKey operation fails and returns XksKeyInvalidConfigurationException with one of the following error messages.

```
The key spec of the external key must be AES_256. The key spec of specified external key is
<key-spec> .
```

- The external key must be a 256-bit symmetric encryption key with a key spec of AES_256. If the specified external key is a different type, specify the ID of an external key that fulfills this requirement.

```
The status of the external key must be ENABLED. The status of specified external key is
<status>.
```

- The external key must be enabled in the external key manager. If the specified external key is not enabled, use your external key manager tools to enable it, or specify an enabled external key.

```
The key usage of the external key must include ENCRYPT and DECRYPT. The key use of specified
external key is <key-usage >.
```

- The external key must be configured for encryption and decryption in the external key manager. If the specified external key does not include these operations, use your external key manager tools to change the operations, or specify a different external key.

Cryptographic operation errors for the external key

Exceptions: `KMSInvalidStateException`

When the external key store proxy cannot find the external key associated with the KMS key, or the external key doesn't fulfill the requirements for use with a KMS key, the cryptographic operation fails.

External key issues that are detected during a cryptographic operation are more difficult to resolve than external key issues detected before creating the KMS key. You cannot change the external key ID after the KMS key is created. If the KMS key has not yet encrypted any data, you can delete the KMS key and create a new one with a different external key ID. However, ciphertext generated with the KMS key cannot be decrypted by any other KMS key, even one with the same external key, because keys will have different key metadata and different AWS KMS key material. Instead, to the extent possible, use your external key manager tools to resolve the problem with the external key.

When the external key store proxy reports a problem with the external key, cryptographic operations return `KMSInvalidStateException` with an error message that identifies the problem.

External key not found

When the external key store proxy reports that it cannot find the external key using the external key ID (XksKeyId) for the KMS key, cryptographic operations return a `KMSInvalidStateException` with the following error message.

```
The external key store proxy rejected the request because it could not find the external key.
```

This error might occur for the following reasons:

- The ID of the external key (`XksKeyId`) for the KMS key is no longer valid.

  To find the external key ID associated with your KMS key, view the details of the KMS key. To find the ID that your external key proxy uses to identify the external key, see your external key store proxy or external key manager documentation.

  AWS KMS verifies the external key ID when it creates a KMS key in an external key store. However, the ID might become invalid, especially if the external key ID value is an alias or mutable name. You cannot change the external key ID associated with an existing KMS key. To decrypt any ciphertext encrypted under the KMS key, you must re-associate the external key with the existing external key ID.

  If you have not yet used the KMS key to encrypt data, you can create a new KMS key with a valid external key ID. However, if you have generated ciphertext with the KMS key, you cannot use any other KMS key to decrypt the ciphertext, even if uses the same external key.
- The external key might have been deleted from your external key manager. To investigate, use your external key manager tools. If possible, try to recover the key material from a copy or backup of your external key manager. If the external key is permanently deleted, any ciphertext encrypted under the associated KMS key is unrecoverable.

External key configuration errors

When the external key store proxy reports that the external key doesn't fulfill the requirements for use with a KMS key, the cryptographic operation returns `KMSInvalidStateException` with the one of the following error messages.

```
The external key store proxy rejected the request because the external key does not support the requested operation.
```

- The external key must support both encryption and decryption. If the key usage does not include encryption and decryption, use your external key manager tools to change the key usage.

```
The external key store proxy rejected the request because the external key is not enabled in the external key manager.
```

- The external key must be enabled and available for use in the external key manager. If the status of the external key is not `Enabled`, use your external key manager tools to enable it.

Proxy issues

Exceptions:

`CustomKeyStoreInvalidStateException` (`CreateKey`), `KMSInvalidStateException` (cryptographic operations), `UnsupportedOperationException`, `XksProxyUriUnreachableException`, `XksProxyInvalidResponseException` (management operations other than `CreateKey`)

The external key store proxy mediates all communication between AWS KMS and the external key manager. It translates generic AWS KMS requests into a format that your external key manager can understand. If the external key store proxy doesn't conform to the AWS KMS External Key Store Proxy API Specification, or if isn't operating properly, or can't communicate with AWS KMS, you won't be able to create or use KMS keys in your external key store.

While many errors mention the external key store proxy because of its critical role in the external key store architecture, those problem might originate in the external key manager or external key.

The issues in this section relate to problems with the design or operation of the external key store proxy. Resolving these issues might require a change to the proxy software. Consult your proxy administrator. To help diagnose proxy issues, AWS KMS provides XKS Proxy Text Client, an open source test client that verifies that your external key store proxy conforms to the AWS KMS External Key Store Proxy API Specification.

`CustomKeyStoreInvalidStateException` , `KMSInvalidStateException` or `XksProxyUriUnreachableException`

The external key store proxy is in an unhealthy state. If you see this message repeatedly, notify your external key store proxy administrator.

- This error can indicate an operational problem or software error in the external key store proxy. You can find CloudTrail log entries for the AWS KMS API operation that generated each error. This error might be resolved by retrying the operation. However, if it persists, notify your external key store proxy administrator.
- When the external key store proxy reports (in a GetHealthStatus response) that all external key manager instances are `UNAVAILABLE`, attempts to create or update an external key store fail with this exception. If this error persists, consult your external key manager documentation.

`CustomKeyStoreInvalidStateException` , `KMSInvalidStateException` or `XksProxyInvalidResponseException`

AWS KMS cannot interpret the response from the external key store proxy. If you see this error repeatedly, consult your external key store proxy administrator.

- AWS KMS operations generate this exception when the proxy returns an undefined response that AWS KMS cannot parse or interpret. This error might occur occasionally due to temporarily external issues or sporadic network errors. However, if it persists, it might indicate that the external key store proxy doesn't conform to the AWS KMS External Key Store Proxy API Specification. Notify your external key store administrator or vendor.

`CustomKeyStoreInvalidStateException` , `KMSInvalidStateException` or `UnsupportedOperationException`

The external key store proxy rejected the request because it does not support the requested cryptographic operation.

- The external key store proxy should support all proxy APIs defined in the AWS KMS External Key Store Proxy API Specification. This error indicates that the proxy does not support the operation that is related to the request. Notify your external key store administrator or vendor.

Proxy authorization issues

Exceptions: `CustomKeyStoreInvalidStateException`, `KMSInvalidStateException`

Some external key store proxies implement authorization requirements for the use of its external keys. An external key store proxy is permitted, but not required, to design and implement an authorization scheme that allows particular users to request particular operations under certain conditions. For example, a proxy might allow a user to encrypt with a particular external key, but not to decrypt with it. For more information, see External key store proxy authorization (optional).

Proxy authorization is based on metadata that AWS KMS includes in its requests to the proxy. The `awsSourceVpc` and `awsSourceVpce` fields are included in the metadata only when the request is from a VPC endpoint and only when the caller is in the same account as the KMS key.

```
"requestMetadata": {
    "awsPrincipalArn": string,
    "awsSourceVpc": string, // optional
    "awsSourceVpce": string, // optional
    "kmsKeyArn": string,
    "kmsOperation": string,
    "kmsRequestId": string,
    "kmsViaService": string // optional
}
```

When the proxy rejects a request due to an authorization failure, the related AWS KMS operation fails. `CreateKey` returns `CustomKeyStoreInvalidStateException`. AWS KMS cryptographic operations return `KMSInvalidStateException`. Both use the following error message:

> The external key store proxy denied access to the operation. Verify that the user and the external key are both authorized for this operation, and try the request again.

- To resolve the error, use your external key manager or external key store proxy tools to determine why authorization failed. Then, update the procedure that caused the unauthorized request or use your external key store proxy tools to update the authorization policy. You cannot resolve this error in AWS KMS.

Key type reference

AWS KMS supports different features for different types of KMS keys. For example, you can only use symmetric encryption KMS keys to generate symmetric data keys and asymmetric data key pairs. Also, importing key material and automatic key rotation are supported only for symmetric encryption KMS keys, and you can create only symmetric encryption KMS keys in a custom key store.

This reference includes two tables.

- The Key type table lists the AWS KMS operations that are valid for symmetric encryption KMS keys, asymmetric KMS keys, and HMAC KMS keys.
- The Special features table lists the AWS KMS operations that are valid for multi-Region KMS keys, KMS keys with imported key material, and KMS keys in custom key stores.

Key type table

You might need to scroll horizontally or vertically to see all of the data in this table.

AWS KMS API operation	Symmetr encryption KMS keys	HMAC KMS keys	Asymmet c KMS keys (ENCRYPT DECRYPT	Asymmetric KMS keys (SIGN_VERIFY)	Asymmetric KMS keys (KEY_AGREEMENT)
CancelKeyDeletion	✓	✓	✓	✓	✓
CreateAlias	✓	✓	✓	✓	✓
CreateGrant	✓	✓	✓	✓	✓
CreateKey	✓	✓	✓	✓	✓

AWS KMS API operation	Symmetr encryption KMS keys	HMAC KMS keys	Asymmetc KMS keys (ENCRYPT DECRYPT	Asymmetric KMS keys (SIGN_VERIFY)	Asymmetric KMS keys (KEY_AGREEMENT)
Decrypt	✓	⊗	✓	⊗	⊗
DeleteAlias	✓	✓	✓	✓	✓
DeleteImportedKeyMaterial Valid only on KMS keys with imported key material (`Origin` is EXTERNAL).	✓	✓	✓	✓	✓
DeriveSharedSecret	⊗	⊗	⊗	⊗	✓
DescribeKey	✓	✓	✓	✓	✓
DisableKey	✓	✓	✓	✓	✓

AWS KMS API operation	Symmetr encryption KMS keys	HMAC KMS keys	Asymmetc KMS keys (ENCRYPT DECRYPT	Asymmetric KMS keys (SIGN_VERIFY)	Asymmetric KMS keys (KEY_AGREEMENT)
DisableKeyRotation	✓ Valid only on KMS keys with AWS KMS key material (`Origin` is `AWS_KMS`)	⊗	⊗	⊗	⊗
EnableKey	✓	✓	✓	✓	✓

AWS KMS API operation	Symmetr encryption KMS keys	HMAC KMS keys	Asymmet c KMS keys (ENCRYPT DECRYPT	Asymmetric KMS keys (SIGN_VERIFY)	Asymmetric KMS keys (KEY_AGREEMENT)
EnableKeyRotation	✓ Valid only on KMS keys with AWS KMS key material (`Origin` is `AWS_KMS`)	⊗	⊗	⊗	⊗
Encrypt	✓	⊗	✓	⊗	⊗
GenerateDataKey	✓	⊗	⊗	⊗	⊗
GenerateDataKeyPair Generates an asymmetric data key pair that is protected by a symmetric encryption KMS key.	✓ Not valid on KMS keys in custom key stores.	⊗	⊗	⊗	⊗

AWS KMS API operation	Symmetr encryption KMS keys	HMAC KMS keys	Asymmet c KMS keys (ENCRYPT DECRYPT	Asymmetric KMS keys (SIGN_VERIFY)	Asymmetric KMS keys (KEY_AGREEMENT)
GenerateDataKeyPairWithoutPlaintext Generates an asymmetric data key pair that is protected by a symmetric encryption KMS key.	✓ Not valid on KMS keys in custom key stores.	⊗	⊗	⊗	⊗
GenerateDataKeyWithoutPlaintext	✓	⊗	⊗	⊗	⊗
GenerateMac	⊗	✓	⊗	⊗	⊗
GetKeyPolicy	✓	✓	✓	✓	✓
GetKeyRotationStatus	✓	✓ (KeyRota onEnabl will always be false.)	✓ (KeyRota onEnabl will always be false.)	✓ (KeyRotationEnabled will always be false.)	✓ (KeyRotationEnabled will always be false.)

AWS KMS API operation	Symmetr encryption KMS keys	HMAC KMS keys	Asymmet c KMS keys (ENCRYPT DECRYPT	Asymmetric KMS keys (SIGN_VERIFY)	Asymmetric KMS keys (KEY_AGREEMENT)
GetParametersForImport Valid only on KMS keys with imported key material (`Origin` is `EXTERNAL`).	✓	✓	✓	✓	✓
GetPublicKey	⊗	⊗	✓	✓	✓
ImportKeyMaterial Valid only on KMS keys with imported key material (`Origin` is `EXTERNAL`).	✓	✓	✓	✓	✓
ListAliases	✓	✓	✓	✓	✓
ListGrants	✓	✓	✓	✓	✓
ListKeyPolicies	✓	✓	✓	✓	✓

AWS KMS API operation	Symmetr encryption KMS keys	HMAC KMS keys	Asymmet c KMS keys (ENCRYPT DECRYPT	Asymmetric KMS keys (SIGN_VERIFY)	Asymmetric KMS keys (KEY_AGREEMENT)
ListKeyRotations	✓	✓ (The `Rotatio` field will always be null or empty.)	✓ (The `Rotatio` field will always be null or empty.)	✓ (The `Rotations` field will always be null or empty.)	✓ (The `Rotations` field will always be null or empty.)
ListResourceTags	✓	✓	✓	✓	✓
ListRetirableGrants	✓	✓	✓	✓	✓
PutKeyPolicy	✓	✓	✓	✓	✓
ReEncrypt	✓	⊗	✓	⊗	⊗
ReplicateKey - Valid only on multi-Region keys	✓	✓	✓	✓	✓
RetireGrant	✓	✓	✓	✓	✓
RevokeGrant	✓	✓	✓	✓	✓

AWS KMS API operation	Symmetr encryption KMS keys	HMAC KMS keys	Asymmet c KMS keys (ENCRYPT DECRYPT	Asymmetric KMS keys (SIGN_VERIFY)	Asymmetric KMS keys (KEY_AGREEMENT)
RotateKeyOnDemand	✓ Valid only on KMS keys with AWS KMS key material (`Origin` is `AWS_KMS`)	⊗	⊗	⊗	⊗
ScheduleKeyDeletion	✓	✓	✓	✓	✓
Sign	⊗	⊗	⊗	✓	⊗
TagResource	✓	✓	✓	✓	✓
UntagResource	✓	✓	✓	✓	✓

AWS KMS API operation	Symmetr encryption KMS keys	HMAC KMS keys	Asymmet c KMS keys (ENCRYPT DECRYPT	Asymmetric KMS keys (SIGN_VERIFY)	Asymmetric KMS keys (KEY_AGREEMENT)
UpdateAlias The current KMS key and the new KMS key must be the same type (both symmetric or both asymmetric or both HMAC) and they must have the same key usage.	✓	✓	✓	✓	✓
UpdateKeyDescription	✓	✓	✓	✓	✓
UpdateReplicaRegion - Valid only on multi-Region keys	✓	✓	✓	✓	✓
Verify	⊗	⊗	⊗	✓	⊗
VerifyMac	⊗	✓	⊗	⊗	⊗

Special features table

This table shows the AWS KMS API operations that are supported on each type of special-purpose key.

While reading this table, be aware of the following interactions:

- Multi-Region keys:
  - Multi-Region keys can be symmetric encryption KMS keys, asymmetric KMS keys, HMAC KMS keys, and KMS keys with imported key material.
  - You cannot create multi-Region keys in a custom key store.
- Imported key material
  - You can import key material for symmetric encryption KMS keys, asymmetric KMS keys, and HMAC KMS keys.
  - You can create multi-Region keys with imported key material.
  - You cannot create keys with imported key material in a custom key store.
  - Automatic key rotation (`EnableKeyRotation`, `DisableKeyRotation`) is not supported for KMS keys with imported key material.
- Custom key stores
  - Custom key stores support only symmetric encryption KMS keys.
  - Symmetric operations on asymmetric key pairs (`GenerateDataKeyPair`, `GenerateDataKeyPairWithoutPlaintext`) are not supported on KMS keys in custom key stores.
  - Automatic key rotation (`EnableKeyRotation`, `DisableKeyRotation`) is not supported on KMS keys in custom key stores.
  - You cannot create multi-Region keys in custom key stores.

You might need to scroll horizontally or vertically to see all of the data in this table.

AWS KMS API operation	Multi-Region keys	Imported key material	KMS keys in a custom key store
CancelKeyDeletion	✓	✓	✓
CreateAlias	✓	✓	✓
CreateGrant	✓	✓	✓

AWS KMS API operation	Multi-Region keys	Imported key material	KMS keys in a custom key store
CreateKey You can use `CreateKey` to create a multi-Region primary key, a KMS key with imported key material, or a KMS key in a custom key store. To create a multi-Region replica key, use `ReplicateKey` .	✓	✓	✓
Decrypt	✓ Valid only when `KeyUsage` is `ENCRYPT_DECRYPT`	✓	✓
DeleteAlias	✓	✓	✓
DeleteImportedKeyMaterial	✓ Valid only for keys with imported key material (`Origin` is `EXTERNAL`)	✓	⊗
DescribeKey	✓	✓	✓
DisableKey	✓	✓	✓

AWS KMS API operation	Multi-Region keys	Imported key material	KMS keys in a custom key store
DisableKeyRotation	✓ Valid only on symmetric encryption keys with AWS KMS key material (`Origin` is `AWS_KMS`).	⊗	⊗
EnableKey	✓ Valid only on symmetric encryption KMS keys	✓	✓
EnableKeyRotation	✓ Valid only on symmetric encryption keys with AWS KMS key material (`Origin` is `AWS_KMS`).	⊗	⊗

AWS KMS API operation	Multi-Region keys	Imported key material	KMS keys in a custom key store
Encrypt	✓ Valid only when `KeyUsage` is `ENCRYPT_DECRYPT`	✓	✓
GenerateDataKey	✓ Valid only on symmetric encryption KMS keys	✓	✓
GenerateDataKeyPair	✓ Valid only on symmetric encryption KMS keys	✓	⊗
GenerateDataKeyPairWithoutPlaintext	✓ Valid only on symmetric encryption KMS keys	✓	⊗

AWS KMS API operation	Multi-Region keys	Imported key material	KMS keys in a custom key store
GenerateDataKeyWithoutPlaintext	✓ Valid only on symmetric encryption KMS keys	✓	✓
GenerateMac Valid only on HMAC KMS keys	✓	✓	⊗
GetKeyPolicy	✓	✓	✓
GetKeyRotationStatus	✓	✓ (`KeyRotationEnabled` will always be `false`.)	⊗
GetParametersForImport	✓ Valid only for keys with imported key material (`Origin` is `EXTERNAL`).	✓	⊗
GetPublicKey Valid only for asymmetric KMS keys.	✓	✓	⊗

AWS KMS API operation	Multi-Region keys	Imported key material	KMS keys in a custom key store
ImportKeyMaterial	✓ Valid only for keys with imported key material (Origin is EXTERNAL).	✓	⊗
ListAliases	✓	✓	✓
ListGrants	✓	✓	✓
ListKeyPolicies	✓	✓	✓
ListResourceTags	✓	✓	✓
ListRetirableGrants	✓	✓	✓
PutKeyPolicy	✓	✓	✓
ReEncrypt	✓ Valid only when KeyUsage is ENCRYPT_DECRYPT	✓	✓

AWS KMS API operation	Multi-Region keys	Imported key material	KMS keys in a custom key store
ReplicateKey	✓ Valid only on multi-Region primary keys.	✓ Valid only on multi-Region primary keys.	⊗
RetireGrant	✓	✓	✓
RevokeGrant	✓	✓	✓
ScheduleKeyDeletion	✓	✓	✓
Sign Valid only on when `KeyUsage` is `SIGN_VERIFY` .	✓	✓	⊗
TagResource	✓	✓	✓
UntagResource	✓	✓	✓
UpdateAlias - The current KMS key and the new KMS key must be the same type (both symmetric or both asymmetri c or both HMAC) and they must have the same key usage.	✓	✓	✓
UpdateKeyDescription	✓	✓	✓

AWS KMS API operation	Multi-Region keys	Imported key material	KMS keys in a custom key store
UpdateReplicaRegion	✓	✓ Valid only on multi-Region keys.	⊗
Verify Valid only when `KeyUsage` is `SIGN_VERIFY` .	✓	✓	⊗
VerifyMac Valid only on HMAC KMS keys	✓	✓	⊗

Security of AWS Key Management Service

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security *of* the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to AWS Key Management Service (AWS KMS), see AWS Services in Scope by Compliance Program.
- **Security *in* the cloud** – Your responsibility is determined by the AWS service that you use. In AWS KMS, in addition to your configuration and use of AWS KMS keys, you are responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations

This documentation helps you understand how to apply the shared responsibility model when using AWS Key Management Service. It shows you how to configure AWS KMS to meet your security and compliance objectives.

Topics

Data protection in AWS Key Management Service

AWS Key Management Service stores and protects your encryption keys to make them highly available while providing you with strong and flexible access control.

Topics

Protecting key material

By default, AWS KMS generates and protects the cryptographic key material for KMS keys. In addition, AWS KMS offers options for key material that is created and protected outside of AWS KMS. For technical details about KMS keys and key material, see AWS Key Management Service Cryptographic Details.

Protecting key material generated in AWS KMS

When you create a KMS key, by default, AWS KMS generates and protects the cryptographic material for the KMS key.

To safeguard key material for KMS keys, AWS KMS relies on a distributed fleet of FIPS 140-2 Security Level 3–validated hardware security modules (HSMs). Each AWS KMS HSM is a dedicated, standalone hardware appliance designed to provide dedicated cryptographic functions to meet the security and scalability requirements of AWS KMS. (The HSMs that AWS KMS uses in China Regions are certified by OSCCA and comply with all pertinent Chinese regulations, but are not validated under the FIPS 140-2 Cryptographic Module Validation Program.)

The key material for a KMS key is encrypted by default when it is generated in the HSM. The key material is decrypted only within HSM volatile memory and only for the few milliseconds that it takes to use it in a cryptographic operation. Whenever the key material is not in active use, it is encrypted within the HSM and transferred to highly durable (99.999999999%), low-latency persistent storage where it remains separate and isolated from the HSMs. Plaintext key material never leaves the HSM security boundary; it is never written to disk or persisted in any storage medium. (The only exception is the public key of an asymmetric key pair, which is not secret.)

AWS asserts as a fundamental security principle that there is no human interaction with plaintext cryptographic key material of any type in any AWS service. There is no mechanism for anyone, including AWS service operators, to view, access, or export plaintext key material. This principle applies even during catastrophic failures and disaster recovery events. Plaintext customer key material in AWS KMS is used for cryptographic operations within AWS KMS FIPS validated HSMs only in response to authorized requests made to the service by the customer or their delegate.

For customer managed keys, the AWS account that creates the key is the sole and non-transferable owner of the key. The owning account has complete and exclusive control over the authorization policies that control access to the key. For AWS managed keys, the AWS account has complete control over the IAM policies that authorize requests to the AWS service.

Protecting key material generated outside of AWS KMS

AWS KMS provides alternatives to key material generated in AWS KMS.

Custom key stores, an optional AWS KMS feature, let you create KMS keys backed by key material that is generated and used outside of AWS KMS. KMS keys in AWS CloudHSM key stores are backed by keys in AWS CloudHSM hardware security modules that you control. These HSMs are certified at FIPS 140-2 Security Level 3. KMS keys in external key stores are backed by keys in an external key manager that you control and manage outside of AWS, such as a physical HSM in your private data center.

Another optional feature lets you import the key material for a KMS key. To protect imported key material while it is in transit to AWS KMS, you encrypt the key material using a public key from an RSA key pair generated in an AWS KMS HSM. The imported key material is decrypted in an AWS KMS HSM and re-encrypted under a symmetric key in the HSM. Like all AWS KMS key material, plaintext imported key material never leaves the HSMs unencrypted. However, the customer who provided the key material is responsible for secure use, durability, and maintenance of the key material outside of AWS KMS.

Data encryption

The data in AWS KMS consists of AWS KMS keys and the encryption key material they represent. This key material exists in plaintext only within AWS KMS hardware security modules (HSMs) and only when in use. Otherwise, the key material is encrypted and stored in durable persistent storage.

The key material that AWS KMS generates for KMS keys never leaves the boundary of AWS KMS HSMs unencrypted. It is not exported or transmitted in any AWS KMS API operations. The exception is for multi-Region keys, where AWS KMS uses a cross-Region replication mechanism to copy the key material for a multi-Region key from an HSM in one AWS Region to an HSM in a different AWS Region. For details, see Replication process for multi-Region keys in AWS Key Management Service Cryptographic Details.

Topics

- Encryption at rest
- Encryption in transit

Encryption at rest

AWS KMS generates key material for AWS KMS keys in FIPS 140-2 Security Level 3–compliant hardware security modules (HSMs). The only exception is China Regions, where the HSMs that AWS KMS uses to generate KMS keys comply with all pertinent Chinese regulations, but are not validated under the FIPS 140-2 Cryptographic Module Validation Program. When not in use, key material is encrypted by an HSM key and written to durable, persistent storage. The key material for KMS keys and the encryption keys that protect the key material never leave the HSMs in plaintext form.

Encryption and management of key material for KMS keys is handled entirely by AWS KMS.

For more details, see Working with AWS KMS keys in AWS Key Management Service Cryptographic Details.

Encryption in transit

Key material that AWS KMS generates for KMS keys is never exported or transmitted in AWS KMS API operations. AWS KMS uses key identifiers to represent the KMS keys in API operations. Similarly, key material for KMS keys in AWS KMS custom key stores is non-exportable and never transmitted in AWS KMS or AWS CloudHSM API operations.

However, some AWS KMS API operations return data keys. Also, customers can use API operations to import key material for selected KMS keys.

All AWS KMS API calls must be signed and transmitted using Transport Layer Security (TLS). AWS KMS requires TLS 1.2 and recommends TLS 1.3 in all regions. AWS KMS also supports hybrid post-quantum TLS for AWS KMS service endpoints in all regions, except China Regions. AWS KMS does not support hybrid post-quantum TLS for FIPS endpoints in AWS GovCloud (US). Calls to AWS

KMS also require a modern cipher suite that supports *perfect forward secrecy*, which means that compromise of any secret, such as a private key, does not also compromise the session key.

If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. To use standard AWS KMS endpoints or AWS KMS FIPS endpoints, clients must support TLS 1.2 or later. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2. For a list of AWS KMS FIPS endpoints, see AWS Key Management Service endpoints and quotas in the AWS General Reference.

Communications between AWS KMS service hosts and HSMs are protected using Elliptic Curve Cryptography (ECC) and Advanced Encryption Standard (AES) in an authenticated encryption scheme. For more details, see Internal communication security in AWS Key Management Service Cryptographic Details.

Internetwork traffic privacy

AWS KMS supports an AWS Management Console and a set of API operations that enable you to create and manage AWS KMS keys and use them in cryptographic operations.

AWS KMS supports two network connectivity options from your private network to AWS.

- An IPSec VPN connection over the internet
- AWS Direct Connect, which links your internal network to an AWS Direct Connect location over a standard Ethernet fiber-optic cable.

All AWS KMS API calls must be signed and be transmitted using Transport Layer Security (TLS). The calls also require a modern cipher suite that supports perfect forward secrecy. Traffic to the hardware security modules (HSMs) that store key material for KMS keys is permitted only from known AWS KMS API hosts over the AWS internal network.

To connect directly to AWS KMS from your virtual private cloud (VPC) without sending traffic over the public internet, use VPC endpoints, powered by AWS PrivateLink. For more information, see Connecting to AWS KMS through a VPC endpoint.

AWS KMS also supports a hybrid post-quantum key exchange option for the Transport Layer Security (TLS) network encryption protocol. You can use this option with TLS when you connect to AWS KMS API endpoints.

Identity and access management for AWS Key Management Service

AWS Identity and Access Management (IAM) helps you securely control access to AWS resources. Administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use AWS KMS resources. For more information, see Using IAM policies with AWS KMS.

Key policies are the primary mechanism for controlling access to KMS keys in AWS KMS. Every KMS key must have a key policy. You can also use IAM policies and grants, along with key policies, to control access to your KMS keys. For more information, see Authentication and access control for AWS KMS.

If you are using an Amazon Virtual Private Cloud (Amazon VPC), you can create an interface VPC endpoint to AWS KMS powered by AWS PrivateLink. You can also use VPC endpoint policies to determine which principals can access your AWS KMS endpoint, which API calls they can make, and which KMS key they can access. For details, see Controlling access to a VPC endpoint.

Logging and monitoring in AWS Key Management Service

Monitoring is an important part of understanding the availability, state, and usage of your AWS KMS keys in AWS KMS. Monitoring helps maintain the security, reliability, availability, and performance of your AWS solutions. AWS provides several tools for monitoring your KMS keys.

AWS CloudTrail Logs

> Every call to an AWS KMS API operation is captured as an event in an AWS CloudTrail log. These logs record all API calls from the AWS KMS console, and calls made by AWS KMS and other AWS services. Cross-account API calls, such as a call to use a KMS key in a different AWS account, are recorded in the CloudTrail logs of both accounts.
>
> When troubleshooting or auditing, you can use the log to reconstruct the lifecycle of a KMS key. You can also view its management and use of the KMS key in cryptographic operations. For more information, see the section called "Logging with AWS CloudTrail".

Amazon CloudWatch Logs

> Monitor, store, and access your log files from AWS CloudTrail and other sources. For more information, see the Amazon CloudWatch User Guide.

For AWS KMS, CloudWatch stores useful information that helps you to prevent problems with your KMS keys and the resources that they protect. For more information, see the section called "Monitoring with CloudWatch".

Amazon EventBridge

AWS KMS generates EventBridge events when your KMS key is rotated or deleted or the imported key material in your KMS key expires. Search for AWS KMS events (API operations) and route them to one or more target functions or streams to capture state information. For more information, see the section called "Monitoring with Amazon EventBridge" and the Amazon EventBridge User Guide.

Amazon CloudWatch Metrics

You can monitor your KMS keys using CloudWatch metrics, which collects and processes raw data from AWS KMS into performance metrics. The data are recorded in two-week intervals so you can view trends of current and historical information. This helps you to understand how your KMS keys are used and how their use changes over time. For information about using CloudWatch metrics to monitor KMS keys, see AWS KMS metrics and dimensions.

Amazon CloudWatch Alarms

Watch a single metric change over a time period that you specify. Then perform actions based on the value of the metric relative to a threshold over a number of time periods. For example, you can create a CloudWatch alarm that is triggered when someone tries to use a KMS key that is scheduled to be deleted in a cryptographic operation. This indicates that the KMS key is still being used and probably should not be deleted. For more information, see the section called "Creating an alarm".

AWS Security Hub

You can monitor your AWS KMS usage for security industry standards and best practices compliance using AWS Security Hub. Security Hub uses security controls to evaluate resource configurations and security standards to help you comply with various compliance frameworks. For more information, see AWS Key Management Service controls in the *AWS Security Hub User Guide*.

Compliance validation for AWS Key Management Service

Third-party auditors assess the security and compliance of AWS Key Management Service as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

Topics

- Compliance and security documents
- Learn more

Compliance and security documents

The following compliance and security documents cover AWS KMS. To view them, use AWS Artifact.

- Cloud Computing Compliance Controls Catalogue (C5)
- ISO 27001:2013 Statement of Applicability (SoA)
- ISO 27001:2013 Certification
- ISO 27017:2015 Statement of Applicability (SoA)
- ISO 27017:2015 Certification
- ISO 27018:2015 Statement of Applicability (SoA)
- ISO 27018:2014 Certification
- ISO 9001:2015 Certification
- PCI DSS Attestation of Compliance (AOC) and Responsibility Summary
- Service Organization Controls (SOC) 1 Report
- Service Organization Controls (SOC) 2 Report
- Service Organization Controls (SOC) 2 Report For Confidentiality
- FedRAMP-High

For help using AWS Artifact, see Downloading Reports in AWS Artifact.

Learn more

Your compliance responsibility when using AWS KMS is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. If your use of AWS KMS is subject to compliance with a published standard, AWS provides resources to help:

- AWS Services in Scope by Compliance Program – This page lists AWS services that are in scope of specific compliance programs. For general information, see AWS Compliance Programs.

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Config – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.

Resilience in AWS Key Management Service

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

In addition to the AWS global infrastructure, AWS KMS offers several features to help support your data resiliency and backup needs. For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Regional isolation

AWS Key Management Service (AWS KMS) is a self-sustaining Regional service that is available in all AWS Regions. The Regionally isolated design of AWS KMS ensures that an availability issue in one AWS Region cannot affect AWS KMS operation in any other Region. AWS KMS is designed to ensure *zero planned downtime*, with all software updates and scaling operations performed seamlessly and imperceptibly.

The AWS KMS Service Level Agreement (SLA) includes a service commitment of 99.999% for all KMS APIs. To fulfill this commitment, AWS KMS ensures that all data and authorization information required to execute an API request is available on all regional hosts that receive the request.

The AWS KMS infrastructure is replicated in at least three Availability Zones (AZs) in each Region. To ensure that multiple host failures do not affect AWS KMS performance, AWS KMS is designed to service customer traffic from any of the AZs in a Region.

Changes that you make to the properties or permissions of a KMS key are replicated to all hosts in the Region to ensure that subsequent request can be processed correctly by any host in the Region. Requests for cryptographic operations using your KMS key are forwarded to a fleet of AWS KMS hardware security modules (HSMs), any of which can perform the operation with the KMS key.

Multi-tenant design

The multi-tenant design of AWS KMS enables it to fulfill the 99.999% availability SLA, and to sustain high request rates, while protecting the confidentiality of your keys and data.

Multiple integrity-enforcing mechanisms are deployed to ensure that the KMS key that you specified for the cryptographic operation is always the one that is used.

The plaintext key material for your KMS keys is protected extensively. The key material is encrypted in the HSM as soon as it is created, and the encrypted key material is immediately moved to secure, low latency storage. The encrypted key is retrieved and decrypted within the HSM just in time for use. The plaintext key remains in HSM memory only for the time needed to complete the cryptographic operation. Then it is re-encrypted in the HSM and the encrypted key is returned to storage. Plaintext key material never leaves the HSMs; it is never written to persistent storage.

For more information about the mechanisms that AWS KMS uses to secure your keys, see AWS Key Management Service Cryptographic Details.

Resilience best practices in AWS KMS

To optimize resilience for your AWS KMS resources, consider the following strategies.

- To support your backup and disaster recovery strategy, consider *multi-Region keys*, which are KMS keys created in one AWS Region and replicated only to Regions that you specify. With multi-Region keys, you can move encrypted resources between AWS Regions (within the same partition) without ever exposing the plaintext, and decrypt the resource, when needed, in any of its destination Regions. Related multi-Region keys are interoperable because they share the same key material and key ID, but they have independent key policies for high-resolution access control. For details, see Multi-Region keys in AWS KMS.
- To protect your keys in a multi-tenant service like AWS KMS, be sure to use access controls, including key policies and IAM policies. In addition, you can send your requests to AWS KMS

using a *VPC interface endpoint* powered by AWS PrivateLink. When you do, all communication between your Amazon VPC and AWS KMS is conducted entirely within the AWS network using a dedicated AWS KMS endpoint restricted to your VPC. You can further secure these requests by creating an additional authorization layer using VPC endpoint policies. For details, see Connecting to AWS KMS through a VPC endpoint.

Infrastructure security in AWS Key Management Service

As a managed service, AWS Key Management Service (AWS KMS) is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes.

To access AWS KMS over the network, you can call the AWS KMS API operations that are described in the AWS Key Management Service API Reference. AWS KMS requires TLS 1.2 and recommends TLS 1.3 in all regions. AWS KMS also supports hybrid post-quantum TLS for AWS KMS service endpoints in all regions, except China Regions. AWS KMS does not support hybrid post-quantum TLS for FIPS endpoints in AWS GovCloud (US). To use standard AWS KMS endpoints or AWS KMS FIPS endpoints, clients must support TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems, such as Java 7 and later, support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

You can call these API operations from any network location, but AWS KMS supports global policy conditions that let you control access to a KMS key based on the source IP address, VPC, and VPC endpoint. You can use these condition keys in key policies and IAM policies. However, these conditions can prevent AWS from using the KMS key on your behalf. For details, see AWS global condition keys.

For example, the following key policy statement allows users who can assume the `KMSTestRole` role to use this AWS KMS key for the specified cryptographic operations unless the source IP address is one of the IP addresses specified in the policy.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"AWS":
    "arn:aws:iam::111122223333:role/KMSTestRole"},
    "Action": [
      "kms:Encrypt",
      "kms:Decrypt",
      "kms:ReEncrypt*",
      "kms:GenerateDataKey*",
      "kms:DescribeKey"
    ],
    "Resource": "*",
    "Condition": {
      "NotIpAddress": {
        "aws:SourceIp": [
          "192.0.2.0/24",
          "203.0.113.0/24"
        ]
      }
    }
  }
}
```

Isolation of Physical Hosts

The security of the physical infrastructure that AWS KMS uses is subject to the controls described in the **Physical and Environmental Security** section of the Amazon Web Services: Overview of Security Processes. You can find more detail in compliance reports and third-party audit findings listed in the previous section.

AWS KMS is supported by dedicated hardened hardware security modules (HSMs) designed with specific controls to resist physical attacks. The HSMs are physical devices that *do not* have a virtualization layer, such as a hypervisor, that shares the physical device among several logical tenants. The key material for AWS KMS keys is stored only in volatile memory on the HSMs, and only while the KMS key is in use. This memory is erased when the HSM moves out of the operational state, including intended and unintended shutdowns and resets. For detailed information about the operation of AWS KMS HSMs, see AWS Key Management Service Cryptographic Details.

Security best practices for AWS Key Management Service

AWS Key Management Service (AWS KMS) supports many security features that you can implement to enhance the protection of your encryption keys, including key policies and IAM policies, an encryption context option for cryptographic operations on symmetric encryption keys, an extensive set of condition keys to refine your key policies and IAM policies, and grant constraints to limit grants.

These security features are described in detail in AWS Key Management Service Best Practices (PDF). The general guidelines in this technical paper do not represent a complete security solution. Because not all best practices are appropriate for all situations, these are not intended to be prescriptive.

See also

- Best practices for IAM policies
- Best practices for AWS KMS grants
- Security best practices in IAM in the *IAM User Guide*

Quotas

To make AWS KMS responsive and performant for all users, AWS KMS applies two types of quotas, resource quotas and request quotas. Each quota is calculated independently for each Region of each AWS account.

All AWS KMS quotas are adjustable, except for the key policy document size resource quota, on-demand rotation resource quota, and the AWS CloudHSM key store request quota. To request a quota increase, see Requesting a quota increase in the *Service Quotas User Guide*. To request a quota decrease, to change a quota that is not listed in Service Quotas, or to change a quota in an AWS Region where Service Quotas for AWS KMS is not available, please visit AWS Support Center and create a case.

Topics

Resource quotas

AWS KMS establishes resource quotas to ensure that it can provide fast and resilient service to all of our customers. Some resource quotas apply only to resources that you create, but not to resources that AWS services create for you. Resources that you use, but that aren't in your AWS account, such as AWS owned keys, do not count against these quotas.

If you have exceeded a resource limit, requests to create an additional resource of that type generate an `LimitExceededException` error message.

All AWS KMS resource quotas are adjustable, except for the key policy document size quota and on-demand rotation resource quota. To request a quota increase, see Requesting a quota increase in the *Service Quotas User Guide*. To request a quota decrease, to change a quota that is not listed in Service Quotas, or to change a quota in an AWS Region where Service Quotas for AWS KMS is not available, please visit AWS Support Center and create a case.

The following table lists and describes the AWS KMS resource quotas in each AWS account and Region.

Quota name	Default value	Applies to	Adjustable
AWS KMS keys	100,000	Customer managed keys	Yes
Aliases per KMS key	50	Customer created aliases	Yes
Grants per KMS key	50,000	Customer managed keys	Yes
Key policy document size	32 KB (32,768 bytes)	Customer managed keys AWS managed keys	No
Custom key store resource quota	10	AWS account and Region	Yes
On-demand rotation	10	Customer managed keys	No

In addition to resource quotas, AWS KMS uses request quotas to ensure the responsiveness of the service. For details, see the section called “Request quotas”.

AWS KMS keys: 100,000

You can have up to 100,000 customer managed keys in each Region of your AWS account. This quota applies to all customer managed keys in all AWS Regions regardless of their key spec or key state. Each KMS key is considered to be one resource. AWS managed keys and AWS owned keys do not count against this quota.

Aliases per KMS key: 50

You can associate up to 50 aliases with each customer managed key. Aliases that AWS associates with AWS managed keys do not count against this quota. You might encounter this quota when you create or update an alias.

> **Note**
>
> The kms:ResourceAliases condition is effective only when the KMS key conforms to this quota. If a KMS key exceeds this quota, principals who are authorized to use the KMS key by the `kms:ResourceAliases` condition are denied access to the KMS key. For details, see Access denied due to alias quota.

The Aliases per KMS key quota replaces the Aliases per Region quota that limited the total number of aliases in each Region of an AWS account. AWS KMS has eliminated the Aliases per Region quota.

Grants per KMS key: 50,000

Each customer managed key can have up to 50,000 grants, including the grants created by AWS services that are integrated with AWS KMS. This quota does not apply to AWS managed keys or AWS owned keys.

One effect of this quota is that you cannot perform more than 50,000 grant-authorized operations that use the same KMS key at the same time. After you reach the quota, you can create new grants on the KMS key only when an active grant is retired or revoked.

For example, when you attach an Amazon Elastic Block Store (Amazon EBS) volume to an Amazon Elastic Compute Cloud (Amazon EC2) instance, the volume is decrypted so you can read it. To get permission to decrypt the data, Amazon EBS creates a grant for each volume. Therefore, if all of your Amazon EBS volumes use the same KMS key, you cannot attach more than 50,000 volumes at one time.

Key policy document size: 32 KB

The maximum length of each key policy document is 32 KB (32,768 bytes). If you use a larger policy document to create or update the key policy for a KMS key, the operation fails.

This quota is not adjustable. You cannot increase it by using Service Quotas or by creating a case in AWS Support. If your key policy is approaching the limit, consider using grants instead of policy statements. Grants are particularly well suited to temporary or very specific permissions.

You use a key policy document whenever you create or change a key policy by using the default view or policy view in the AWS Management Console, or the PutKeyPolicy operation. This quota applies to your key policy document, even if you use the default view in the AWS KMS console, where you don't edit the JSON statements directly.

Custom key stores resource quota: 10

You can create up to 10 custom key stores in each AWS account and Region. If you try to create more, the CreateCustomKeyStore operation fails.

This quota applies to the total number of custom key stores in each account and region, including all AWS CloudHSM key stores and external key stores, regardless of their connection state.

On-demand rotation: 10

You can perform on-demand key rotation a maximum of 10 times per KMS key. If you try to perform more on-demand rotations, the RotateKeyOnDemand operation fails.

This quota is not adjustable. You cannot increase it by using Service Quotas or by creating a case in AWS Support. To prevent reaching the on-demand rotation quota, we recommend using automatic key rotation whenever possible.

Request quotas

AWS KMS establishes quotas for the number of API operations requested in each second. The request quotas differ with the API operation, the AWS Region, and other factors, such as the KMS key type. When you exceed an API request quota, AWS KMS throttles the request.

All AWS KMS request quotas are adjustable, except for the AWS CloudHSM key store request quota. To request a quota increase, see Requesting a quota increase in the *Service Quotas User Guide*. To request a quota decrease, to change a quota that is not listed in Service Quotas, or to change a quota in an AWS Region where Service Quotas for AWS KMS is not available, please visit AWS Support Center and create a case.

If you are exceeding the request quota for the GenerateDataKey operation, consider using the data key caching feature of the AWS Encryption SDK. Reusing data keys might reduce the frequency of your requests to AWS KMS.

In addition to request quotas, AWS KMS uses resource quotas to ensure capacity for all users. For details, see Resource quotas.

To view trends in your request rates, use the Service Quotas console. You can also create an Amazon CloudWatch alarm that alerts you when your request rate reaches a certain percentage of a quota value. For details, see Manage your AWS KMS API request rates using Service Quotas and Amazon CloudWatch in the *AWS Security Blog*.

Topics

Request quotas for each AWS KMS API operation

This table lists the Service Quotas quota code and the default value for each AWS KMS request quota. All AWS KMS request quotas are adjustable, except for the AWS CloudHSM key store request quota.

> **Note**
>
> You might need to scroll horizontally or vertically to see all of the data in this table.

Quota name	Default value (requests per second)
`Cryptographic operations (symmetric) request rate` Applies to: - `Decrypt` - `Encrypt` - `GenerateDataKey` - `GenerateDataKeyWithoutPlaintext` - `GenerateMac`	These shared quotas vary with the AWS Region and the type of KMS key used in the request. Each quota is calculated separately. - 5,500 (shared) - 10,000 (shared) in the following Regions: - US East (Ohio), us-east-2 - Asia Pacific (Singapore), ap-southeast-1 - Asia Pacific (Sydney), ap-southeast-2 - Asia Pacific (Tokyo), ap-northeast-1 - Europe (Frankfurt), eu-central-1

Quota name	Default value (requests per second)
- GenerateRandom - ReEncrypt - VerifyMac	- Europe (London), eu-west-2 - 50,000 (shared) in the following Regions: - US East (N. Virginia), us-east-1 - US West (Oregon), us-west-2 - Europe (Ireland), eu-west-1
`Cryptographic operations (RSA) request rate` Applies to: - Decrypt - Encrypt - ReEncrypt - Sign - Verify	500 (shared) for RSA KMS keys
`Cryptographic operations (ECC and SM2) request rate` Applies to: - Decrypt—only supported for SM2 (China Regions only) KMS keys - Encrypt—only supported for SM2 (China Regions only) KMS keys - ReEncrypt —only supported for SM2 (China Regions only) KMS keys - Sign - Verify	300 (shared) for elliptic curve (ECC) and SM2 (China Regions only) KMS keys

Quota name	Default value (requests per second)
`Custom key store request quotas` Applies to: - `Decrypt` - `DeriveSharedSecret` - `Encrypt` - `GenerateDataKey` - `GenerateDataKeyWithoutPlaintext` - `GenerateRandom` - `ReEncrypt`	Custom key store request quotas are calculated separately for each custom key store - 1,800 (shared) for each AWS CloudHSM key store - 1,800 (shared) for each external key store
`CancelKeyDeletion request rate`	5
`ConnectCustomKeyStore request rate`	5
`CreateAlias request rate`	5
`CreateCustomKeyStore request rate`	5
`CreateGrant request rate`	50
`CreateKey request rate`	5
`DeleteAlias request rate`	15
`DeleteCustomKeyStore request rate`	5
`DeleteImportedKeyMaterial request rate`	5
`DescribeCustomKeyStores request rate`	5
`DescribeKey request rate`	2000

Quota name	Default value (requests per second)
DisableKey request rate	5
DisableKeyRotation request rate	5
DisconnectCustomKeyStore request rate	5
EnableKey request rate	5
EnableKeyRotation request rate	15
GenerateDataKeyPair (ECC_NIST_P256) request rate Applies to: • GenerateDataKeyPair • GenerateDataKeyPairWithoutPlaintext	100
GenerateDataKeyPair (ECC_NIST_P384) request rate Applies to: • GenerateDataKeyPair • GenerateDataKeyPairWithoutPlaintext	100
GenerateDataKeyPair (ECC_NIST_P521) request rate Applies to: • GenerateDataKeyPair • GenerateDataKeyPairWithoutPlaintext	100

Quota name	Default value (requests per second)
`GenerateDataKeyPair (ECC_SECG_P256K1) request rate` Applies to: • `GenerateDataKeyPair` • `GenerateDataKeyPairWithoutPlaintext`	100
`GenerateDataKeyPair (RSA_2048) request rate` Applies to: • `GenerateDataKeyPair` • `GenerateDataKeyPairWithoutPlaintext`	1
`GenerateDataKeyPair (RSA_3072) request rate` Applies to: • `GenerateDataKeyPair` • `GenerateDataKeyPairWithoutPlaintext`	0.5 (1 in each 2-second interval)
`GenerateDataKeyPair (RSA_4096) request rate` Applies to: • `GenerateDataKeyPair` • `GenerateDataKeyPairWithoutPlaintext`	0.1 (1 in each 10-second interval)

Quota name	Default value (requests per second)
GenerateDataKeyPair (SM2 — China Regions only) request rate Applies to: • GenerateDataKeyPair • GenerateDataKeyPairWithoutPlaintext	25
GetKeyPolicy request rate	1000
GetKeyRotationStatus request rate	1000
GetParametersForImport request rate	0.25 (1 in each 4-second interval)
GetPublicKey request rate	2000
ImportKeyMaterial request rate	5
ListAliases request rate	500
ListGrants request rate	100
ListKeyPolicies request rate	100
ListKeys request rate	500
ListKeyRotations request rate	100
ListResourceTags request rate	2000
ListRetirableGrants request rate	100
PutKeyPolicy request rate	15

Quota name	Default value (requests per second)
`ReplicateKey request rate` A `ReplicateKey` operation counts as one `ReplicateKey` request in the primary key's Region and two `CreateKey` requests in the replica's Region. One of the `CreateKey` requests is a dry run to detect potential problems before creating the key.	5
`RetireGrant request rate`	30
`RevokeGrant request rate`	30
`RotateKeyOnDemand request rate`	5
`ScheduleKeyDeletion request rate`	15
`TagResource request rate`	10
`UntagResource request rate`	5
`UpdateAlias request rate`	5
`UpdateCustomKeyStore request rate`	5
`UpdateKeyDescription request rate`	5
`UpdatePrimaryRegion request rate` An `UpdatePrimaryRegion` operation counts as two `UpdatePrimaryRegion` requests; one request in each of the two affected Regions.	5

Applying request quotas

When reviewing request quotas, keep in mind the following information.

- Request quotas apply to both customer managed keys and AWS managed keys. The use of AWS owned keys does not count against request quotas for your AWS account, even when they are used to protect resources in your account.
- Request quotas apply to requests sent to FIPS endpoints and non-FIPS endpoints. For a list of AWS KMS service endpoints, see AWS Key Management Service endpoints and quotas in the AWS General Reference.
- Throttling is based on all requests on KMS keys of all types in the Region. This total includes requests from all principals in the AWS account, including requests from AWS services on your behalf.
- Each request quota is calculated independently. For example, requests for the CreateKey operation have no effect on the request quota for the CreateAlias operation. If your `CreateAlias` requests are throttled, your `CreateKey` requests can still complete successfully.
- Although cryptographic operations share a quota, the shared quota is calculated independently of quotas for other operations. For example, calls to the Encrypt and Decrypt operations share a request quota, but that quota is independent of the quota for management operations, such as EnableKey. For example, in the Europe (London) Region, you can perform 10,000 cryptographic operations on symmetric KMS keys *plus* 5 `EnableKey` operations per second without being throttled.

Shared quotas for cryptographic operations

AWS KMS cryptographic operations share request quotas. You can request any combination of the cryptographic operations that are supported by the KMS key, just so the total number of cryptographic operations doesn't exceed the request quota for that type of KMS key. The exceptions are GenerateDataKeyPair and GenerateDataKeyPairWithoutPlaintext, which share a separate quota.

The quotas for different types of KMS keys are calculated independently. Each quota applies to all requests for these operations in the AWS account and Region with the given key type in each one-second interval.

- *Cryptographic operations (symmetric) request rate* is the shared request quota for cryptographic operations using symmetric KMS keys in an account and region. This quota applies to cryptographic operations with symmetric encryption keys and HMAC keys, which are also symmetric.

For example, you might be using symmetric KMS keys in an AWS Region with a shared quota of 10,000 requests per second. When you make 7,000 GenerateDataKey requests per second and 2,000 Decrypt requests per second, AWS KMS doesn't throttle your requests. However, when you make 9,500 `GenerateDataKey` requests and 1,000 Encrypt and requests per second, AWS KMS throttles your requests because they exceed the shared quota.

Cryptographic operations on the symmetric encryption KMS keys in a custom key store count toward both the *Cryptographic operations (symmetric) request rate* for the account and the custom key store request quota for the custom key store.

- *Cryptographic operations (RSA) request rate* is the shared request quota for cryptographic operations using RSA asymmetric KMS keys.

  For example, with a request quota of 500 operations per second, you can make 200 Encrypt requests and 100 Decrypt requests with RSA KMS keys that can encrypt and decrypt, plus 50 Sign requests and 150 Verify requests with RSA KMS keys that can sign and verify.
- *Cryptographic operations (ECC) request rate* is the shared request quota for cryptographic operations using elliptic curve (ECC) asymmetric KMS keys.

  For example, with a request quota of 300 operations per second, you can make 100 Sign requests and 200 Verify requests with RSA KMS keys that can sign and verify.
- *Cryptographic operations (SM — China Regions only) request rate* is the shared request quota for cryptographic operations using SM asymmetric KMS keys.

  For example, with a request quota of 300 operations per second, you can make 100 Encrypt requests and 100 Decrypt requests with SM2 KMS keys that can encrypt and decrypt, plus 50 Sign requests and 50 Verify requests with SM2 KMS keys that can sign and verify.
- *Custom key store request quota* is the shared request quota for cryptographic operations on KMS keys in a custom key store. This quota is calculated separately for each custom key store.

  Cryptographic operations on the symmetric encryption KMS keys in a custom key store count toward both the *Cryptographic operations (symmetric) request rate* for the account and the custom key store request quota for the custom key store.

The quotas for different key types are also calculated independently. For example, in the Asia Pacific (Singapore) Region, if you use both symmetric and asymmetric KMS keys, you can make up to 10,000 calls per second with symmetric KMS keys (including HMAC keys) *plus* up to 500 additional calls per second with your RSA asymmetric KMS keys, *plus* up to 300 additional requests per second with your ECC-based KMS keys.

API requests made on your behalf

You can make API requests directly or by using an integrated AWS service that makes API requests to AWS KMS on your behalf. The quota applies to both kinds of requests.

For example, you might store data in Amazon S3 using server-side encryption with a KMS key (SSE-KMS). Each time you upload or download an S3 object that's encrypted with SSE-KMS, Amazon S3 makes a `GenerateDataKey` (for uploads) or `Decrypt` (for downloads) request to AWS KMS on your behalf. These requests count toward your quota, so AWS KMS throttles the requests if you exceed a combined total of 5,500 (or 10,000 or 50,000 depending upon your AWS Region) uploads or downloads per second of S3 objects encrypted with SSE-KMS.

Cross-account requests

When an application in one AWS account uses a KMS key owned by a different account, it's known as a *cross-account request*. For cross-account requests, AWS KMS throttles the account that makes the requests, not the account that owns the KMS key. For example, if an application in account A uses a KMS key in account B, the KMS key use applies only to the quotas in account A.

Custom key store request quotas

AWS KMS maintains request quotas for cryptographic operations on the KMS keys in a custom key store. These request quotas are calculated separately for each custom key store.

Custom key store request quota	Default value (requests per second) for each custom key store	Adjustable
AWS CloudHSM key store request quota	1800	No
External key store request quota	1800	Yes

Note

AWS KMS custom key store request quotas do not appear in the Service Quotas console. You cannot view or manage these quotas by using Service Quotas API operations. To request a change to your external key store request quota, visit the AWS Support Center and create a case.

If the AWS CloudHSM cluster associated with an AWS CloudHSM key store is processing numerous commands, including those unrelated to the custom key store, you might get an AWS KMS `ThrottlingException` at a lower-than-expected rate. If this occurs, lower your request rate to AWS KMS, reduce the unrelated load, or use a dedicated AWS CloudHSM cluster for your AWS CloudHSM key store.

AWS KMS reports throttling of external key store requests in the `ExternalKeyStoreThrottle` CloudWatch metric. You can use this metric to view throttling patterns, create alarms, and adjust your external key store request quota.

A request for a cryptographic operation on a KMS key in a custom key store counts toward two quotas:

- Cryptographic operations (symmetric) request rate quota (per account)

  Requests for cryptographic operations on KMS keys in a custom key store count toward the `Cryptographic operations (symmetric) request rate` quota for each AWS account and Region. For example, in US East (N. Virginia) (us-east-1), each AWS account can have up to 50,000 requests per second on symmetric encryption KMS keys, including requests that use a KMS key in a custom key store.
- Custom key store request quota (per custom key store)

  Requests for cryptographic operations on KMS keys in a custom key store also count toward a `Custom key store request quota` of 1,800 operations per second. These quotas are calculated separately for each custom key store. They might include requests from multiple AWS accounts that use KMS keys in the custom key store.

For example, an Encrypt operation on a KMS key in a custom key store (either type) in the US East (N. Virginia) (us-east-1) Region counts toward the `Cryptographic operations (symmetric) request rate` account-level quota (50,000 requests per second) for its account and Region, and toward a `Custom key store request quota` (1,800 requests per second) for its custom key store. However, a request for a management operation, such as PutKeyPolicy, on a KMS key in a custom key store applies only to its account-level quota (15 requests per second).

Throttling AWS KMS requests

To ensure that AWS KMS can provide fast and reliable responses to API requests from all customers, it throttles API requests that exceed certain boundaries.

Throttling occurs when AWS KMS rejects a request that might otherwise be valid, and returns a `ThrottlingException` error like the following one.

```
You have exceeded the rate at which you may call KMS. Reduce the frequency of your
 calls.
(Service: AWSKMS; Status Code: 400; Error Code: ThrottlingException; Request ID: <ID>
```

AWS KMS throttles requests for the following conditions.

- The rate of requests per second exceeds the AWS KMS request quota for an account and Region.

  For example, if users in your account submit 1000 `DescribeKey` requests in a second, AWS KMS throttles all subsequent `DescribeKey` requests in that second.

  To respond to throttling, use a backoff and retry strategy. This strategy is implemented automatically for HTTP 400 errors in some AWS SDKs.
- A burst or sustained high rate of requests to change the state of the same KMS key. This condition is often known as a "hot key."

  For example, if an application in your account sends a persistent volley of `EnableKey` and `DisableKey` requests for the same KMS key, AWS KMS throttles the requests. This throttling occurs even if the requests don't exceed the request-per-second request limit for the `EnableKey` and `DisableKey` operations.

  To respond to throttling, adjust your application logic so it makes only required requests or it consolidates the requests of multiple functions.
- Requests for operations on KMS keys in a AWS CloudHSM key store might be throttled at a lower-than-expected rate when the AWS CloudHSM cluster associated with the AWS CloudHSM key store is processing numerous commands, including those unrelated to the AWS CloudHSM key store.

(AWS KMS no longer throttles requests for operations on KMS keys in a AWS CloudHSM key store when there are no available PKCS #11 sessions for the AWS CloudHSM cluster. Instead, it throws a `KMSInternalException` and recommends that you retry your request.)

To view trends in your request rates, use the Service Quotas console. You can also create an Amazon CloudWatch alarm that alerts you when your request rate reaches a certain percentage of a quota value. For details, see Manage your AWS KMS API request rates using Service Quotas and Amazon CloudWatch in the *AWS Security Blog*.

All AWS KMS quotas are adjustable, except for the key policy document size resource quota, on-demand rotation resource quota, and the AWS CloudHSM key store request quota. To request a quota increase, see Requesting a quota increase in the *Service Quotas User Guide*. To request a quota decrease, to change a quota that is not listed in Service Quotas, or to change a quota in an AWS Region where Service Quotas for AWS KMS is not available, please visit AWS Support Center and create a case.

Note

AWS KMS custom key store request quotas do not appear in the Service Quotas console. You cannot view or manage these quotas by using Service Quotas API operations. To request a change to your external key store request quota, visit the AWS Support Center and create a case.

How AWS services use AWS KMS

Many AWS services use AWS KMS to support encryption of your data. When an AWS service is integrated with AWS KMS, you can use the AWS KMS keys in your account to protect the data that the service receives, stores, or manages for you. For the complete list of AWS services integrated with AWS KMS, see AWS Service Integration.

The following topics discuss in detail how particular services use AWS KMS, including the KMS keys they support, how they manage data keys, the permissions they require, and how to track each service's use of the KMS keys in your account.

⚠ **Important**

AWS services that are integrated with AWS KMS use only symmetric encryption KMS keys to encrypt your data. These services do not support encryption with asymmetric KMS keys. For help determining whether a KMS key is symmetric or asymmetric, see Identifying asymmetric KMS keys.

Topics

How AWS CloudTrail uses AWS KMS

You can use AWS CloudTrail to record AWS API calls and other activity for your AWS account and to save the recorded information to log files in an Amazon Simple Storage Service (Amazon S3) bucket that you choose. By default, the log files that CloudTrail puts in your S3 bucket are encrypted using server-side encryption with Amazon S3–managed encryption keys (SSE-S3). But you can choose instead to use server-side encryption with a KMS key (SSE-KMS). To learn how to encrypt your CloudTrail log files with AWS KMS, see Encrypting CloudTrail Log Files with AWS KMS keys (SSE-KMS) in the *AWS CloudTrail User Guide*.

⚠ **Important**

AWS CloudTrail and Amazon S3 support only symmetric AWS KMS keys. You cannot use an asymmetric KMS key to encrypt your CloudTrail Logs. For help determining whether a KMS key is symmetric or asymmetric, see Identifying asymmetric KMS keys.

You do not pay a key usage charge when CloudTrail reads or writes log files encrypted with an SSE-KMS key. However, you pay a key usage charge when you access CloudTrail log files encrypted with an SSE-KMS key. For information about AWS KMS pricing, see AWS Key Management Service Pricing. For information about CloudTrail pricing, see AWS CloudTrail pricing and Managing costs in the *AWS CloudTrail User Guide*.

Topics

Understanding when your KMS key is used

Encrypting CloudTrail log files with AWS KMS builds on the Amazon S3 feature called server-side encryption with an AWS KMS key (SSE-KMS). To learn more about SSE-KMS, see How Amazon Simple Storage Service (Amazon S3) uses AWS KMS in this guide or Protecting data using server-side encryption with KMS keys (SSE-KMS) in the *Amazon Simple Storage Service User Guide*.

When you configure AWS CloudTrail to use SSE-KMS to encrypt your log files, CloudTrail and Amazon S3 use your AWS KMS keys when you perform certain actions with those services. The following sections explain when and how those services can use your KMS key, and provide additional information that you can use to validate this explanation.

Actions that cause CloudTrail and Amazon S3 to use your KMS key

- You configure CloudTrail to encrypt log files with your AWS KMS key
- CloudTrail puts a log file into your S3 bucket
- You get an encrypted log file from your S3 bucket

You configure CloudTrail to encrypt log files with your AWS KMS key

When you update your CloudTrail configuration to use your KMS key, CloudTrail sends a GenerateDataKey request to AWS KMS to verify that the KMS key exists and that CloudTrail has permission to use it for encryption. CloudTrail does not use the resulting data key.

The GenerateDataKey request includes the following information for the encryption context:

- The Amazon Resource Name (ARN) of the CloudTrail trail
- The ARN of the S3 bucket and path where the CloudTrail log files are delivered

The GenerateDataKey request results in an entry in your CloudTrail logs similar to the following example. When you see a log entry like this one, you can determine that CloudTrail

(❶)

called the AWS KMS

(❷)

GenerateDataKey operation

(❸)

for a specific trail

(❹).

AWS KMS created the data key under a specific KMS key

(❺).

> **Note**
>
> You might need to scroll to the right to see some of the callouts in the following example log entry.

```
{
  "eventVersion": "1.02",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::086441151436:user/
AWSCloudTrail",❶
    "accountId": "086441151436",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "userName": "AWSCloudTrail",
    "sessionContext": {"attributes": {
      "mfaAuthenticated": "false",
      "creationDate": "2015-11-11T21:15:33Z"
    }},
    "invokedBy": "internal.amazonaws.com"
  },
  "eventTime": "2015-11-11T21:15:33Z",
  "eventSource":
 "kms.amazonaws.com",❷
  "eventName":
 "GenerateDataKey",❸
  "awsRegion": "us-west-2",
  "sourceIPAddress": "internal.amazonaws.com",
  "userAgent": "internal.amazonaws.com",
  "requestParameters": {
    "keyId": "arn:aws:kms:us-west-2:111122223333:alias/ExampleAliasForCloudTrailKMS
 key",
    "encryptionContext": {
      "aws:cloudtrail:arn": "arn:aws:cloudtrail:us-west-2:111122223333:trail/
Default",❹
      "aws:s3:arn": "arn:aws:s3:::example-bucket-for-CT-logs/AWSLogs/111122223333/"
    },
    "keySpec": "AES_256"
  },
  "responseElements": null,
  "requestID": "581f1f11-88b9-11e5-9c9c-595a1fb59ac0",
  "eventID": "3cdb2457-c035-4890-93b6-181832b9e766",
  "readOnly": true,
  "resources": [{
    "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",❺
    "accountId": "111122223333"
  }],
  "eventType": "AwsServiceEvent",
  "recipientAccountId": "111122223333"
}
```

CloudTrail puts a log file into your S3 bucket

Each time CloudTrail puts a log file into your S3 bucket, Amazon S3 sends a `GenerateDataKey` request to AWS KMS on behalf of CloudTrail. In response to this request, AWS KMS generates a unique data key and then sends Amazon S3 two copies of the data key, one in plaintext and one that is encrypted with the specified KMS key. Amazon S3 uses the plaintext data key to encrypt the CloudTrail log file and then removes the plaintext data key from memory as soon as possible after use. Amazon S3 stores the encrypted data key as metadata with the encrypted CloudTrail log file.

The `GenerateDataKey` request includes the following information for the encryption context:

- The Amazon Resource Name (ARN) of the CloudTrail trail
- The ARN of the S3 object (the CloudTrail log file)

Each `GenerateDataKey` request results in an entry in your CloudTrail logs similar to the following example. When you see a log entry like this one, you can determine that CloudTrail (❶) called the AWS KMS (❷) `GenerateDataKey` operation (❸) for a specific trail (❹) to protect a specific log file (❺). AWS KMS created the data key under the specified KMS key (❻), shown twice in the same log entry.

> **Note**
>
> You might need to scroll to the right to see some of the callouts in the following example log entry.

```
{
  "eventVersion": "1.02",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROACKCEVSQ6C2EXAMPLE:i-34755b85",
    "arn": "arn:aws:sts::086441151436:assumed-role/AWSCloudTrail/
i-34755b85",❶
    "accountId": "086441151436",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2015-11-11T20:45:25Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::086441151436:role/AWSCloudTrail",
        "accountId": "086441151436",
        "userName": "AWSCloudTrail"
      }
    },
    "invokedBy": "internal.amazonaws.com"
  },
  "eventTime": "2015-11-11T21:15:58Z",
  "eventSource":
 "kms.amazonaws.com",❷
  "eventName":
 "GenerateDataKey",❸
  "awsRegion": "us-west-2",
  "sourceIPAddress": "internal.amazonaws.com",
  "userAgent": "internal.amazonaws.com",
  "requestParameters": {
    "encryptionContext": {
      "aws:cloudtrail:arn": "arn:aws:cloudtrail:us-west-2:111122223333:trail/
Default",❹
      "aws:s3:arn": "arn:aws:s3:::example-bucket-for-CT-logs/
AWSLogs/111122223333/CloudTrail/us-west-2/2015/11/11/111122223333_CloudTrail_us-
west-2_20151111T2115Z_7JREEBimdK8d2nC9.json.gz"❺
    },
    "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",❻
    "keySpec": "AES_256"
  },
  "responseElements": null,
  "requestID": "66f3f74a-88b9-11e5-b7fb-63d925c72ffe",
  "eventID": "7738554f-92ab-4e27-83e3-03354b1aa898",
  "readOnly": true,
  "resources": [{
    "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",❻
    "accountId": "111122223333"
  }],
  "eventType": "AwsServiceEvent",
  "recipientAccountId": "111122223333"
}
```

You get an encrypted log file from your S3 bucket

Each time you get an encrypted CloudTrail log file from your S3 bucket, Amazon S3 sends a `Decrypt` request to AWS KMS on your behalf to decrypt the log file's encrypted data key. In response to this request, AWS KMS uses your KMS key to decrypt the data key and then sends the plaintext data key to Amazon S3. Amazon S3 uses the plaintext data key to decrypt the CloudTrail log file and then removes the plaintext data key from memory as soon as possible after use.

The `Decrypt` request includes the following information for the encryption context:

- The Amazon Resource Name (ARN) of the CloudTrail trail
- The ARN of the S3 object (the CloudTrail log file)

Each `Decrypt` request results in an entry in your CloudTrail logs similar to the following example. When you see a log entry like this one, you can determine that a user in your AWS account (❶) called the AWS KMS (❷) `Decrypt` operation (❸) for a specific trail

(4)
and a specific log file
(5).
AWS KMS decrypted the data key under a specific KMS key
(6).

Note

You might need to scroll to the right to see some of the callouts in the following example log entry.

```
{
  "eventVersion": "1.02",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::111122223333:role/cloudtrail-
admin",1
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "cloudtrail-admin",
    "sessionContext": {"attributes": {
      "mfaAuthenticated": "false",
      "creationDate": "2015-11-11T20:48:04Z"
    }},
    "invokedBy": "signin.amazonaws.com"
  },
  "eventTime": "2015-11-11T21:20:52Z",
  "eventSource":
 "kms.amazonaws.com",2
  "eventName":
 "Decrypt",3
  "awsRegion": "us-west-2",
  "sourceIPAddress": "internal.amazonaws.com",
  "userAgent": "internal.amazonaws.com",
  "requestParameters": {
    "encryptionContext": {
      "aws:cloudtrail:arn": "arn:aws:cloudtrail:us-west-2:111122223333:trail/
Default",4
```

```
      "aws:s3:arn": "arn:aws:s3:::example-bucket-for-CT-logs/AWSLogs/111122223333/CloudTrail/us-west-2/2015/11/11/111122223333_CloudTrail_us-west-2_20151111T2115Z_7JREEBimdK8d2nC9.json.gz" 5
    }
  },
  "responseElements": null,
  "requestID": "16a0590a-88ba-11e5-b406-436f15c3ac01",
  "eventID": "9525bee7-5145-42b0-bed5-ab7196a16daa",
  "readOnly": true,
  "resources": [{
    "ARN": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab", 6
    "accountId": "111122223333"
  }],
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

How Amazon DynamoDB uses AWS KMS

Amazon DynamoDB is a fully managed, scalable NoSQL database service. DynamoDB integrates with AWS Key Management Service (AWS KMS) to support the encryption at rest server-side encryption feature.

With *encryption at rest*, DynamoDB transparently encrypts all customer data in a DynamoDB table, including its primary key and local and global secondary indexes, whenever the table is persisted to disk. (If your table has a sort key, some of the sort keys that mark range boundaries are stored in plaintext in the table metadata.) When you access your table, DynamoDB decrypts the table data transparently. You do not need to change your applications to use or manage encrypted tables.

Encryption at rest also protects DynamoDB streams, global tables, and backups whenever these objects are saved to durable media. Statements about tables in this topic apply to these objects, too.

All DynamoDB tables are encrypted. There is no option to enable or disable encryption for new or existing tables. By default, all tables are encrypted under an AWS owned key in the DynamoDB service account. However, you can select an option to encrypt some or all of your tables under a customer managed key or the AWS managed key for DynamoDB in your account.

For details about Amazon DynamoDB support for KMS keys, see DynamoDB encryption at rest in the *Amazon DynamoDB Developer Guide*.

How Amazon Elastic Block Store (Amazon EBS) uses AWS KMS

This topic discusses in detail how Amazon Elastic Block Store (Amazon EBS) uses AWS KMS to encrypt volumes and snapshots. For basic instructions about encrypting Amazon EBS volumes, see Amazon EBS Encryption.

Topics

- Amazon EBS encryption
- Using KMS keys and data keys
- Amazon EBS encryption context
- Detecting Amazon EBS failures
- Using AWS CloudFormation to create encrypted Amazon EBS volumes

Amazon EBS encryption

When you attach an encrypted Amazon EBS volume to a supported Amazon Elastic Compute Cloud (Amazon EC2) instance type, data stored at rest on the volume, disk I/O, and snapshots created from the volume are all encrypted. The encryption occurs on the servers that host Amazon EC2 instances.

This feature is supported on all Amazon EBS volume types. You access encrypted volumes the same way you access other volumes; encryption and decryption are handled transparently and they require no additional action from you, your EC2 instance, or your application. Snapshots of encrypted volumes are automatically encrypted, and volumes that are created from encrypted snapshots are also automatically encrypted.

The encryption status of an EBS volume is determined when you create the volume. You cannot change the encryption status of an existing volume. However, you can migrate data between encrypted and unencrypted volumes and apply a new encryption status while copying a snapshot.

Amazon EBS supports optional encryption by default. You can enable encryption automatically on all new EBS volumes and snapshot copies in your AWS account and Region. This configuration setting doesn't affect existing volumes or snapshots. For details, see **Encryption by default** in the Amazon EC2 User Guide or Amazon EC2 User Guide.

Using KMS keys and data keys

When you create an encrypted Amazon EBS volume, you specify an AWS KMS key. By default, Amazon EBS uses the AWS managed key for Amazon EBS in your account (`aws/ebs`). However, you can specify a customer managed key that you create and manage.

To use a customer managed key, you must give Amazon EBS permission to use the KMS key on your behalf. For a list of required permissions, see *Permissions for IAM users* in the Amazon EC2 User Guide or Amazon EC2 User Guide.

> ⚠ **Important**
>
> Amazon EBS supports only symmetric KMS keys. You cannot use an asymmetric KMS key to encrypt an Amazon EBS volume. For help determining whether a KMS key is symmetric or asymmetric, see Identifying asymmetric KMS keys.

For each volume, Amazon EBS asks AWS KMS to generate a unique data key encrypted under the KMS key that you specify. Amazon EBS stores the encrypted data key with the volume. Then, when you attach the volume to an Amazon EC2 instance, Amazon EBS calls AWS KMS to decrypt the data key. Amazon EBS uses the plaintext data key in hypervisor memory to encrypt all disk I/O to the volume. For details, see *How EBS encryption works* in the Amazon EC2 User Guide or Amazon EC2 User Guide.

Amazon EBS encryption context

In its GenerateDataKeyWithoutPlaintext and Decrypt requests to AWS KMS, Amazon EBS uses an encryption context with a name-value pair that identifies the volume or snapshot in the request. The name in the encryption context does not vary.

An encryption context is a set of key–value pairs that contain arbitrary nonsecret data. When you include an encryption context in a request to encrypt data, AWS KMS cryptographically binds the encryption context to the encrypted data. To decrypt the data, you must pass in the same encryption context.

For all volumes and for encrypted snapshots created with the Amazon EBS CreateSnapshot operation, Amazon EBS uses the volume ID as encryption context value. In the `requestParameters` field of a CloudTrail log entry, the encryption context looks similar to the following:

```
"encryptionContext": {
  "aws:ebs:id": "vol-0cfb133e847d28be9"
}
```

For encrypted snapshots created with the Amazon EC2 CopySnapshot operation, Amazon EBS uses the snapshot ID as encryption context value. In the `requestParameters` field of a CloudTrail log entry, the encryption context looks similar to the following:

```
"encryptionContext": {
  "aws:ebs:id": "snap-069a655b568de654f"
}
```

Detecting Amazon EBS failures

To create an encrypted EBS volume or attach the volume to an EC2 instance, Amazon EBS and the Amazon EC2 infrastructure must be able to use the KMS key that you specified for EBS volume encryption. When the KMS key is not usable—for example, when its key state is not `Enabled` —the volume creation or volume attachment fails.

In this case, Amazon EBS sends an *event* to Amazon EventBridge (formerly CloudWatch Events) to notify you about the failure. In EventBridge, you can establish rules that trigger automatic actions in response to these events. For more information, see Amazon CloudWatch Events for Amazon EBS in the *Amazon EC2 User Guide*, especially the following sections:

- Invalid Encryption Key on Volume Attach or Reattach
- Invalid Encryption Key on Create Volume

To fix these failures, ensure that the KMS key that you specified for EBS volume encryption is enabled. To do this, first view the KMS key to determine its current key state (the **Status** column in the AWS Management Console). Then, see the information at one of the following links:

- If the KMS key's key state is disabled, enable it.
- If the KMS key's key state is pending import, import key material.
- If the KMS key's key state is pending deletion, cancel key deletion.

Using AWS CloudFormation to create encrypted Amazon EBS volumes

You can use AWS CloudFormation to create encrypted Amazon EBS volumes. For more information, see AWS::EC2::Volume in the *AWS CloudFormation User Guide*.

How Amazon Elastic Transcoder uses AWS KMS

You can use Amazon Elastic Transcoder to convert media files stored in an Amazon S3 bucket into formats required by consumer playback devices. Both input and output files can be encrypted and decrypted. The following sections discuss how AWS KMS is used for both processes.

Topics

Encrypting the input file

Before you can use Elastic Transcoder, you must create an Amazon S3 bucket and upload your media file into it. You can encrypt the file before uploading by using AES client-side encryption or after uploading by using Amazon S3 server-side encryption.

If you choose client-side encryption using AES, you are responsible for encrypting the file before uploading it to Amazon S3, and you must provide Elastic Transcoder access to the encryption key. You do this by using a symmetric AWS KMS AWS KMS key to protect the AES encryption key you used to encrypt the media file.

If you choose server-side encryption, you allow Amazon S3 to encrypt and decrypt all files on your behalf. You can configure Amazon S3 to use one of three different types of encryption keys to protect the unique data key that encrypts your file:

- An Amazon S3 key, an encryption key that Amazon S3 owns and manages. It is not part of your AWS account.
- The AWS managed key for Amazon S3, a KMS key that is part of your account, but is created and managed by AWS

- Any symmetric customer managed key that you create by using AWS KMS

⚠ **Important**

For both client-side and server-side encryption, Elastic Transcoder supports only symmetric KMS keys. You cannot use an asymmetric KMS key to encrypt your Elastic Transcoder files. For help determining whether a KMS key is symmetric or asymmetric, see Identifying asymmetric KMS keys.

You can enable encryption and specify a key by using the Amazon S3 console or the appropriate Amazon S3 APIs. For more information about how Amazon S3 performs encryption, see Protecting data using server-side encryption with KMS keys (SSE-KMS) in the *Amazon Simple Storage Service User Guide*.

When you protect your input file by using the AWS managed key for Amazon S3 in your account or a customer managed key, Amazon S3 and AWS KMS interact in the following manner:

1. Amazon S3 requests a plaintext data key and a copy of the data key encrypted under the specified KMS key.
2. AWS KMS creates a data key, encrypts it with the specified KMS key, and then sends both the plaintext data key and the encrypted data key to Amazon S3.
3. Amazon S3 uses the plaintext data key to encrypt the media file and then stores the file in the specified Amazon S3 bucket.
4. Amazon S3 stores the encrypted data key alongside of the encrypted media file.

Decrypting the input file

If you choose Amazon S3 server-side encryption to encrypt the input file, Elastic Transcoder does not decrypt the file. Instead, Elastic Transcoder relies on Amazon S3 to perform decryption depending on the settings you specify when you create a job and a pipeline.

The following combination of settings are available.

Encryption mode	AWS KMS key	Meaning
`S3`	Default	Amazon S3 creates and manages the keys used to encrypt and decrypt the media file. The process is opaque to the user.
`S3-AWS-KMS`	Default	Amazon S3 uses a data key encrypted by the default AWS managed key for Amazon S3 in your account to encrypt the media file.
`S3-AWS-KMS`	Custom (with ARN)	Amazon S3 uses a data key encrypted by the specified customer managed key to encrypt the media file.

When `S3-AWS-KMS` is specified, Amazon S3 and AWS KMS work together in the following manner to perform the decryption.

1. Amazon S3 sends the encrypted data key to AWS KMS.
2. AWS KMS decrypts the data key by using the appropriate KMS key, and then sends the plaintext data key back to Amazon S3.
3. Amazon S3 uses the plaintext data key to decrypt the ciphertext.

If you choose client-side encryption using an AES key, Elastic Transcoder retrieves the encrypted file from the Amazon S3 bucket and decrypts it. Elastic Transcoder uses the KMS key you specified when you created the pipeline to decrypt the AES key and then uses the AES key to decrypt the media file.

Encrypting the output file

Elastic Transcoder encrypts the output file depending on how you specify the encryption settings when you create a job and a pipeline. The following options are available.

Encryption mode	AWS KMS key	Meaning
`S3`	Default	Amazon S3 creates and manages the keys used to encrypt the output file.
`S3-AWS-KMS`	Default	Amazon S3 uses a data key created by AWS KMS and encrypted by the AWS managed key for Amazon S3 in your account.
`S3-AWS-KMS`	Custom (with ARN)	Amazon S3 uses a data key encrypted by using the customer managed key specified by the ARN to encrypt the media file.
`AES-`	Default	Elastic Transcoder uses the AWS managed key for Amazon S3 in your account to decrypt the specified AES key you provide and uses that key to encrypt the output file.
`AES-`	Custom (with ARN)	Elastic Transcoder uses the customer managed key specified by the ARN to decrypt the specified AES key you provide and uses that key to encrypt the output file.

When you specify that the AWS managed key for Amazon S3 in your account or a customer managed key is used to encrypt the output file, Amazon S3 and AWS KMS interact in the following manner:

1. Amazon S3 requests a plaintext data key and a copy of the data key encrypted under the specified KMS key.
2. AWS KMS creates a data key, encrypts it under the KMS key, and sends both the plaintext data key and the encrypted data key to Amazon S3.
3. Amazon S3 encrypts the media using the data key and stores it in the specified Amazon S3 bucket.
4. Amazon S3 stores the encrypted data key alongside the encrypted media file.

When you specify that your provided AES key be used to encrypt the output file, the AES key must be encrypted using a KMS key in AWS KMS. Elastic Transcoder, AWS KMS, and you interact in the following manner:

1. You encrypt your AES key by calling the Encrypt operation in the AWS KMS API. AWS KMS encrypts the key by using the specified KMS key. You specify which KMS key to use when you are creating the pipeline.
2. You specify the file containing the encrypted AES key when you create the Elastic Transcoder job.
3. Elastic Transcoder decrypts the key by calling the Decrypt operation in the AWS KMS API, passing the encrypted key as ciphertext.
4. Elastic Transcoder uses the decrypted AES key to encrypt the output media file and then deletes the decrypted AES key from memory. Only the encrypted copy you originally defined in the job is saved to disk.
5. You can download the encrypted output file and decrypt it locally by using the original AES key that you defined.

⚠ **Important**

AWS never stores your private encryption keys. Therefore, it is important that you manage your keys safely and securely. If you lose them, you won't be able to decrypt your data.

HLS content protection

HTTP Live Streaming (HLS) is an adaptive streaming protocol. Elastic Transcoder supports HLS by breaking your input file into smaller individual files called *media segments*. A set of corresponding individual media segments contain the same material encoded at different bit rates, thereby enabling the player to select the stream that best fits the available bandwidth. Elastic Transcoder also creates playlists that contain metadata for the various segments that are available to be streamed.

When you enable *HLS content protection*, each media segment is encrypted using a 128-bit AES encryption key. When the content is viewed, during the playback process, the player downloads the key and decrypts the media segments.

Two types of keys are used: a KMS key and a data key. You must create a KMS key to use to encrypt and decrypt the data key. Elastic Transcoder uses the data key to encrypt and decrypt media segments. The data key must be AES-128. All variations and segments of the same content are encrypted using the same data key. You can provide a data key or have Elastic Transcoder create it for you.

The KMS key can be used to encrypt the data key at the following points:

- If you provide your own data key, you must encrypt it before passing it to Elastic Transcoder.
- If you request that Elastic Transcoder generate the data key, then Elastic Transcoder encrypts the data key for you.

The KMS key can be used to decrypt the data key at the following points:

- Elastic Transcoder decrypts your provided data key when it needs to use the data key to encrypt the output file or decrypt the input file.
- You decrypt a data key generated by Elastic Transcoder and use it to decrypt output files.

For more information, see HLS Content Protection in the *Amazon Elastic Transcoder Developer Guide*.

Elastic Transcoder encryption context

An encryption context is a set of key–value pairs that contain arbitrary nonsecret data. When you include an encryption context in a request to encrypt data, AWS KMS cryptographically binds the encryption context to the encrypted data. To decrypt the data, you must pass in the same encryption context.

Elastic Transcoder uses the same encryption context in all AWS KMS API requests to generate data keys, encrypt, and decrypt.

```
"service" : "elastictranscoder.amazonaws.com"
```

The encryption context is written to CloudTrail logs to help you understand how a given AWS KMS KMS key was used. In the `requestParameters` field of a CloudTrail log file, the encryption context looks similar to the following:

```
"encryptionContext": {
  "service" : "elastictranscoder.amazonaws.com"
}
```

For more information about how to configure Elastic Transcoder jobs to use one of the supported encryption options, see Data Encryption Options in the *Amazon Elastic Transcoder Developer Guide*.

How Amazon EMR uses AWS KMS

When you use an Amazon EMR cluster, you can configure the cluster to encrypt data *at rest* before saving it to a persistent storage location. You can encrypt data at rest on the EMR File System (EMRFS), on the storage volumes of cluster nodes, or both. To encrypt data at rest, you can use an AWS KMS key. The following topics explain how an Amazon EMR cluster uses a KMS key to encrypt data at rest.

> ⚠ **Important**
>
> Amazon EMR supports only symmetric KMS keys. You cannot use an asymmetric KMS key to encrypt data at rest in an Amazon EMR cluster. For help determining whether a KMS key is symmetric or asymmetric, see Identifying asymmetric KMS keys.

Amazon EMR clusters also encrypt data *in transit*, which means the cluster encrypts data before sending it through the network. You cannot use a KMS key to encrypt data in transit. For more information, see In-Transit Data Encryption in the *Amazon EMR Management Guide*.

For more information about all the encryption options available in Amazon EMR, see Encryption Options in the *Amazon EMR Management Guide*.

Topics

- Encrypting data on the EMR file system (EMRFS)

- Encrypting data on the storage volumes of cluster nodes
- Encryption context

Encrypting data on the EMR file system (EMRFS)

Amazon EMR clusters use two distributed files systems:

- The Hadoop Distributed File System (HDFS). HDFS encryption does not use a KMS key in AWS KMS.
- The EMR File System (EMRFS). EMRFS is an implementation of HDFS that allows Amazon EMR clusters to store data in Amazon Simple Storage Service (Amazon S3). EMRFS supports four encryption options, two of which use a KMS key in AWS KMS. For more information about all four of the EMRFS encryption options, see Encryption Options in the *Amazon EMR Management Guide*.

The two EMRFS encryption options that use a KMS key use the following encryption features offered by Amazon S3:

- Protecting data using server-side encryption with AWS Key Management Service (SSE-KMS). The Amazon EMR cluster sends data to Amazon S3. Amazon S3 uses a KMS key to encrypt the data before saving it to an S3 bucket. For more information about how this works, see Process for encrypting data on EMRFS with SSE-KMS.
- Protecting data using client-side encryption (CSE-KMS). Data in an Amazon EMR is encrypted under an AWS KMS key before it's sent to Amazon S3 for storage. For more information about how this works, see Process for encrypting data on EMRFS with CSE-KMS.

When you configure an Amazon EMR cluster to encrypt data on EMRFS with a KMS key, you choose the KMS key that you want Amazon S3 or the Amazon EMR cluster to use. With SSE-KMS, you can choose the AWS managed key for Amazon S3 with the alias **aws/s3**, or a symmetric customer managed key that you create. With client-side encryption, you must choose a symmetric customer managed key that you create. When you choose a customer managed key, you must ensure that your Amazon EMR cluster has permission to use the KMS key. For more information, see Using AWS KMS keys for encryption in the *Amazon EMR Management Guide*.

For both server-side and client-side encryption, the KMS key you choose is the root key in an envelope encryption workflow. The data is encrypted with a unique data key that is encrypted under the KMS key in AWS KMS. The encrypted data and an encrypted copy of its data key are stored together as a single encrypted object in an S3 bucket. For more information about how this works, see the following topics.

Topics

- Process for encrypting data on EMRFS with SSE-KMS
- Process for encrypting data on EMRFS with CSE-KMS

Process for encrypting data on EMRFS with SSE-KMS

When you configure an Amazon EMR cluster to use SSE-KMS, the encryption process works like this:

1. The cluster sends data to Amazon S3 for storage in an S3 bucket.
2. Amazon S3 sends a GenerateDataKey request to AWS KMS, specifying the key ID of the KMS key that you chose when you configured the cluster to use SSE-KMS. The request includes encryption context; for more information, see Encryption context.
3. AWS KMS generates a unique data encryption key (data key) and then sends two copies of this data key to Amazon S3. One copy is unencrypted (plaintext), and the other copy is encrypted under the KMS key.
4. Amazon S3 uses the plaintext data key to encrypt the data that it received in step 1, and then removes the plaintext data key from memory as soon as possible after use.
5. Amazon S3 stores the encrypted data and the encrypted copy of the data key together as a single encrypted object in an S3 bucket.

The decryption process works like this:

1. The cluster requests an encrypted data object from an S3 bucket.
2. Amazon S3 extracts the encrypted data key from the S3 object, and then sends the encrypted data key to AWS KMS with a Decrypt request. The request includes an encryption context.
3. AWS KMS decrypts the encrypted data key using the same KMS key that was used to encrypt it, and then sends the decrypted (plaintext) data key to Amazon S3.
4. Amazon S3 uses the plaintext data key to decrypt the encrypted data, and then removes the plaintext data key from memory as soon as possible after use.
5. Amazon S3 sends the decrypted data to the cluster.

Process for encrypting data on EMRFS with CSE-KMS

When you configure an Amazon EMR cluster to use CSE-KMS, the encryption process works like this:

1. When it's ready to store data in Amazon S3, the cluster sends a GenerateDataKey request to AWS KMS, specifying the key ID of the KMS key that you chose when you configured the cluster to use CSE-KMS. The request includes encryption context; for more information, see Encryption context.
2. AWS KMS generates a unique data encryption key (data key) and then sends two copies of this data key to the cluster. One copy is unencrypted (plaintext), and the other copy is encrypted under the KMS key.
3. The cluster uses the plaintext data key to encrypt the data, and then removes the plaintext data key from memory as soon as possible after use.
4. The cluster combines the encrypted data and the encrypted copy of the data key together into a single encrypted object.
5. The cluster sends the encrypted object to Amazon S3 for storage.

The decryption process works like this:

1. The cluster requests the encrypted data object from an S3 bucket.
2. Amazon S3 sends the encrypted object to the cluster.
3. The cluster extracts the encrypted data key from the encrypted object, and then sends the encrypted data key to AWS KMS with a Decrypt request. The request includes encryption context.
4. AWS KMS decrypts the encrypted data key using the same KMS key that was used to encrypt it, and then sends the decrypted (plaintext) data key to the cluster.
5. The cluster uses the plaintext data key to decrypt the encrypted data, and then removes the plaintext data key from memory as soon as possible after use.

Encrypting data on the storage volumes of cluster nodes

An Amazon EMR cluster is a collection of Amazon Elastic Compute Cloud (Amazon EC2) instances. Each instance in the cluster is called a *cluster node* or *node*. Each node can have two types of storage volumes: instance store volumes, and Amazon Elastic Block Store (Amazon EBS) volumes.

You can configure the cluster to use Linux Unified Key Setup (LUKS) to encrypt both types of storage volumes on the nodes (but not the boot volume of each node). This is called *local disk encryption*.

When you enable local disk encryption for a cluster, you can choose to encrypt the LUKS key with a KMS key in AWS KMS. You must choose a customer managed key that you create; you cannot use an AWS managed key. If you choose a customer managed key, you must ensure that your Amazon EMR cluster has permission to use the KMS key. For more information, see Using AWS KMS keys for encryption in the *Amazon EMR Management Guide*.

When you enable local disk encryption using a KMS key, the encryption process works like this:

1. When each cluster node launches, it sends a GenerateDataKey request to AWS KMS, specifying the key ID of the KMS key that you chose when you enabled local disk encryption for the cluster.
2. AWS KMS generates a unique data encryption key (data key) and then sends two copies of this data key to the node. One copy is unencrypted (plaintext), and the other copy is encrypted under the KMS key.
3. The node uses a base64-encoded version of the plaintext data key as the password that protects the LUKS key. The node saves the encrypted copy of the data key on its boot volume.
4. If the node reboots, the rebooted node sends the encrypted data key to AWS KMS with a Decrypt request.
5. AWS KMS decrypts the encrypted data key using the same KMS key that was used to encrypt it, and then sends the decrypted (plaintext) data key to the node.
6. The node uses the base64-encoded version of the plaintext data key as the password to unlock the LUKS key.

Encryption context

Each AWS service integrated with AWS KMS can specify an encryption context when the service uses AWS KMS to generate data keys or to encrypt or decrypt data. Encryption context is additional authenticated information that AWS KMS uses to check for data integrity. When a service specifies encryption context for an encryption operation, it must specify the same encryption context for the corresponding decryption operation or decryption will fail. Encryption context is also written to AWS CloudTrail log files, which can help you understand why a specific KMS key was used.

The following section explain the encryption context that is used in each Amazon EMR encryption scenario that uses a KMS key.

Encryption context for EMRFS encryption with SSE-KMS

With SSE-KMS, the Amazon EMR cluster sends data to Amazon S3, and then Amazon S3 uses a KMS key to encrypt the data before saving it to an S3 bucket. In this case, Amazon S3 uses the Amazon Resource Name (ARN) of the S3 object as encryption context with each GenerateDataKey and Decrypt request that it sends to AWS KMS. The following example shows a JSON representation of the encryption context that Amazon S3 uses.

```
{ "aws:s3:arn" : "arn:aws:s3:::S3_bucket_name/S3_object_key" }
```

Encryption context for EMRFS encryption with CSE-KMS

With CSE-KMS, the Amazon EMR cluster uses a KMS key to encrypt data before sending it to Amazon S3 for storage. In this case, the cluster uses the Amazon Resource Name (ARN) of the KMS key as encryption context with each GenerateDataKey and Decrypt request that it sends to AWS KMS. The following example shows a JSON representation of the encryption context that the cluster uses.

```
{ "kms_cmk_id" : "arn:aws:kms:us-east-2:111122223333:key/0987ab65-43cd-21ef-09ab-87654321cdef" }
```

Encryption context for local disk encryption with LUKS

When an Amazon EMR cluster uses local disk encryption with LUKS, the cluster nodes do not specify encryption context with the GenerateDataKey and Decrypt requests that they send to AWS KMS.

How AWS Nitro Enclaves uses AWS KMS

AWS KMS supports *cryptographic attestation* for AWS Nitro Enclaves. Applications that support AWS Nitro Enclaves call the following AWS KMS cryptographic operations with a signed attestation document for the enclave. These AWS KMS APIs verify that the attestation document came from a Nitro enclave. Then, instead of returning plaintext data in the response, these APIs encrypt the plaintext with the public key from the attestation document and return ciphertext that can be decrypted only by the corresponding private key in the enclave.

- Decrypt
- DeriveSharedSecret

- GenerateDataKey
- GenerateDataKeyPair
- GenerateRandom

The following table shows how the response to Nitro enclave requests differs from the standard response for each API operation.

AWS KMS operation	Standard response	Response for AWS Nitro Enclaves
`Decrypt`	Returns plaintext data	Returns the plaintext data encrypted by the public key from the attestation document
`DeriveSharedSecret`	Returns raw shared secret	Returns the raw shared secret encrypted by the public key from the attestation document
`GenerateDataKey`	Returns a plaintext copy of the data key (Also returns a copy of the data key encrypted by a KMS key)	Returns a copy of the data key encrypted by the public key from the attestation document (Also returns a copy of the data key encrypted by a KMS key)
`GenerateDataKeyPair`	Returns a plaintext copy of the private key (Also returns the public key and a copy of the private key encrypted by a KMS key)	Returns a copy of the private key encrypted by the public key from the attestation document (Also returns the public key and a copy of the private key encrypted by a KMS key)

AWS KMS operation	Standard response	Response for AWS Nitro Enclaves
`GenerateRandom`	Returns a random byte string	Returns the random byte string encrypted by the public key from the attestation document

AWS KMS supports policy condition keys that you can use to allow or deny enclave operations with an AWS KMS key based on the content of the attestation document. You can also monitor requests to AWS KMS for your Nitro enclave in your AWS CloudTrail logs.

Topics

- How to call AWS KMS APIs for a Nitro enclave
- AWS KMS condition keys for AWS Nitro Enclaves
- Monitoring requests for Nitro enclaves

How to call AWS KMS APIs for a Nitro enclave

To call AWS KMS APIs for a Nitro enclave, use the `Recipient` parameter in the request to provide the signed attestation document for the enclave and the encryption algorithm to use with the enclave's public key. When a request includes the `Recipient` parameter with a signed attestation document, the response includes a `CiphertextForRecipient` field with the ciphertext encrypted by the public key. The plaintext field is null or empty.

The `Recipient` parameter must specify a signed attestation document from an AWS Nitro enclave. AWS KMS relies on the digital signature for the enclave’s attestation document to prove that the public key in the request came from a valid enclave. You cannot supply your own certificate to digitally sign the attestation document.

To specify the `Recipient` parameter, use the AWS Nitro Enclaves SDK or any AWS SDK. The AWS Nitro Enclaves SDK, which is supported only within a Nitro enclave, automatically adds the `Recipient` parameter and its values to every AWS KMS request. To make requests for Nitro enclaves in the AWS SDKs, you have to specify the `Recipient` parameter and its values. Support for Nitro enclave cryptographic attestation in the AWS SDKs was introduced in March 2023.

AWS KMS supports policy condition keys that you can use to allow or deny enclave operations with an AWS KMS key based on the content of the attestation document. You can also monitor requests to AWS KMS for your Nitro enclave in your AWS CloudTrail logs.

For detailed information about the `Recipient` parameter and the AWS `CiphertextForRecipient` response field, see the Decrypt, DeriveSharedSecret, GenerateDataKey, GenerateDataKeyPair, and GenerateRandom topics in the *AWS Key Management Service API Reference*, the AWS Nitro Enclaves SDK, or any AWS SDK. For information about setting up your data and data keys for encryption, see Using cryptographic attestation with AWS KMS.

AWS KMS condition keys for AWS Nitro Enclaves

You can specify condition keys in the key policies and IAM policies that control access to your AWS KMS resources. Policy statements that includes a condition key are effective only when its conditions are satisfied.

AWS KMS provides condition keys that limit the permissions for the Decrypt, DeriveSharedSecret, GenerateDataKey, GenerateDataKeyPair, and GenerateRandom operations based on the contents of the signed attestation document in the request. These condition keys that work only when a request for an AWS KMS operation includes the `Recipient` parameter with a valid attestation document from an AWS Nitro enclave. To specify the `Recipient` parameter, use the AWS Nitro Enclaves SDK or any AWS SDK.

The enclave-specific AWS KMS condition keys are valid in key policy statements and IAM policy statements even though they do not appear in the IAM console or the IAM *Service Authorization Reference*.

kms:RecipientAttestation:ImageSha384

AWS KMS Condition Keys	Condition Type	Value type	API Operations	Policy Type

`kms:RecipientAttestation:ImageSha384`	String	Single-valued	`Decrypt`
			`DeriveSharedSecret`
			`GenerateDataKey`
				Key policies and IAM policies

AWS KMS Condition Keys	Condition Type	Value type	API Operations	Policy Type
			`GenerateDataKeyPair` `GenerateRandom`	

The `kms:RecipientAttestation:ImageSha384` condition key controls access to `Decrypt`, `DeriveSharedSecret`, `GenerateDataKey`, `GenerateDataKeyPair`, and `GenerateRandom` with a KMS key when the image digest from the signed attestation document in the request matches the value in the condition key. The `ImageSha384` value corresponds to PCR0 in the attestation document. This condition key is effective only when the `Recipient` parameter in the request specifies a signed attestation document for an AWS Nitro enclave.

This value is also included in CloudTrail events for requests to AWS KMS for Nitro enclaves.

> **Note**
>
> This condition key is valid in key policy statements and IAM policy statements even though it does not appear in the IAM console or the IAM *Service Authorization Reference*.

For example, the following key policy statement allows the `data-processing` role to use the KMS key for Decrypt, DeriveSharedSecret, GenerateDataKey, GenerateDataKeyPair, and GenerateRandom operations. The `kms:RecipientAttestation:ImageSha384` condition key allows the operations only when the image digest value (PCR0) of the attestation document in the request matches the image digest value in the condition. This condition key is effective only when the `Recipient` parameter in the request specifies a signed attestation document for an AWS Nitro enclave.

If the request does not include a valid attestation document from an AWS Nitro enclave, permission is denied because this condition is not satisfied.

```
{
  "Sid" : "Enable enclave data processing",
  "Effect" : "Allow",
  "Principal" : {
    "AWS" : "arn:aws:iam::111122223333:role/data-processing"
  },
  "Action": [
    "kms:Decrypt",
    "kms:DeriveSharedSecret",
    "kms:GenerateDataKey",
    "kms:GenerateDataKeyPair",
    "kms:GenerateRandom"
  ],
  "Resource" : "*",
  "Condition": {
    "StringEqualsIgnoreCase": {
      "kms:RecipientAttestation:ImageSha384":
 "9fedcba8abcdef7abcdef6abcdef5abcdef4abcdef3abcdef2abcdef1abcdef1abcdef0abcdef1abcdef2abcdef3a
    }
  }
}
```

kms:RecipientAttestation:PCR<PCR_ID>

AWS KMS Condition Keys	Condition Type	Value type	API Operations	Policy Type

`kms:RecipientAttestation:PCR<PCR_ID>`	String	Single-valued	`Decrypt`
			`DeriveSharedSecret`
			`GenerateDataKey`
			`GenerateDataKeyPair`
			`GenerateRandom`
				Key policies and IAM policies

The `kms:RecipientAttestation:PCR<PCR_ID>` condition key controls access to `Decrypt`, `DeriveSharedSecret`, `GenerateDataKey`, `GenerateDataKeyPair`, and `GenerateRandom` with a KMS key only when the platform configuration registers (PCRs) from the signed attestation document in the request match the PCRs in the condition key. This condition key is effective only when the `Recipient` parameter in the request specifies a signed attestation document from an AWS Nitro enclave.

This value is also included in CloudTrail events that represent requests to AWS KMS for Nitro enclaves.

> **Note**
>
> This condition key is valid in key policy statements and IAM policy statements even though it does not appear in the IAM console or the IAM *Service Authorization Reference*.

To specify a PCR value, use the following format. Concatenate the PCR ID to the condition key name. The PCR value must be a lower-case hexadecimal string of up to 96 bytes.

```
"kms:RecipientAttestation:PCRPCR_ID": "PCR_value"
```

For example, the following condition key specifies a particular value for PCR1, which corresponds to the hash of the kernel used for the enclave and the bootstrap process.

```
kms:RecipientAttestation:PCR1:
 "0x1abcdef2abcdef3abcdef4abcdef5abcdef6abcdef7abcdef8abcdef9abcdef8abcdef7abcdef6abcdef5abcdef
```

The following example key policy statement allows the `data-processing` role to use the KMS key for the Decrypt operation.

The `kms:RecipientAttestation:PCR` condition key in this statement allows the operation only when the PCR1 value in the signed attestation document in the request matches `kms:RecipientAttestation:PCR1` value in the condition. Use the `StringEqualsIgnoreCase` policy operator to require a case-insensitive comparison of the PCR values.

If the request does not include an attestation document, permission is denied because this condition is not satisfied.

```
{
  "Sid" : "Enable enclave data processing",
  "Effect" : "Allow",
  "Principal" : {
    "AWS" : "arn:aws:iam::111122223333:role/data-processing"
  },
  "Action": "kms:Decrypt",
  "Resource" : "*",
  "Condition": {
    "StringEqualsIgnoreCase": {
      "kms:RecipientAttestation:PCR1":
 "0x1de4f2dcf774f6e3b679f62e5f120065b2e408dcea327bd1c9dddaea6664e7af7935581474844767453082c6f15
    }
  }
}
```

Monitoring requests for Nitro enclaves

You can use your AWS CloudTrail logs to monitor Decrypt, DeriveSharedSecret, GenerateDataKey, GenerateDataKeyPair, and GenerateRandom operations for an AWS Nitro enclave. In these log entries, the `additionalEventData` field has a `recipient` field with the module ID (`attestationDocumentModuleId`), image digest (`attestationDocumentEnclaveImageDigest`), and platform configuration registers (PCRs) from the attestation document in the request. These fields are included only when the `Recipient` parameter in the request specifies a signed attestation document from an AWS Nitro enclave.

The module ID is the enclave ID of the Nitro enclave. The image digest is the SHA384 hash of the enclave image. You can use the image digest and PCR values in conditions for key policies and IAM policies. For information about the PCRs, see Where to get an enclave's measurements in the *AWS Nitro Enclaves User Guide*.

This section shows an example CloudTrail log entry for each of the supported Nitro enclave requests to AWS KMS.

Decrypt (for an enclave)

The following example shows an AWS CloudTrail log entry of a Decrypt operation for an AWS Nitro enclave.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2020-07-27T22:58:24Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "additionalEventData": {
        "recipient": {
            "attestationDocumentModuleId": "i-123456789abcde123-enc123456789abcde12",
            "attestationDocumentEnclaveImageDigest": "<AttestationDocument.PCR0>",
            "attestationDocumentEnclavePCR1": "<AttestationDocument.PCR1>",
            "attestationDocumentEnclavePCR2": "<AttestationDocument.PCR2>",
            "attestationDocumentEnclavePCR3": "<AttestationDocument.PCR3>",
            "attestationDocumentEnclavePCR4": "<AttestationDocument.PCR4>",
            "attestationDocumentEnclavePCR8": "<AttestationDocument.PCR8>"
        }
    },
    "requestID": "b4a65126-30d5-4b28-98b9-9153da559963",
    "eventID": "e5a2f202-ba1a-467c-b4ba-f729d45ae521",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

GenerateDataKey (for an enclave)

The following example shows an AWS CloudTrail log entry of a GenerateDataKey operation for an AWS Nitro enclave.

```
{
    "eventVersion": "1.02",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-11-04T00:52:40Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "numberOfBytes": 32
    },
    "responseElements": null,
    "additionalEventData": {
        "recipient": {
            "attestationDocumentModuleId": "i-123456789abcde123-enc123456789abcde12",
            "attestationDocumentEnclaveImageDigest": "<AttestationDocument.PCR0>",
            "attestationDocumentEnclavePCR1": "<AttestationDocument.PCR1>",
            "attestationDocumentEnclavePCR2": "<AttestationDocument.PCR2>",
            "attestationDocumentEnclavePCR3": "<AttestationDocument.PCR3>",
            "attestationDocumentEnclavePCR4": "<AttestationDocument.PCR4>",
            "attestationDocumentEnclavePCR8": "<AttestationDocument.PCR8>"
        }
    },
    "requestID": "e0eb83e3-63bc-11e4-bc2b-4198b6150d5c",
    "eventID": "a9dea4f9-8395-46c0-942c-f509c02c2b71",
    "readOnly": true,
    "resources": [{
        "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "accountId": "111122223333"
    }],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

GenerateDataKeyPair (for an enclave)

The following example shows an AWS CloudTrail log entry of a GenerateDataKeyPair operation for an AWS Nitro enclave.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::111122223333:user/Alice",
            "accountId": "111122223333",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
    },
    "eventTime": "2020-07-27T18:57:57Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKeyPair",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyPairSpec": "RSA_3072",
        "encryptionContext": {
            "Project": "Alpha"
        },
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "additionalEventData": {
        "recipient": {
            "attestationDocumentModuleId": "i-123456789abcde123-enc123456789abcde12",
            "attestationDocumentEnclaveImageDigest": "<AttestationDocument.PCR0>",
            "attestationDocumentEnclavePCR1": "<AttestationDocument.PCR1>",
            "attestationDocumentEnclavePCR2": "<AttestationDocument.PCR2>",
            "attestationDocumentEnclavePCR3": "<AttestationDocument.PCR3>",
            "attestationDocumentEnclavePCR4": "<AttestationDocument.PCR4>",
            "attestationDocumentEnclavePCR8": "<AttestationDocument.PCR8>"
        }
    },
    "requestID": "52fb127b-0fe5-42bb-8e5e-f560febde6b0",
    "eventID": "9b6bd6d2-529d-4890-a949-593b13800ad7",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

GenerateRandom (for an enclave)

The following example shows an AWS CloudTrail log entry of a GenerateRandom operation for an AWS Nitro enclave.

```
{
    "eventVersion": "1.02",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:user/Alice",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-11-04T00:52:37Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateRandom",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "AWS Internal",
    "requestParameters": null,
    "responseElements": null,
    "additionalEventData": {
        "recipient": {
            "attestationDocumentModuleId": "i-123456789abcde123-enc123456789abcde12",
            "attestationDocumentEnclaveImageDigest": "<AttestationDocument.PCR0>",
            "attestationDocumentEnclavePCR1": "<AttestationDocument.PCR1>",
            "attestationDocumentEnclavePCR2": "<AttestationDocument.PCR2>",
            "attestationDocumentEnclavePCR3": "<AttestationDocument.PCR3>",
            "attestationDocumentEnclavePCR4": "<AttestationDocument.PCR4>",
            "attestationDocumentEnclavePCR8": "<AttestationDocument.PCR8>"
        }
    },
    "requestID": "df1e3de6-63bc-11e4-bc2b-4198b6150d5c",
    "eventID": "239cb9f7-ae05-4c94-9221-6ea30eef0442",
    "readOnly": true,
    "resources": [],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

How Amazon Redshift uses AWS KMS

This topic discusses how Amazon Redshift uses AWS KMS to encrypt data.

Topics

- Amazon Redshift encryption
- Encryption context

Amazon Redshift encryption

An Amazon Redshift data warehouse is a collection of computing resources called nodes, which are organized into a group called a cluster. Each cluster runs an Amazon Redshift engine and contains one or more databases.

Amazon Redshift uses a four-tier, key-based architecture for encryption. The architecture consists of data encryption keys, a database key, a cluster key, and a root key. You can use an AWS KMS key as the root key.

Data encryption keys encrypt data blocks in the cluster. Each data block is assigned a randomly-generated AES-256 key. These keys are encrypted by using the database key for the cluster.

The database key encrypts data encryption keys in the cluster. The database key is a randomly-generated AES-256 key. It is stored on disk in a separate network from the Amazon Redshift cluster and passed to the cluster across a secure channel.

The cluster key encrypts the database key for the Amazon Redshift cluster. You can use AWS KMS, AWS CloudHSM, or an external hardware security module (HSM) to manage the cluster key. See the Amazon Redshift Database Encryption documentation for more details.

You can request encryption by checking the appropriate box in the Amazon Redshift console. You can specify a customer managed key by choosing one from the list that appears below the encryption box. If you do not specify a customer managed key, Amazon Redshift uses the AWS managed key for Amazon Redshift under your account.

⚠ **Important**

Amazon Redshift supports only symmetric encryption KMS keys. You cannot use an asymmetric KMS key in an Amazon Redshift encryption workflow. For help determining whether a KMS key is symmetric or asymmetric, see Identifying asymmetric KMS keys.

Encryption context

Each service that is integrated with AWS KMS specifies an encryption context when requesting data keys, encrypting, and decrypting. The encryption context is additional authenticated data (AAD) that AWS KMS uses to check for data integrity. That is, when an encryption context is specified for an encryption operation, the service also specifies it for the decryption operation or decryption will not succeed. Amazon Redshift uses the cluster ID and the creation time for the encryption context. In the `requestParameters` field of a CloudTrail log file, the encryption context will look similar to this.

```
"encryptionContext": {
    "aws:redshift:arn": "arn:aws:redshift:region:account_ID:cluster:cluster_name",
    "aws:redshift:createtime": "20150206T1832Z"
},
```

You can search on the cluster name in your CloudTrail logs to understand what operations were performed by using an AWS KMS key (KMS key). The operations include cluster encryption, cluster decryption, and generating data keys.

How Amazon Relational Database Service (Amazon RDS) uses AWS KMS

You can use the Amazon Relational Database Service (Amazon RDS) to set up, operate, and scale a relational database in the cloud. You can encrypt your Amazon RDS resources under an AWS managed key or customer managed key. Amazon RDS builds on Amazon Elastic Block Store (Amazon EBS) encryption to provide full disk encryption for database volumes.

For detailed information about how Amazon RDS uses KMS keys to protect your resources, see Encrypting Amazon RDS resources and AWS KMS key management in the *Amazon RDS User Guide*.

How AWS Secrets Manager uses AWS KMS

AWS Secrets Manager is an AWS service that encrypts and stores your secrets, and transparently decrypts and returns them to you in plaintext. It's designed especially to store application secrets, such as login credentials, that change periodically and should not be hard-coded or stored in plaintext in the application. In place of hard-coded credentials or table lookups, your application calls Secrets Manager.

Secrets Manager also supports features that periodically rotate the secrets associated with commonly used databases. It always encrypts newly rotated secrets before they are stored.

Secrets Manager integrates with AWS Key Management Service (AWS KMS) to encrypt every version of every secret value with a unique data key that is protected by an AWS KMS key. This integration protects your secrets under encryption keys that never leave AWS KMS unencrypted. It also enables you to set custom permissions on the KMS key and audit the operations that generate, encrypt, and decrypt the data keys that protect your secrets.

For information about how Secrets Manager uses KMS keys to protect your secrets, see Encrypting and decrypting secrets in the *AWS Secrets Manager User Guide*.

How Amazon Simple Email Service (Amazon SES) uses AWS KMS

You can use Amazon Simple Email Service (Amazon SES) to receive email, and (optionally) to encrypt the received email messages before storing them in an Amazon Simple Storage Service (Amazon S3) bucket that you choose. When you configure Amazon SES to encrypt email messages, you must choose the AWS KMS AWS KMS key under which Amazon SES encrypts the messages. You can choose the AWS managed key for Amazon SES (its alias is `aws/ses`), or you can choose a symmetric customer managed key that you created in AWS KMS.

> ⚠ **Important**
>
> Amazon SES supports only symmetric KMS keys. You cannot use an asymmetric KMS key to encrypt your Amazon SES email messages. For help determining whether a KMS key is symmetric or asymmetric, see Identifying asymmetric KMS keys.

For more information about receiving email using Amazon SES, go to Receiving Email with Amazon SES in the *Amazon Simple Email Service Developer Guide*.

Topics

Overview of Amazon SES encryption using AWS KMS

When you configure Amazon SES to receive email and encrypt the email messages before saving them to your S3 bucket, the process works like this:

1. You create a receipt rule for Amazon SES, specifying the S3 action, an S3 bucket for storage, and an AWS KMS key for encryption.
2. Amazon SES receives an email message that matches your receipt rule.
3. Amazon SES requests a unique data key encrypted with the KMS key that you specified in the applicable receipt rule.
4. AWS KMS creates a new data key, encrypts it with the specified KMS key, and then sends the encrypted and plaintext copies of the data key to Amazon SES.
5. Amazon SES uses the plaintext data key to encrypt the email message and then removes the plaintext data key from memory as soon as possible after use.
6. Amazon SES puts the encrypted email message and the encrypted data key in the specified S3 bucket. The encrypted data key is stored as metadata with the encrypted email message.

To accomplish Step 3 through Step 6, Amazon SES uses the AWS–provided Amazon S3 encryption client. Use the same client to retrieve your encrypted email messages from Amazon S3 and decrypt them. For more information, see Getting and decrypting email messages.

Amazon SES encryption context

When Amazon SES requests a data key to encrypt your received email messages (Step 3 in the Overview of Amazon SES encryption using AWS KMS), it includes an encryption context in the request. The encryption context provides additional authenticated data (AAD) that AWS KMS uses to ensure data integrity. The encryption context is also written to your AWS CloudTrail log files, which can help you understand why a given AWS KMS key (KMS key) was used. Amazon SES uses the following encryption context:

- The ID of the AWS account in which you've configured Amazon SES to receive email messages
- The rule name of the Amazon SES receipt rule that invoked the S3 action on the email message
- The Amazon SES message ID for the email message

The following example shows a JSON representation of the encryption context that Amazon SES uses:

```
{
  "aws:ses:source-account": "111122223333",
  "aws:ses:rule-name": "example-receipt-rule-name",
  "aws:ses:message-id": "d6iitobk75ur44p8kdnnp7g2n800"
}
```

Giving Amazon SES permission to use your AWS KMS key

To encrypt your email messages, you can use the AWS managed key in your account for Amazon SES (`aws/ses`), or you can use a customer managed key that you create. Amazon SES already has permission to use the AWS managed key on your behalf. However, if you specify a customer managed key when you add the S3 action to your Amazon SES receipt rule, you must give Amazon SES permission to use the KMS key to encrypt your email messages.

To give Amazon SES permission to use your customer managed key, add the following statement to that KMS key's key policy:

```
{
  "Sid": "Allow SES to encrypt messages using this KMS key",
  "Effect": "Allow",
  "Principal": {"Service": "ses.amazonaws.com"},
  "Action": [
    "kms:Encrypt",
    "kms:GenerateDataKey*"
  ],
  "Resource": "*",
  "Condition": {
    "Null": {
      "kms:EncryptionContext:aws:ses:rule-name": false,
      "kms:EncryptionContext:aws:ses:message-id": false
    },
    "StringEquals": {"kms:EncryptionContext:aws:ses:source-account": "ACCOUNT-ID-
WITHOUT-HYPHENS"}
  }
}
```

Replace *ACCOUNT-ID-WITHOUT-HYPHENS* with the 12-digit ID of the AWS account where you've configured Amazon SES to receive email messages. This policy statement allows Amazon SES to encrypt data with this KMS key only under these conditions:

- Amazon SES must specify `aws:ses:rule-name` and `aws:ses:message-id` in the `EncryptionContext` of their AWS KMS API requests.
- Amazon SES must specify `aws:ses:source-account` in the `EncryptionContext` of their AWS KMS API requests, and the value for `aws:ses:source-account` must match the AWS account ID specified in the key policy.

For more information about the encryption context that Amazon SES uses when encrypting your email messages, see Amazon SES encryption context. For general information about how AWS KMS uses the encryption context, see encryption context.

Getting and decrypting email messages

Amazon SES does not have permission to decrypt your encrypted email messages and cannot decrypt them for you. You must write code to get your email messages from Amazon S3 and decrypt them. To make this easier, use the Amazon S3 encryption client. The following AWS SDKs include the Amazon S3 encryption client:

- AWS SDK for Java – See AmazonS3EncryptionClient and AmazonS3EncryptionClientV2 in the *AWS SDK for Java API Reference*.

- AWS SDK for Ruby – See Aws::S3::Encryption::Client in the *AWS SDK for Ruby API Reference*.
- AWS SDK for .NET – See AmazonS3EncryptionClient in the *AWS SDK for .NET API Reference*.
- AWS SDK for Go – See s3crypto in the *AWS SDK for Go API Reference*.

The Amazon S3 encryption client simplifies the work of constructing the necessary requests to Amazon S3 to retrieve the encrypted email message and to AWS KMS to decrypt the message's encrypted data key, and of decrypting the email message. For example, to successfully decrypt the encrypted data key you must pass the same encryption context that Amazon SES passed when requesting the data key from AWS KMS (Step 3 in the Overview of Amazon SES encryption using AWS KMS). The Amazon S3 encryption client handles this, and much of the other work, for you.

For sample code that uses the Amazon S3 encryption client in the AWS SDK for Java to do client-side decryption, see the following:

- Using a KMS key stored in AWS KMS in the *Amazon Simple Storage Service User Guide*.
- Amazon S3 Encryption with AWS Key Management Service on the AWS Developer Blog.

How Amazon Simple Storage Service (Amazon S3) uses AWS KMS

Amazon Simple Storage Service (Amazon S3) is an object storage service that stores data as objects within buckets. Buckets and the objects in them are private and can be accessed only if you explicitly grant access permissions.

Amazon S3 integrates with AWS Key Management Service (AWS KMS) to provide server-side encryption of Amazon S3 objects. Amazon S3 uses AWS KMS keys to encrypt your Amazon S3 objects. The encryption keys that protect your objects never leave AWS KMS unencrypted. This integration also enables you to set permissions on the AWS KMS key and audit the operations that generate, encrypt, and decrypt the data keys that protect your secrets.

To reduce the volume of Amazon S3 calls to AWS KMS, use Amazon S3 bucket keys, which are KMS key-protected key-encryption-keys that are reused for a limited time within Amazon S3. Bucket keys can reduce costs for AWS KMS requests by up to 99 percent. You can configure a bucket key for all objects in an Amazon S3 bucket, or for a particular object in an Amazon S3 bucket.

For more information about how Amazon S3 uses AWS KMS, see Protecting data using server-side encryption with KMS keys (SSE-KMS) in the *Amazon S3 User Guide*.

How AWS Systems Manager Parameter Store uses AWS KMS

With AWS Systems Manager Parameter Store, you can create secure string parameters, which are parameters that have a plaintext parameter name and an encrypted parameter value. Parameter Store uses AWS KMS to encrypt and decrypt the parameter values of secure string parameters.

With Parameter Store you can create, store, and manage data as parameters with values. You can create a parameter in Parameter Store and use it in multiple applications and services subject to policies and permissions that you design. When you need to change a parameter value, you change one instance, rather than managing error-prone changes to numerous sources. Parameter Store supports a hierarchical structure for parameter names, so you can qualify a parameter for specific uses.

To manage sensitive data, you can create secure string parameters. Parameter Store uses AWS KMS keys to encrypt the parameter values of secure string parameters when you create or change them. It also uses KMS keys to decrypt the parameter values when you access them. You can use the AWS managed key that Parameter Store creates for your account or specify your own customer managed key.

> ⚠ **Important**
>
> Parameter Store supports only symmetric KMS keys. You cannot use an asymmetric KMS key to encrypt your parameters. For help determining whether a KMS key is symmetric or asymmetric, see Identifying asymmetric KMS keys.

Parameter Store supports two tiers of secure string parameters: *standard* and *advanced*. Standard parameters, which cannot exceed 4096 bytes, are encrypted and decrypted directly under the KMS key that you specify. To encrypt and decrypt advanced secure string parameters, Parameter Store uses envelope encryption with the AWS Encryption SDK. You can convert a standard secure string parameter to an advanced parameter, but you cannot convert an advanced parameter to a standard one. For more information about the difference between standard and advanced secure string parameters, see About Systems Manager Advanced Parameters in the AWS Systems Manager User Guide.

Topics

- Setting permissions to encrypt and decrypt parameter values
- Parameter Store encryption context
- Troubleshooting KMS key issues in Parameter Store

Protecting standard secure string parameters

Parameter Store does not perform any cryptographic operations. Instead, it relies on AWS KMS to encrypt and decrypt secure string parameter values. When you create or change a standard secure string parameter value, Parameter Store calls the AWS KMS Encrypt operation. This operation uses a symmetric encryption KMS key directly to encrypt the parameter value instead of using the KMS key to generate a data key.

You can select the KMS key that Parameter Store uses to encrypt the parameter value. If you do not specify a KMS key, Parameter Store uses the AWS managed key that Systems Manager automatically creates in your account. This KMS key has the `aws/ssm` alias.

To view the default `aws/ssm` KMS key for your account, use the DescribeKey operation in the AWS KMS API. The following example uses the `describe-key` command in the AWS Command Line Interface (AWS CLI) with the `aws/ssm` alias name.

```
aws kms describe-key --key-id alias/aws/ssm
```

To create a standard secure string parameter, use the PutParameter operation in the Systems Manager API. Omit the `Tier` parameter or specify a value of `Standard`, which is the default. Include a `Type` parameter with a value of `SecureString`. To specify a KMS key, use the `KeyId` parameter. The default is the AWS managed key for your account, `aws/ssm`.

Parameter Store then calls the AWS KMS `Encrypt` operation with the KMS key and the plaintext parameter value. AWS KMS returns the encrypted parameter value, which Parameter Store stores with the parameter name.

The following example uses the Systems Manager put-parameter command and its `--type` parameter in the AWS CLI to create a secure string parameter. Because the command omits the optional `--tier` and `--key-id` parameters, Parameter Store creates a standard secure string parameter and encrypts it under the AWS managed key

```
aws ssm put-parameter --name MyParameter --value "secret_value" --type SecureString
```

The following similar example uses the `--key-id` parameter to specify a customer managed key. The example uses a KMS key ID to identify the KMS key, but you can use any valid KMS key identifier. Because the command omits the `Tier` parameter (`--tier`), Parameter Store creates a standard secure string parameter, not an advanced one.

```
aws ssm put-parameter --name param1 --value "secret" --type SecureString --key-id
 1234abcd-12ab-34cd-56ef-1234567890ab
```

When you get a secure string parameter from Parameter Store, its value is encrypted. To get a parameter, use the GetParameter operation in the Systems Manager API.

The following example uses the Systems Manager get-parameter command in the AWS CLI to get the `MyParameter` parameter from Parameter Store without decrypting its value.

```
$  aws ssm get-parameter --name MyParameter

{
    "Parameter": {
        "Type": "SecureString",
        "Name": "MyParameter",
        "Value":
 "AQECAHgnOkMROh5LaLXkA4j0+vYi6tmM17Lg/9E464VRo68cvwAAAG8wbQYJKoZIhvcNAQcGoGAwXgIBADBZBgkqhkiG9
    }
}
```

To decrypt the parameter value before returning it, set the `WithDecryption` parameter of `GetParameter` to `true`. When you use `WithDecryption`, Parameter Store calls the AWS KMS Decrypt operation on your behalf to decrypt the parameter value. As a result, the `GetParameter` request returns the parameter with a plaintext parameter value, as shown in the following example.

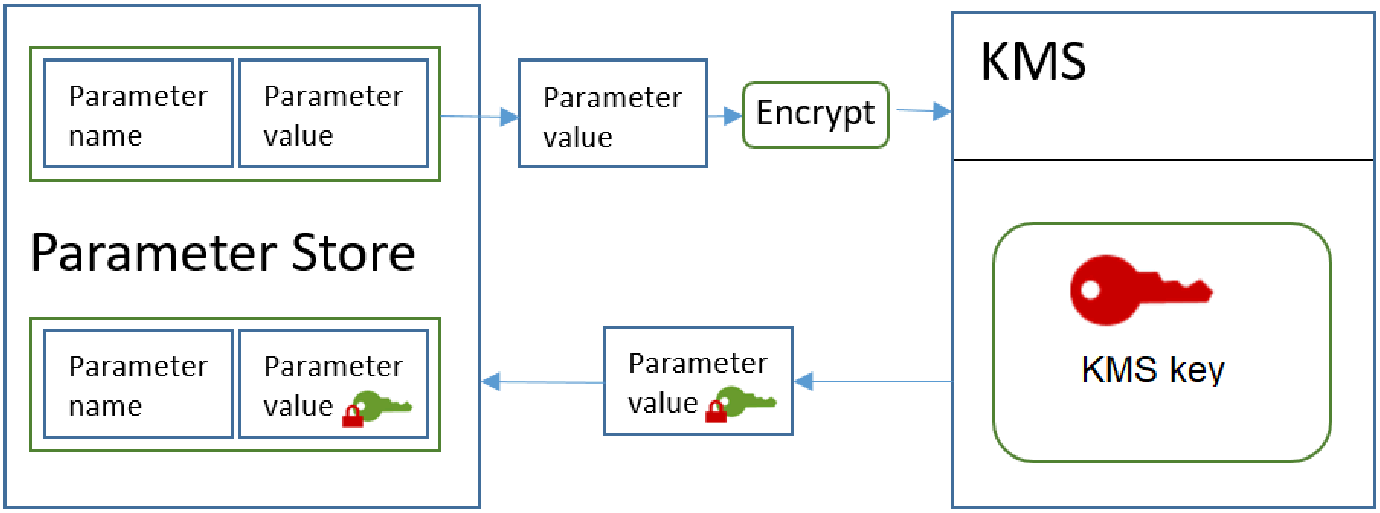
```
$  aws ssm get-parameter --name MyParameter --with-decryption

{
    "Parameter": {
        "Type": "SecureString",
        "Name": "MyParameter",
        "Value": "secret_value"
    }
}
```

The following workflow shows how Parameter Store uses a KMS key to encrypt and decrypt a standard secure string parameter.

Encrypt a standard parameter

1. When you use `PutParameter` to create a secure string parameter, Parameter Store sends an `Encrypt` request to AWS KMS. That request includes the plaintext parameter value, the KMS key that you chose, and the Parameter Store encryption context. During transmission to AWS KMS, the plaintext value in the secure string parameter is protected by Transport Layer Security (TLS).
2. AWS KMS encrypts the parameter value with the specified KMS key and encryption context. It returns the ciphertext to Parameter Store, which stores the parameter name and its encrypted value.



Decrypt a standard parameter

1. When you include the `WithDecryption` parameter in a `GetParameter` request, Parameter Store sends a `Decrypt` request to AWS KMS with the encrypted secure string parameter value and the Parameter Store encryption context.
2. AWS KMS uses the same KMS key and the supplied encryption context to decrypt the encrypted value. It returns the plaintext (decrypted) parameter value to Parameter Store. During transmission, the plaintext data is protected by TLS.
3. Parameter Store returns the plaintext parameter value to you in the `GetParameter` response.

Protecting advanced secure string parameters

When you use PutParameter to create an advanced secure string parameter, Parameter Store uses envelope encryption with the AWS Encryption SDK and a symmetric encryption AWS KMS key to protect the parameter value. Each advanced parameter value is encrypted under a unique data key, and the data key is encrypted under a KMS key. You can use the AWS managed key for the account (aws/ssm) or any customer managed key.

The AWS Encryption SDK is an open-source, client-side library that helps you to encrypt and decrypt data using industry standards and best practices. It's supported on multiple platforms and in multiple programming languages, including a command-line interface. You can view the source code and contribute to its development in GitHub.

For each secure string parameter value, Parameter Store calls the AWS Encryption SDK to encrypt the parameter value using a unique data key that AWS KMS generates (GenerateDataKey). The AWS Encryption SDK returns to Parameter Store an encrypted message that includes the encrypted parameter value and an encrypted copy of the unique data key. Parameter Store stores the entire encrypted message in the secure string parameter value. Then, when you get an advanced secure string parameter value, Parameter Store uses the AWS Encryption SDK to decrypt the parameter value. This requires a call to AWS KMS to decrypt the encrypted data key.

To create an advanced secure string parameter, use the PutParameter operation in the Systems Manager API. Set the value of Tier parameter to Advanced. Include a Type parameter with a value of SecureString. To specify a KMS key, use the KeyId parameter. The default is the AWS managed key for your account, aws/ssm.

```
aws ssm put-parameter --name MyParameter --value "secret_value" --type SecureString --
tier Advanced
```

The following similar example uses the --key-id parameter to specify a customer managed key. The example uses the Amazon Resource Name (ARN) of the KMS key, but you can use any valid KMS key identifier.

```
aws ssm put-parameter --name MyParameter --value "secret_value"
 --type SecureString --tier Advanced --key-id arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

When you get a secure string parameter from Parameter Store, its value is the encrypted message that the AWS Encryption SDK returned. To get a parameter, use the GetParameter operation in the Systems Manager API.

The following example uses the Systems Manager `GetParameter` operation to get the `MyParameter` parameter from Parameter Store without decrypting its value.

```
$  aws ssm get-parameter --name MyParameter

{
    "Parameter": {
        "Type": "SecureString",
        "Name": "MyParameter",
        "Value":
  "AQECAHgnOkMROh5LaLXkA4j0+vYi6tmM17Lg/9E464VRo68cvwAAAG8wbQYJKoZIhvcNAQcGoGAwXgIBADBZBgkqhkiG9
    }
}
```

To decrypt the parameter value before returning it, set the `WithDecryption` parameter of `GetParameter` to `true`. When you use `WithDecryption`, Parameter Store calls the AWS KMS Decrypt operation on your behalf to decrypt the parameter value. As a result, the `GetParameter` request returns the parameter with a plaintext parameter value, as shown in the following example.

```
$  aws ssm get-parameter --name MyParameter --with-decryption

{
    "Parameter": {
        "Type": "SecureString",
        "Name": "MyParameter",
        "Value": "secret_value"
    }
}
```

You cannot convert an advanced secure string parameter to a standard one, but you can convert a standard secure string to an advanced one. To convert a standard secure string parameter to an advanced secure string, use the `PutParameter` operation with the `Overwrite` parameter. The `Type` must be `SecureString` and the `Tier` value must be `Advanced`. The `KeyId` parameter, which identifies a customer managed key, is optional. If you omit it, Parameter Store uses the AWS managed key for the account. You can specify any KMS key that the principal has permission to use, even if you used a different KMS key to encrypt the standard parameter.

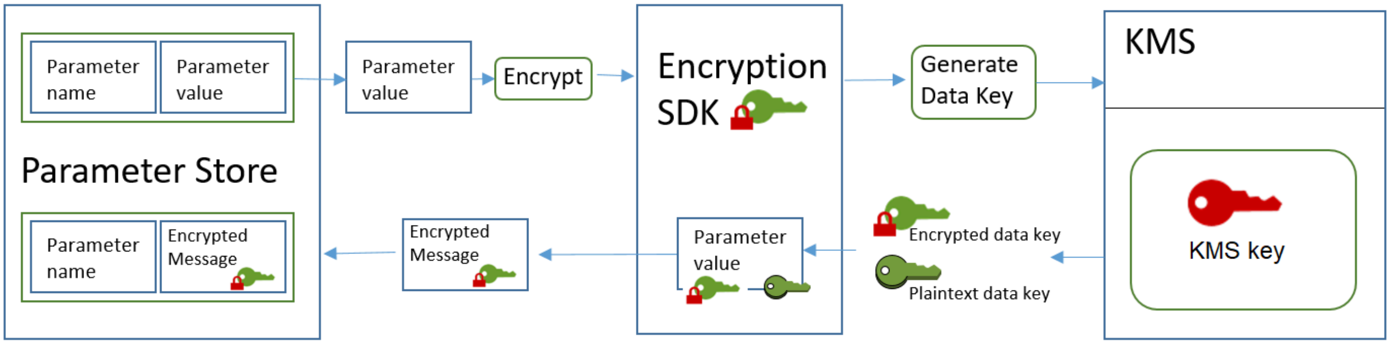
When you use the `Overwrite` parameter, Parameter Store uses the AWS Encryption SDK to encrypt the parameter value. Then it stores the newly encrypted message in Parameter Store.

```
$ aws ssm put-parameter --name myStdParameter --value "secret_value"  --type
 SecureString --tier Advanced --key-id 1234abcd-12ab-34cd-56ef-1234567890ab --overwrite
```

The following workflow shows how Parameter Store uses a KMS key to encrypt and decrypt an advanced secure string parameter.

Encrypt an advanced parameter

1. When you use `PutParameter` to create an advanced secure string parameter, Parameter Store uses the AWS Encryption SDK and AWS KMS to encrypt the parameter value. Parameter Store calls the AWS Encryption SDK with the parameter value, the KMS key that you specified, and the Parameter Store encryption context.
2. The AWS Encryption SDK sends a GenerateDataKey request to AWS KMS with the identifier of the KMS key that you specified and the Parameter Store encryption context. AWS KMS returns two copies of the unique data key: one in plaintext and one encrypted under the KMS key. (The encryption context is used when encrypting the data key.)
3. The AWS Encryption SDK uses the plaintext data key to encrypt the parameter value. It returns an encrypted message that includes the encrypted parameter value, the encrypted data key, and other data, including the Parameter Store encryption context.
4. Parameter Store stores the encrypted message as the parameter value.



Decrypt an advanced parameter

1. You can include the WithDecryption parameter in a GetParameter request to get an advanced secure string parameter. When you do, Parameter Store passes the encrypted message from the parameter value to a decryption method of the AWS Encryption SDK.
2. The AWS Encryption SDK calls the AWS KMS Decrypt operation. It passes in the encrypted data key and the Parameter Store encryption context from the encrypted message.
3. AWS KMS uses the KMS key and the Parameter Store encryption context to decrypt the encrypted data key. Then it returns the plaintext (decrypted) data key to the AWS Encryption SDK.
4. The AWS Encryption SDK uses the plaintext data key to decrypt the parameter value. It returns the plaintext parameter value to Parameter Store.
5. Parameter Store verifies the encryption context and returns the plaintext parameter value to you in the GetParameter response.

Setting permissions to encrypt and decrypt parameter values

To encrypt a standard secure string parameter value, the user needs kms:Encrypt permission. To encrypt an advanced secure string parameter value, the user needs kms:GenerateDataKey permission. To decrypt either type of secure string parameter value, the user needs kms:Decrypt permission.

You can use IAM policies to allow or deny permission for a user to call the Systems Manager PutParameter and GetParameter operations.

If you are using customer managed keys to encrypt your secure string parameter values, you can use IAM policies and key policies to manage encrypt and decrypt permissions. However, you cannot establish access control policies for the default aws/ssm KMS key. For detailed information about controlling access to customer managed keys, see Authentication and access control for AWS KMS.

The following example shows an IAM policy designed for standard secure string parameters. It allows the user to call the Systems Manager PutParameter operation on all parameters in the FinancialParameters path. The policy also allows the user to call the AWS KMS Encrypt operation on an example customer managed key.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:PutParameter"
            ],
            "Resource": "arn:aws:ssm:us-west-2:111122223333:parameter/
FinancialParameters/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Encrypt"
            ],
            "Resource": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ]
}
```

The next example shows an IAM policy that is designed for advanced secure string parameters. It allows the user to call the Systems Manager `PutParameter` operation on all parameters in the `ReservedParameters` path. The policy also allows the user to call the AWS KMS `GenerateDataKey` operation on an example customer managed key.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:PutParameter"
            ],
            "Resource": "arn:aws:ssm:us-west-2:111122223333:parameter/
ReservedParameters/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ]
}
```

The final example also shows an IAM policy that can be used for standard or advanced secure string parameters. It allows the user to call the Systems Manager `GetParameter` operations (and related operations) on all parameters in the `ITParameters` path. The policy also allows the user to call the AWS KMS `Decrypt` operation on an example customer managed key.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameter*"
            ],
            "Resource": "arn:aws:ssm:us-west-2:111122223333:parameter/ITParameters/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ]
}
```

Parameter Store encryption context

An *encryption context* is a set of key–value pairs that contain arbitrary nonsecret data. When you include an encryption context in a request to encrypt data, AWS KMS cryptographically binds the encryption context to the encrypted data. To decrypt the data, you must pass in the same encryption context.

You can also use the encryption context to identify a cryptographic operation in audit records and logs. The encryption context appears in plaintext in logs, such as AWS CloudTrail logs.

The AWS Encryption SDK also takes an encryption context, although it handles it differently. Parameter Store supplies the encryption context to the encryption method. The AWS Encryption

SDK cryptographically binds the encryption context to the encrypted data. It also includes the encryption context in plain text in the header of the encrypted message that it returns. However, unlike AWS KMS, the AWS Encryption SDK decryption methods do not take an encryption context as input. Instead, when it decrypts data, the AWS Encryption SDK gets the encryption context from the encrypted message. Parameter Store verifies that the encryption context includes the value that it expects before returning the plaintext parameter value to you.

Parameter Store uses the following encryption context in its cryptographic operations:

- Key: `PARAMETER_ARN`
- Value: The Amazon Resource Name (ARN) of the parameter that is being encrypted.

The format of the encryption context is as follows:

```
"PARAMETER_ARN":"arn:aws:ssm:<REGION_NAME>:<ACCOUNT_ID>:parameter/<parameter-name>"
```

For example, Parameter Store includes this encryption context in calls to encrypt and decrypt the `MyParameter` parameter in an example AWS account and region.

```
"PARAMETER_ARN":"arn:aws:ssm:us-west-2:111122223333:parameter/MyParameter"
```

If the parameter is in a Parameter Store hierarchical path, the path and name are included in the encryption context. For example, this encryption context is used when encrypting and decrypting the `MyParameter` parameter in the `/ReadableParameters` path in an example AWS account and region.

```
"PARAMETER_ARN":"arn:aws:ssm:us-west-2:111122223333:parameter/ReadableParameters/
MyParameter"
```

You can decrypt an encrypted secure string parameter value by calling the AWS KMS `Decrypt` operation with the correct encryption context and the encrypted parameter value that the Systems Manager `GetParameter` operation returns. However, we encourage you to decrypt Parameter Store parameter values by using the `GetParameter` operation with the `WithDecryption` parameter.

You can also include the encryption context in an IAM policy. For example, you can permit a user to decrypt only one particular parameter value or set of parameter values.

The following example IAM policy statement allows the user to the get value of the `MyParameter` parameter and to decrypt its value using the specified KMS key. However the permissions apply only when the encryption context matches specified string. These permissions do not apply to any other parameter or KMS key, and the call to `GetParameter` fails if the encryption context does not match the string.

Before using a policy statement like this one, replace the example ARNs with valid values.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameter*"
            ],
            "Resource": "arn:aws:ssm:us-west-2:111122223333:parameter/MyParameter"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:PARAMETER_ARN":"arn:aws:ssm:us-
west-2:111122223333:parameter/MyParameter"
                }
            }
        }
    ]
}
```

Troubleshooting KMS key issues in Parameter Store

To perform any operation on a secure string parameter, Parameter Store must be able to use the AWS KMS KMS key that you specify for your intended operation. Most of the Parameter Store failures related to KMS keys are caused by the following problems:

- The credentials that an application is using do not have permission to perform the specified action on the KMS key.

  To fix this error, run the application with different credentials or revise the IAM or key policy that is preventing the operation. For help with AWS KMS IAM and key policies, see Authentication and access control for AWS KMS.
- The KMS key is not found.

  This typically happens when you use an incorrect identifier for the KMS key. Find the correct identifiers for the KMS key and try the command again.
- The KMS key is not enabled. When this occurs, Parameter Store returns an InvalidKeyId exception with a detailed error message from AWS KMS. If the KMS key state is `Disabled`, enable it. If it is `Pending Import`, complete the import procedure. If the key state is `Pending Deletion`, cancel the key deletion or use a different KMS key.

  To find the key state of a KMS key in the AWS KMS console, on the **Customer managed keys** or **AWS managed keys** page, see the Status column. To use the AWS KMS API to find the status of a KMS key, use the DescribeKey operation.

How Amazon WorkMail uses AWS KMS

This topic discusses how Amazon WorkMail uses AWS KMS to encrypt email messages.

Topics

Amazon WorkMail overview

Amazon WorkMail is a secure, managed business email and calendaring service with support for existing desktop and mobile email clients. You can create an Amazon WorkMail organization and assign to it one or more email domains that you own. Then you can create mailboxes for the email users and distribution groups in the organization.

Amazon WorkMail transparently encrypts all messages in the mailboxes of all Amazon WorkMail organizations before the messages are written to disk and transparently decrypts the messages when users access them. There is no option to disable encryption. To protect the encryption keys that protect the messages, Amazon WorkMail is integrated with AWS Key Management Service (AWS KMS).

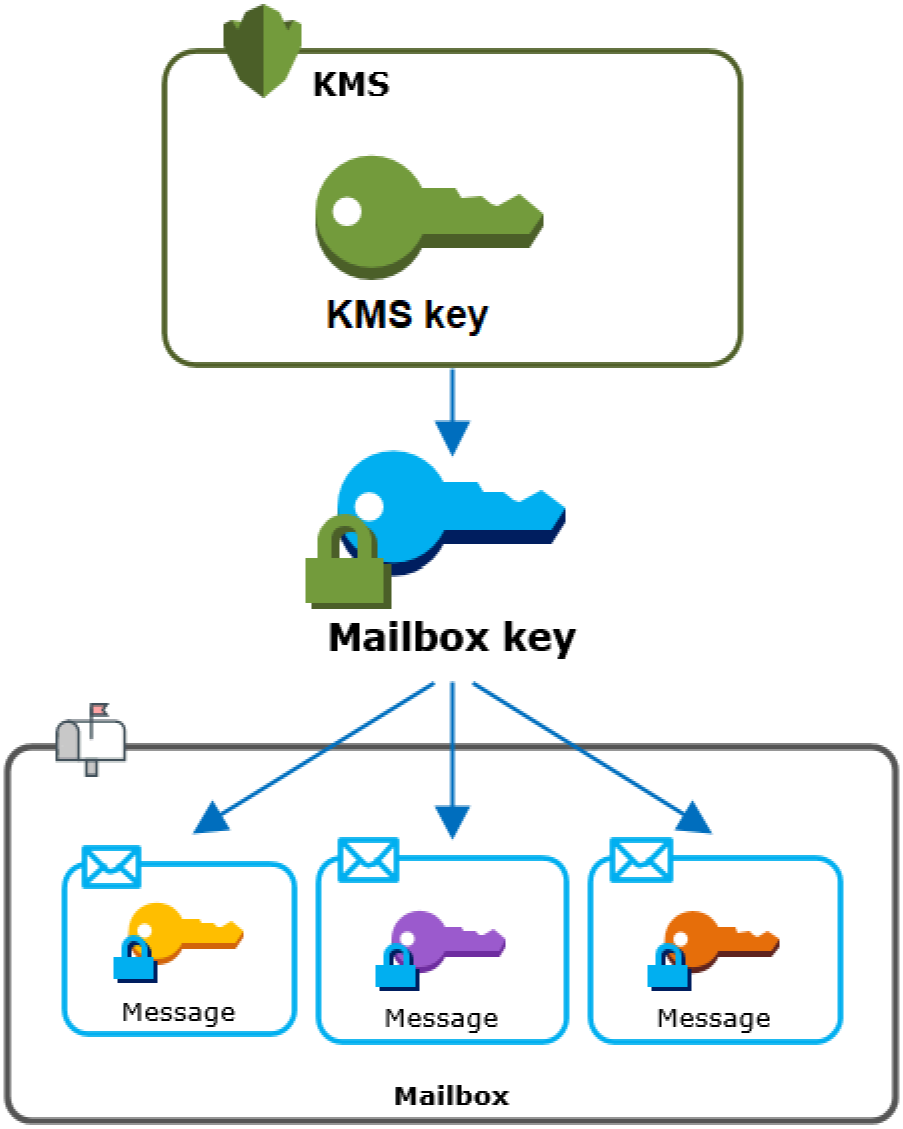
Amazon WorkMail also provides an option for enabling users to send signed or encrypted email. This encryption feature does not use AWS KMS.

Amazon WorkMail encryption

In Amazon WorkMail, each organization can contain multiple mailboxes, one for each user in the organization. All messages, including email and calendar items, are stored in the user's mailbox.

To protect the contents of the mailboxes in your Amazon WorkMail organizations, Amazon WorkMail encrypts all mailbox messages before they are written to disk. No customer-provided information is stored in plaintext.

Each message is encrypted under a unique data encryption key. The message key is protected by a mailbox key, which is a unique encryption key that is used only for that mailbox. The mailbox key is encrypted under an AWS KMS key for the organization that never leaves AWS KMS unencrypted. The following diagram shows the relationship of the encrypted messages, encrypted message keys, encrypted mailbox key, and the KMS key for the organization in AWS KMS.



A KMS key for the organization

When you create an Amazon WorkMail organization, you can select an AWS KMS keyfor the organization. This KMS key protects all mailbox keys in that organization.

If you use the Quick Setup procedure to create your organization, Amazon WorkMail uses the AWS managed key for Amazon WorkMail (`aws/workmail`) in your AWS account. If you use the Standard Setup, you can select the AWS managed key for Amazon WorkMail or a customer managed key that you own and manage. You can select the same KMS key or a different KMS key for each of your organizations, but you cannot change the KMS key once you have selected it.

Important

Amazon WorkMail supports only symmetric encryption KMS keys. You cannot use an asymmetric KMS key to encrypt data in Amazon WorkMail. For help determining whether a KMS key is symmetric or asymmetric, see Identifying asymmetric KMS keys.

To find the KMS key for your organization, use the AWS CloudTrail log entry that records calls to AWS KMS.

A unique encryption key for each mailbox

When you create a new mailbox, Amazon WorkMail generates a unique 256-bit Advanced Encryption Standard (AES) symmetric encryption key for the mailbox, known as its *mailbox key*, outside of AWS KMS. Amazon WorkMail uses the mailbox key to protect the encryption keys for each message in the mailbox.

To protect the mailbox key, Amazon WorkMail calls AWS KMS to encrypt the mailbox key under the KMS key for the organization. Then it stores the encrypted mailbox key in the mailbox metadata.

Note

Amazon WorkMail uses a symmetric mailbox encryption key to protect message keys. Previously, Amazon WorkMail protected each mailbox with an asymmetric key pair. It used the public key to encrypt each message key and the private key to decrypt it. The private mailbox key was protected by the KMS key for the organization. Existing mailboxes might still use an asymmetric mailbox key pair. This change does not affect the security of the mailbox or its messages.

A unique encryption key for each message

When a message is added to the mailbox, Amazon WorkMail generates a unique 256-bit AES symmetric encryption key for the message outside of AWS KMS. It uses this *message key* to encrypt the message. Amazon WorkMail encrypts the message key under the mailbox key and stores the encrypted message key with the message. Then, it encrypts the mailbox key under the KMS key for the organization.

Creating a new mailbox

When Amazon WorkMail creates a new mailbox, it uses the following process to prepare the mailbox to hold encrypted messages.

- Amazon WorkMail generates a unique 256-bit AES symmetric encryption key for the mailbox outside of AWS KMS.
- Amazon WorkMail calls the AWS KMS Encrypt operation. It passes in the mailbox key and the identifier of the AWS KMS key for the organization. AWS KMS returns a ciphertext of the mailbox key encrypted under the KMS key.
- Amazon WorkMail stores the encrypted mailbox key with the mailbox metadata.

Encrypting a mailbox message

To encrypt a message, Amazon WorkMail uses the following process.

1. Amazon WorkMail generates a unique 256-bit AES symmetric key for the message. It uses the plaintext message key and the Advanced Encryption Standard (AES) algorithm to encrypt the message outside of AWS KMS.
2. To protect the message key under the mailbox key, Amazon WorkMail needs to decrypt the mailbox key, which is always stored in its encrypted form.

   Amazon WorkMail calls the AWS KMS Decrypt operation and passes in the encrypted mailbox key. AWS KMS uses the KMS key for the organization to decrypt the mailbox key and it returns the plaintext mailbox key to Amazon WorkMail.
3. Amazon WorkMail uses the plaintext mailbox key and the Advanced Encryption Standard (AES) algorithm to encrypt the message key outside of AWS KMS.
4. Amazon WorkMail stores the encrypted message key in the metadata of the encrypted message so it is available to decrypt it.

Decrypting a mailbox message

To decrypt a message, Amazon WorkMail uses the following process.

1. Amazon WorkMail calls the AWS KMS Decrypt operation and passes in the encrypted mailbox key. AWS KMS uses the KMS key for the organization to decrypt the mailbox key and it returns the plaintext mailbox key to Amazon WorkMail.

2. Amazon WorkMail uses the plaintext mailbox key and the Advanced Encryption Standard (AES) algorithm to decrypt the encrypted message key outside of AWS KMS.
3. Amazon WorkMail uses the plaintext message key to decrypt the encrypted message.

Caching mailbox keys

To improve performance and minimize calls to AWS KMS, Amazon WorkMail caches each plaintext mailbox key for each client locally for up to one minute. At the end of the caching period, the mailbox key is removed. If the mailbox key for that client is required during the caching period, Amazon WorkMail can get it from the cache instead of calling AWS KMS. The mailbox key is protected in the cache and is never written to disk in plaintext.

Authorizing use of the KMS key

When Amazon WorkMail uses an AWS KMS key in cryptographic operations, it acts on behalf of the mailbox administrator.

To use the AWS KMS key for a secret on your behalf, the administrator must have the following permissions. You can specify these required permissions in an IAM policy or key policy.

- `kms:Encrypt`
- `kms:Decrypt`
- `kms:CreateGrant`

To allow the KMS key to be used only for requests that originate in Amazon WorkMail, you can use the kms:ViaService condition key with the `workmail.<region>.amazonaws.com` value.

You can also use the keys or values in the encryption context as a condition for using the KMS key for cryptographic operations. For example, you can use a string condition operator in an IAM or key policy document or use a grant constraint in a grant.

Key policy for the AWS managed key

The key policy for the AWS managed key for Amazon WorkMail gives users permission to use the KMS key for specified operations only when Amazon WorkMail makes the request on the user's behalf. The key policy does not allow any user to use the KMS key directly.

This key policy, like the policies of all AWS managed keys, is established by the service. You cannot change the key policy, but you can view it at any time. For details, see Viewing a key policy.

The policy statements in the key policy have the following effect:

- Allow users in the account and Region to use the KMS key for cryptographic operations and to create grants, but only when the request comes from Amazon WorkMail on their behalf. The `kms:ViaService` condition key enforces this restriction.
- Allows the AWS account to create IAM policies that allow users to view KMS key properties and revoke grants.

The following is a key policy for an example AWS managed key for Amazon WorkMail.

```
{
  "Version" : "2012-10-17",
  "Id" : "auto-workmail-1",
  "Statement" : [ {
    "Sid" : "Allow access through WorkMail for all principals in the account that are
 authorized to use WorkMail",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "*"
    },
    "Action" : [ "kms:Decrypt", "kms:CreateGrant", "kms:ReEncrypt*", "kms:DescribeKey",
 "kms:Encrypt" ],
    "Resource" : "*",
    "Condition" : {
      "StringEquals" : {
        "kms:ViaService" : "workmail.us-east-1.amazonaws.com",
        "kms:CallerAccount" : "111122223333"
      }
    }
  }, {
    "Sid" : "Allow direct access to key metadata to the account",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::111122223333:root"
    },
    "Action" : [ "kms:Describe*", "kms:List*", "kms:Get*", "kms:RevokeGrant" ],
    "Resource" : "*"
  } ]
}
```

Using grants to authorize Amazon WorkMail

In addition to key policies, Amazon WorkMail uses grants to add permissions to the KMS key for each organization. To view the grants on the KMS key in your account, use the ListGrants operation.

Amazon WorkMail uses grants to add the following permissions to the KMS key for the organization.

- Add the `kms:Encrypt` permission to allow Amazon WorkMail to encrypt the mailbox key.
- Add the `kms:Decrypt` permission to allow Amazon WorkMail to use the KMS key to decrypt the mailbox key. Amazon WorkMail requires this permission in a grant because the request to read mailbox messages uses the security context of the user who is reading the message. The request does not use the credentials of the AWS account. Amazon WorkMail creates this grant when you select a KMS key for the organization.

To create the grants, Amazon WorkMail calls CreateGrant on behalf of the user who created the organization. Permission to create the grant comes from the key policy. This policy allows account users to call `CreateGrant` on the KMS key for the organization when Amazon WorkMail makes the request on an authorized user's behalf.

The key policy also allows the account root to revoke the grant on the AWS managed key. However, if you revoke the grant, Amazon WorkMail cannot decrypt the encrypted data in your mailboxes.

Amazon WorkMail encryption context

An encryption context is a set of key-value pairs that contain arbitrary nonsecret data. When you include an encryption context in a request to encrypt data, AWS KMS cryptographically binds the encryption context to the encrypted data. To decrypt the data, you must pass in the same encryption context.

Amazon WorkMail uses the same encryption context format in all AWS KMS cryptographic operations. You can use the encryption context to identify a cryptographic operation in audit records and logs, such as AWS CloudTrail, and as a condition for authorization in policies and grants.

In its Encrypt and Decrypt requests to AWS KMS, Amazon WorkMail uses an encryption context where the key is `aws:workmail:arn` and the value is the Amazon Resource Name (ARN) of the organization.

```
"aws:workmail:arn":"arn:aws:workmail:region:account ID:organization/organization ID"
```

For example, the following encryption context includes an example organization ARN in the US East (Ohio) (`us-east-2`) Region.

```
"aws:workmail:arn":"arn:aws:workmail:us-east-2:111122223333:organization/
m-68755160c4cb4e29a2b2f8fb58f359d7"
```

Monitoring Amazon WorkMail interaction with AWS KMS

You can use AWS CloudTrail and Amazon CloudWatch Logs to track the requests that Amazon WorkMail sends to AWS KMS on your behalf.

Encrypt

When you create a new mailbox, Amazon WorkMail generates a mailbox key and calls AWS KMS to encrypt the mailbox key. Amazon WorkMail sends an Encrypt request to AWS KMS with the plaintext mailbox key and an identifier for the KMS key of the Amazon WorkMail organization.

The event that records the `Encrypt` operation is similar to the following example event. The user is the Amazon WorkMail service. The parameters include the KMS key ID (`keyId`) and the encryption context for the Amazon WorkMail organization. Amazon WorkMail also passes in the mailbox key, but that is not recorded in the CloudTrail log.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "workmail.eu-west-1.amazonaws.com"
    },
    "eventTime": "2019-02-19T10:01:09Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Encrypt",
    "awsRegion": "eu-west-1",
    "sourceIPAddress": "workmail.eu-west-1.amazonaws.com",
    "userAgent": "workmail.eu-west-1.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:workmail:arn": "arn:aws:workmail:eu-west-1:111122223333:organization/
m-c6981ff7642446fa8772ba99c690e455"
        },
        "keyId": "arn:aws:kms:eu-
west-1:111122223333:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d"
    },
    "responseElements": null,
    "requestID": "76e96b96-7e24-4faf-a2d6-08ded2eaf63c",
    "eventID": "d5a59c18-128a-4082-aa5b-729f7734626a",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:kms:eu-
west-1:111122223333:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333",
    "sharedEventID": "d08e60f1-097e-4a00-b7e9-10bc3872d50c"
}
```

Decrypt

When you add, view, or delete a mailbox message, Amazon WorkMail asks AWS KMS to decrypt the mailbox key. Amazon WorkMail sends an Decrypt request to AWS KMS with the encrypted mailbox key and an identifier for the KMS key of the Amazon WorkMail organization.

The event that records the `Decrypt` operation is similar to the following example event. The user is the Amazon WorkMail service. The parameters include the encrypted mailbox key (as a ciphertext blob), which is not recorded in the log, and the encryption context for the Amazon WorkMail organization. AWS KMS derives the ID of the KMS key from the ciphertext.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "workmail.eu-west-1.amazonaws.com"
    },
    "eventTime": "2019-02-20T11:51:10Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "eu-west-1",
    "sourceIPAddress": "workmail.eu-west-1.amazonaws.com",
    "userAgent": "workmail.eu-west-1.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:workmail:arn": "arn:aws:workmail:eu-west-1:111122223333:organization/
m-c6981ff7642446fa8772ba99c690e455"
        }
    },
    "responseElements": null,
    "requestID": "4a32dda1-34d9-4100-9718-674b8e0782c9",
    "eventID": "ea9fd966-98e9-4b7b-b377-6e5a397a71de",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:kms:eu-
west-1:111122223333:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333",
    "sharedEventID": "241e1e5b-ff64-427a-a5b3-7949164d0214"
}
```

How WorkSpaces uses AWS KMS

You can use WorkSpaces to provision a cloud-based desktop (a *WorkSpace*) for each of your end users. When you launch a new WorkSpace, you can choose to encrypt its volumes and decide which AWS KMS key to use for the encryption. You can choose the AWS managed key for WorkSpaces (**aws/workspaces**) or a symmetric customer managed key.

> ⚠ **Important**
>
> WorkSpaces supports only symmetric encryption KMS keys. You cannot use an asymmetric KMS key to encrypt the volumes in an WorkSpaces. For help determining whether a KMS key is symmetric or asymmetric, see Identifying asymmetric KMS keys.

For more information about creating WorkSpaces with encrypted volumes, go to Encrypt a WorkSpace in the *Amazon WorkSpaces Administration Guide*.

Topics

- WorkSpaces encryption context
- Giving WorkSpaces permission to use a KMS key on your behalf

Overview of WorkSpaces encryption using AWS KMS

When you create WorkSpaces with encrypted volumes, WorkSpaces uses Amazon Elastic Block Store (Amazon EBS) to create and manage those volumes. Both services use your AWS KMS key to work with the encrypted volumes. For more information about EBS volume encryption, see the following documentation:

- How Amazon Elastic Block Store (Amazon EBS) uses AWS KMS in this guide
- Amazon EBS Encryption in the *Amazon EC2 User Guide*

When you launch WorkSpaces with encrypted volumes, the end-to-end process works like this:

1. You specify the KMS key to use for encryption as well as the WorkSpace's user and directory. This action creates a grant that allows WorkSpaces to use your KMS key only for this WorkSpace—that is, only for the WorkSpace associated with the specified user and directory.
2. WorkSpaces creates an encrypted EBS volume for the WorkSpace and specifies the KMS key to use as well as the volume's user and directory (the same information that you specified at Step 1). This action creates a grant that allows Amazon EBS to use your KMS key only for this WorkSpace and volume—that is, only for the WorkSpace associated with the specified user and directory, and only for the specified volume.
3. Amazon EBS requests a volume data key that is encrypted under your KMS key and specifies the WorkSpace user's `Sid` and directory ID as well as the volume ID as encryption context.
4. AWS KMS creates a new data key, encrypts it under your KMS key, and then sends the encrypted data key to Amazon EBS.
5. WorkSpaces uses Amazon EBS to attach the encrypted volume to your WorkSpace. Amazon EBS sends the encrypted data key to AWS KMS with a `Decrypt` request and specifies the WorkSpace user's `Sid`, its directory ID, and the volume ID, which is used as the encryption context.
6. AWS KMS uses your KMS key to decrypt the data key, and then sends the plaintext data key to Amazon EBS.

7. Amazon EBS uses the plaintext data key to encrypt all data going to and from the encrypted volume. Amazon EBS keeps the plaintext data key in memory for as long as the volume is attached to the WorkSpace.
8. Amazon EBS stores the encrypted data key (received at Step 4) with the volume metadata for future use in case you reboot or rebuild the WorkSpace.
9. When you use the AWS Management Console to remove a WorkSpace (or use the `TerminateWorkspaces` action in the WorkSpaces API), WorkSpaces and Amazon EBS retire the grants that allowed them to use your KMS key for that WorkSpace.

WorkSpaces encryption context

WorkSpaces doesn't use your AWS KMS key directly for cryptographic operations (such as `Encrypt`, `Decrypt`, `GenerateDataKey`, etc.), which means WorkSpaces doesn't send requests to AWS KMS that include an encryption context. However, when Amazon EBS requests an encrypted data key for the encrypted volumes of your WorkSpaces (Step 3 in the Overview of WorkSpaces encryption using AWS KMS) and when it requests a plaintext copy of that data key (Step 5), it includes encryption context in the request. The encryption context provides additional authenticated data (AAD) that AWS KMS uses to ensure data integrity. The encryption context is also written to your AWS CloudTrail log files, which can help you understand why a given AWS KMS key was used. Amazon EBS uses the following for the encryption context:

- The `sid` of the AWS Directory Service user that is associated with the WorkSpace
- The directory ID of the AWS Directory Service directory that is associated with the WorkSpace
- The volume ID of the encrypted volume

The following example shows a JSON representation of the encryption context that Amazon EBS uses:

```
{
  "aws:workspaces:sid-directoryid":
 "[S-1-5-21-277731876-1789304096-451871588-1107]@[d-1234abcd01]",
  "aws:ebs:id": "vol-1234abcd"
}
```

Giving WorkSpaces permission to use a KMS key on your behalf

You can protect your workspace data under the AWS managed key for WorkSpaces (**aws/workspaces**) or a customer managed key If you use a customer managed key, you need to give WorkSpaces permission to use the KMS key on behalf of the WorkSpaces administrators in your account. The AWS managed key for WorkSpaces has the required permissions by default.

To prepare your customer managed key for use with WorkSpaces, use the following procedure.

1. Add the WorkSpaces administrators to the list of key users in the KMS key's key policy
2. Give the WorkSpaces administrators additional permissions with an IAM policy

WorkSpaces administrators also need permission to use WorkSpaces. For more information about these permissions, go to Controlling Access to WorkSpaces Resources in the *Amazon WorkSpaces Administration Guide*.

Part 1: Adding WorkSpaces administrators to a KMS key's key users

To give WorkSpaces administrators the permissions that they require, you can use the AWS Management Console or the AWS KMS API.

To add WorkSpaces administrators as key users for a KMS key (console)

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Choose the key ID or alias of your preferred customer managed key
5. Choose the **Key policy** tab. Under **Key users**, choose **Add**.
6. In the list of IAM users and roles, select the users and roles that correspond to your WorkSpaces administrators, and then choose **Attach**.

To add WorkSpaces administrators as key users for a KMS key (AWS KMS API)

1. Use the GetKeyPolicy operation to get the existing key policy, and then save the policy document to a file.

2. Open the policy document in your preferred text editor. Add the IAM users and roles that correspond to your WorkSpaces administrators to the policy statements that give permission to key users. Then save the file.
3. Use the PutKeyPolicy operation to apply the key policy to the KMS key.

Part 2: Giving WorkSpaces administrators extra permissions

If you are using a customer managed key to protect your WorkSpaces data, in addition to the permissions in the key users section of the default key policy, WorkSpaces administrators need permission to create grants on the KMS key. Also, if they use the AWS Management Console to create WorkSpaces with encrypted volumes, WorkSpaces administrators need permission to list aliases and list keys. For information about creating and editing IAM user policies, see Managed Policies and Inline Policies in the *IAM User Guide*.

To give these permissions to your WorkSpaces administrators, use an IAM policy.
Add an policy statement similar to the following example to the IAM policy for each WorkSpaces administrator. Replace the example KMS key ARN (*`arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab`*) with a valid one. If your WorkSpaces administrators use only the WorkSpaces API (not the console), you can omit the second policy statement with the "`kms:ListAliases`" and "`kms:ListKeys`" permissions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "kms:CreateGrant",
      "Resource": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:ListAliases",
        "kms:ListKeys"
      ],
      "Resource": "*"
    }
  ]
}
```

Programming the AWS KMS API

You can use the AWS KMS API to create and manage KMS keys and special features, such as custom key stores, and use KMS keys in cryptographic operations. For detailed information, see the *AWS Key Management Service API Reference*.

The sample code in the following topics show how to use the AWS SDKs to call the AWS KMS API.

For information about using the AWS KMS console to perform some of these tasks, see Managing keys.

Topics

Creating a client

To use the AWS SDK for Java, the AWS SDK for .NET, the AWS SDK for Python (Boto3), the AWS SDK for Ruby, the AWS SDK for PHP, or the AWS SDK for JavaScript in Node.js to write code that uses the AWS Key Management Service (AWS KMS) API, start by creating an AWS KMS client.

The client object that you create is used in the example code in the topics that follow.

Java

To create an AWS KMS client in Java, use the client builder.

```
AWSKMS kmsClient = AWSKMSClientBuilder.standard().build();
```

For more information about using the Java client builder, see the following resources.

- Fluent Client Builders on the AWS Developer Blog
- Creating Service Clients in the *AWS SDK for Java Developer Guide*
- AWSKMSClientBuilder in the *AWS SDK for Java API Reference*

C#

```
AmazonKeyManagementServiceClient kmsClient = new AmazonKeyManagementServiceClient();
```

Python

```
kms_client = boto3.client('kms')
```

Ruby

```
require 'aws-sdk-kms'  # in v2: require 'aws-sdk'

kmsClient = Aws::KMS::Client.new
```

PHP

To create an AWS KMS client in PHP, use an AWS KMS client object, and specify version `2014-11-01`. For more information see the KMSClient class in the AWS SDK for PHP API Reference.

```
// Create a KMSClient
$KmsClient = new Aws\Kms\KmsClient([
    'profile' => 'default',
    'version' => '2014-11-01',
    'region'  => 'us-east-1'
]);
```

Node.js

```
const kmsClient = new AWS.KMS();
```

Working with keys

The examples in this topic use the AWS KMS API to create, view, enable, and disable AWS KMS AWS KMS keys, and to generate data keys.

Topics

- Creating a KMS key
- Generating a data key
- Viewing an AWS KMS key
- Getting key IDs and key ARNs of KMS keys
- Enabling AWS KMS keys
- Disabling AWS KMS key

Creating a KMS key

To create an AWS KMS key (KMS key), use the CreateKey operation. The examples in this section create a symmetric encryption KMS key. The `Description` parameter used in these examples is optional.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

For help with creating KMS keys in the AWS KMS console, see Creating keys.

Java

For details, see the createKey method in the *AWS SDK for Java API Reference*.

```
// Create a KMS key
//
String desc = "Key for protecting critical data";

CreateKeyRequest req = new CreateKeyRequest().withDescription(desc);
CreateKeyResult result = kmsClient.createKey(req);
```

C#

For details, see the CreateKey method in the *AWS SDK for .NET*.

```
// Create a KMS key
//
String desc = "Key for protecting critical data";

CreateKeyRequest req = new CreateKeyRequest()
{
    Description = desc
};
CreateKeyResponse response = kmsClient.CreateKey(req);
```

Python

For details, see the `create_key` method in the AWS SDK for Python (Boto3).

```
# Create a KMS key

desc = 'Key for protecting critical data'

response = kms_client.create_key(
    Description=desc
)
```

Ruby

For details, see the `create_key` instance method in the AWS SDK for Ruby.

```
# Create a KMS key

desc = 'Key for protecting critical data'

response = kmsClient.create_key({
  description: desc
})
```

PHP

For details, see the CreateKey method in the *AWS SDK for PHP*.

```
// Create a KMS key
//
$desc = "Key for protecting critical data";
```

```
$result = $KmsClient->createKey([
    'Description' => $desc
]);
```

Node.js

For details, see the createKey property in the *AWS SDK for JavaScript in Node.js*.

```
// Create a KMS key
//
const Description = 'Key for protecting critical data';

kmsClient.createKey({ Description }, (err, data) => {
  ...
});
```

PowerShell

To create a KMS key in PowerShell, use the New-KmsKey cmdlet.

```
# Create a KMS key

$desc = 'Key for protecting critical data'
New-KmsKey -Description $desc
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Generating a data key

To generate a symmetric data key, use the GenerateDataKey operation. This operation returns a plaintext data key and a copy of that data key encrypted under a symmetric encryption KMS key that you specify. You must specify either a `KeySpec` or `NumberOfBytes` (but not both) in each command.

For help using the data key to encrypt data, see the AWS Encryption SDK. You can also use the data key in HMAC operations.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

Java

For details, see the generateDataKey method in the *AWS SDK for Java API Reference*.

```
// Generate a data key
//
// Replace the following example key ARN with any valid key identfier
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

GenerateDataKeyRequest dataKeyRequest = new GenerateDataKeyRequest();
dataKeyRequest.setKeyId(keyId);
dataKeyRequest.setKeySpec("AES_256");

GenerateDataKeyResult dataKeyResult = kmsClient.generateDataKey(dataKeyRequest);

ByteBuffer plaintextKey = dataKeyResult.getPlaintext();

ByteBuffer encryptedKey = dataKeyResult.getCiphertextBlob();
```

C#

For details, see the GenerateDataKey method in the *AWS SDK for .NET*.

```
// Generate a data key
//
// Replace the following example key ARN with any valid key identfier
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
GenerateDataKeyRequest dataKeyRequest = new GenerateDataKeyRequest()
{
    KeyId = keyId,
    KeySpec = DataKeySpec.AES_256
};

GenerateDataKeyResponse dataKeyResponse = kmsClient.GenerateDataKey(dataKeyRequest);

MemoryStream plaintextKey = dataKeyResponse.Plaintext;

MemoryStream encryptedKey = dataKeyResponse.CiphertextBlob;
```

Python

For details, see the [generate_data_key method] in the AWS SDK for Python (Boto3).

```
# Generate a data key

# Replace the following example key ARN with any valid key identfier
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kms_client.generate_data_key(
    KeyId=key_id,
    KeySpec='AES_256'
)

plaintext_key = response['Plaintext']

encrypted_key = response['CiphertextBlob']
```

Ruby

For details, see the generate_data_key instance method in the AWS SDK for Ruby.

```
# Generate a data key

# Replace the following example key ARN with any valid key identfier
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kmsClient.generate_data_key({
  key_id: key_id,
  key_spec: 'AES_256'
})

plaintext_key = response.plaintext

encrypted_key = response.ciphertext_blob
```

PHP

For details, see the GenerateDataKey method in the *AWS SDK for PHP*.

```
// Generate a data key
```

```
//
// Replace the following example key ARN with any valid key identfier
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$keySpec = 'AES_256';

$result = $KmsClient->generateDataKey([
    'KeyId' => $keyId,
    'KeySpec' => $keySpec,
]);

$plaintextKey = $result['Plaintext'];

$encryptedKey = $result['CiphertextBlob'];
```

Node.js

For details, see the generateDataKey property in the *AWS SDK for JavaScript in Node.js*.

```
// Generate a data key
//
// Replace the following example key ARN with any valid key identfier
const KeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
const KeySpec = 'AES_256';
kmsClient.generateDataKey({ KeyId, KeySpec }, (err, data) => {
  if (err) console.log(err, err.stack);
  else {
    const { CiphertextBlob, Plaintext } = data;
      ...
  }
});
```

PowerShell

To generate a symmetric data key, use the New-KMSDataKey cmdlet.

In the output, the plaintext key (in the `Plaintext` property) and the encrypted key (in the `CiphertextBlob` property) are MemoryStream objects. To convert them to strings, use the methods of the `MemoryStream` class, or a cmdlet or function that converts `MemoryStream` objects to strings, such as the ConvertFrom-MemoryStream and ConvertFrom-Base64 functions in the Convert module.

```
# Generate a data key

# Replace the following example key ARN with any valid key identfier
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
$keySpec = 'AES_256'

$response = New-KmsDataKey -KeyId $keyId -KeySpec $keySpec
$plaintextKey = $response.Plaintext
$encryptedKey = $response.CiphertextBlob
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Viewing an AWS KMS key

To get detailed information about an AWS KMS key, including the KMS key ARN and key state, use the DescribeKey operation.

`DescribeKey` does not get aliases. To get aliases, use the ListAliases operation. For examples, see Working with aliases.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

For help with viewing KMS keys in the AWS KMS console, see Viewing keys.

Java

For details, see the describeKey method in the *AWS SDK for Java API Reference*.

```
// Describe a KMS key
//
// Replace the following example key ARN with any valid key identfier
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

DescribeKeyRequest req = new DescribeKeyRequest().withKeyId(keyId);
DescribeKeyResult result = kmsClient.describeKey(req);
```

C#

For details, see the DescribeKey method in the *AWS SDK for .NET*.

```
// Describe a KMS key
//
// Replace the following example key ARN with any valid key identfier
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

DescribeKeyRequest describeKeyRequest = new DescribeKeyRequest()
{
    KeyId = keyId
};

DescribeKeyResponse describeKeyResponse = kmsClient.DescribeKey(describeKeyRequest);
```

Python

For details, see the `describe_key` method in the AWS SDK for Python (Boto3).

```
# Describe a KMS key

# Replace the following example key ARN with any valid key identfier
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kms_client.describe_key(
    KeyId=key_id
)
```

Ruby

For details, see the `describe_key` instance method in the AWS SDK for Ruby.

```
# Describe a KMS key

# Replace the following example key ARN with any valid key identfier
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kmsClient.describe_key({
  key_id: key_id
})
```

PHP

For details, see the DescribeKey method in the *AWS SDK for PHP*.

```
// Describe a KMS key
//
// Replace the following example key ARN with any valid key identfier
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';

$result = $KmsClient->describeKey([
    'KeyId' => $keyId,
]);
```

Node.js

For details, see the describeKey property in the *AWS SDK for JavaScript in Node.js*.

```
// Describe a KMS key
//
// Replace the following example key ARN with any valid key identfier
const KeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
kmsClient.describeKey({ KeyId }, (err, data) => {
  ...
});
```

PowerShell

To get detailed information about a KMS key, use the Get-KmsKey cmdlet.

```
# Describe a KMS key

# Replace the following example key ARN with any valid key identfier
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
Get-KmsKey -KeyId $keyId
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Getting key IDs and key ARNs of KMS keys

To get the key IDs and key ARNs of the AWS KMS keys, use the ListKeys operation. These examples use the optional `Limit` parameter, which sets the maximum number of KMS keys returned in each call. For help identifying a KMS key in an AWS KMS operations, see Key identifiers (KeyId).

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

For help with finding key IDs and key ARNs in the AWS KMS console, see Finding the key ID and key ARN.

Java

For details, see the listKeys method in the *AWS SDK for Java API Reference*.

```
// List KMS keys in this account
//
Integer limit = 10;

ListKeysRequest req = new ListKeysRequest().withLimit(limit);
ListKeysResult result = kmsClient.listKeys(req);
```

C#

For details, see the ListKeys method in the *AWS SDK for .NET*.

```
// List KMS keys in this account
//
int limit = 10;

ListKeysRequest listKeysRequest = new ListKeysRequest()
{
    Limit = limit
};
ListKeysResponse listKeysResponse = kmsClient.ListKeys(listKeysRequest);
```

Python

For details, see the `list_keys method` in the AWS SDK for Python (Boto3).

```
# List KMS keys in this account
```

```
response = kms_client.list_keys(
    Limit=10
)
```

Ruby

For details, see the `list_keys` instance method in the AWS SDK for Ruby.

```
# List KMS keys in this account

response = kmsClient.list_keys({
  limit: 10
})
```

PHP

For details, see the ListKeys method in the *AWS SDK for PHP*.

```
// List KMS keys in this account
//
$limit = 10;

$result = $KmsClient->listKeys([
    'Limit' => $limit,
]);
```

Node.js

For details, see the listKeys property in the *AWS SDK for JavaScript in Node.js*.

```
// List KMS keys in this account
//
const Limit = 10;
kmsClient.listKeys({ Limit }, (err, data) => {
  ...
});
```

PowerShell

To get the key ID and key ARN of all KMS keys in the account and Region, use the Get-KmsKeyList cmdlet.

To limit the number of output objects, this example uses the Select-Object cmdlet, instead of the Limit parameter, which is being deprecated in list cmdlets. For help with paginating output in AWS Tools for PowerShell, see Output Pagination with AWS Tools for PowerShell.

```
# List KMS keys in this account

$limit = 10
Get-KmsKeyList | Select-Object -First $limit
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Enabling AWS KMS keys

To enable a disabled AWS KMS key, use the EnableKey operation.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

For help with enabling and disabling KMS keys in the AWS KMS console, see Enabling and disabling keys.

Java

For details about the Java implementation, see the enableKey method in the *AWS SDK for Java API Reference*.

```
// Enable a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

EnableKeyRequest req = new EnableKeyRequest().withKeyId(keyId);
kmsClient.enableKey(req);
```

C#

For details, see the EnableKey method in the *AWS SDK for .NET*.

```
// Enable a KMS key
```

```
//
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

EnableKeyRequest enableKeyRequest = new EnableKeyRequest()
{
    KeyId = keyId
};
kmsClient.EnableKey(enableKeyRequest);
```

Python

For details, see the enable_key method in the AWS SDK for Python (Boto3).

```
# Enable a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kms_client.enable_key(
    KeyId=key_id
)
```

Ruby

For details, see the enable_key instance method in the AWS SDK for Ruby.

```
# Enable a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kmsClient.enable_key({
  key_id: key_id
})
```

PHP

For details, see the EnableKey method in the *AWS SDK for PHP*.

```
// Enable a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';

$result = $KmsClient->enableKey([
    'KeyId' => $keyId,
]);
```

Node.js

For details, see the enableKey property in the *AWS SDK for JavaScript in Node.js*.

```
// Enable a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
const KeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
kmsClient.enableKey({ KeyId }, (err, data) => {
  ...
});
```

PowerShell

To enable a KMS key, use the Enable-KmsKey cmdlet.

```
# Enable a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
Enable-KmsKey -KeyId $keyId
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Disabling AWS KMS key

To disable a KMS key, use the DisableKey operation. Disabling a KMS key prevents it from being used in cryptographic operations.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

For help with enabling and disabling KMS keys in the AWS KMS console, see Enabling and disabling keys.

Java

For details, see the disableKey method in the *AWS SDK for Java API Reference*.

```
// Disable a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

DisableKeyRequest req = new DisableKeyRequest().withKeyId(keyId);
kmsClient.disableKey(req);
```

C#

For details, see the DisableKey method in the *AWS SDK for .NET*.

```
// Disable a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

DisableKeyRequest disableKeyRequest = new DisableKeyRequest()
{
    KeyId = keyId
};
kmsClient.DisableKey(disableKeyRequest);
```

Python

For details, see the `disable_key` method in the AWS SDK for Python (Boto3).

```
# Disable a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
```

```
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kms_client.disable_key(
    KeyId=key_id
)
```

Ruby

For details, see the `disable_key` instance method in the AWS SDK for Ruby.

```
# Disable a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kmsClient.disable_key({
  key_id: key_id
})
```

PHP

For details, see the DisableKey method in the *AWS SDK for PHP*.

```
// Disable a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';

$result = $KmsClient->disableKey([
    'KeyId' => $keyId,
]);
```

Node.js

For details, see the disableKey property in the *AWS SDK for JavaScript in Node.js*.

```
// Disable a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
```

```
const KeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
kmsClient.disableKey({ KeyId }, (err, data) => {
  ...
});
```

PowerShell

To disable a KMS key, use the Disable-KmsKey cmdlet.

```
# Disable a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
Disable-KmsKey -KeyId $keyId
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Working with aliases

The examples in this topic use the AWS KMS API to create, view, update, and delete aliases. For information about aliases, see the section called “Using aliases”.

Topics

Creating an alias

When you create an AWS KMS key in the AWS Management Console, you must create an alias for it. However, the CreateKey operation that creates a KMS key does not create an alias.

To create an alias, use the CreateAlias operation. The alias must be unique in the account and Region. You cannot create an alias that begins with `aws/`. The `aws/` prefix is reserved by Amazon Web Services for AWS managed keys.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

Java

For details, see the createAlias method in the *AWS SDK for Java API Reference*.

```
// Create an alias for a KMS key
//
String aliasName = "alias/projectKey1";
// Replace the following example key ARN with a valid key ID or key ARN
String targetKeyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

CreateAliasRequest req = new
 CreateAliasRequest().withAliasName(aliasName).withTargetKeyId(targetKeyId);
kmsClient.createAlias(req);
```

C#

For details, see the CreateAlias method in the *AWS SDK for .NET*.

```
// Create an alias for a KMS key
//
String aliasName = "alias/projectKey1";
// Replace the following example key ARN with a valid key ID or key ARN
String targetKeyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

CreateAliasRequest createAliasRequest = new CreateAliasRequest()
{
    AliasName = aliasName,
    TargetKeyId = targetKeyId
};
kmsClient.CreateAlias(createAliasRequest);
```

Python

For details, see the `create_alias` method in the AWS SDK for Python (Boto3).

```
# Create an alias for a KMS key

alias_name = 'alias/projectKey1'
# Replace the following example key ARN with a valid key ID or key ARN
target_key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kms_client.create_alias(
    AliasName=alias_name,
    TargetKeyId=key_id
)
```

Ruby

For details, see the `create_alias` instance method in the AWS SDK for Ruby.

```
# Create an alias for a KMS key

alias_name = 'alias/projectKey1'
# Replace the following example key ARN with a valid key ID or key ARN
target_key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kmsClient.create_alias({
  alias_name: alias_name,
  target_key_id: target_key_id
})
```

PHP

For details, see the CreateAlias method in the *AWS SDK for PHP*.

```
// Create an alias for a KMS key
//
$aliasName = "alias/projectKey1";
// Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';


$result = $KmsClient->createAlias([
    'AliasName' => $aliasName,
    'TargetKeyId' =>  $keyId,
]);
```

Node.js

For details, see the createAlias property in the *AWS SDK for JavaScript in Node.js*.

```
// Create an alias for a KMS key
//
const AliasName = 'alias/projectKey1';

// Replace the following example key ARN with a valid key ID or key ARN
const TargetKeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
kmsClient.createAlias({ AliasName, TargetKeyId }, (err, data) => {
  ...
});
```

PowerShell

To create an alias, use the New-KMSAlias cmdlet. The alias name is case-sensitive.

```
# Create an alias for a KMS key

$aliasName = 'alias/projectKey1'
# Replace the following example key ARN with a valid key ID or key ARN
$targetKeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

New-KMSAlias -TargetKeyId $targetKeyId -AliasName $aliasName
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Listing aliases

To list aliases in the account and region, use the ListAliases operation.

By default, the **ListAliases** command returns all aliases in the account and Region. This includes aliases that you created and associated with your customer managed keys, and aliases that AWS created and associated with your AWS managed keys. The response might also include aliases that have no `TargetKeyId` field. These are predefined aliases that AWS has created but has not yet associated with a KMS key.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

Java

For details about the Java implementation, see the listAliases method in the *AWS SDK for Java API Reference*.

```
// List the aliases in this AWS account
//
Integer limit = 10;

ListAliasesRequest req = new ListAliasesRequest().withLimit(limit);
ListAliasesResult result = kmsClient.listAliases(req);
```

C#

For details, see the ListAliases method in the *AWS SDK for .NET*.

```
// List the aliases in this AWS account
//
int limit = 10;

ListAliasesRequest listAliasesRequest = new ListAliasesRequest()
{
    Limit = limit
};
ListAliasesResponse listAliasesResponse = kmsClient.ListAliases(listAliasesRequest);
```

Python

For details, see the `list_aliases` method in the AWS SDK for Python (Boto3).

```
# List the aliases in this AWS account

response = kms_client.list_aliases(
    Limit=10
)
```

Ruby

For details, see the `list_aliases` instance method in the AWS SDK for Ruby.

```
# List the aliases in this AWS account

response = kmsClient.list_aliases({
  limit: 10
})
```

PHP

For details, see the List Aliases method in the *AWS SDK for PHP*.

```
// List the aliases in this AWS account
//
$limit = 10;

$result = $KmsClient->listAliases([
    'Limit' => $limit,
]);
```

Node.js

For details, see the listAliases property in the *AWS SDK for JavaScript in Node.js*.

```
// List the aliases in this AWS account
//
const Limit = 10;
kmsClient.listAliases({ Limit }, (err, data) => {
  ...
});
```

PowerShell

To list the aliases in the account and Region, use the Get-KMSAliasList cmdlet.

To limit the number of output objects, this example uses the Select-Object cmdlet, instead of the `Limit` parameter, which is being deprecated in list cmdlets. For help with paginating output in AWS Tools for PowerShell, see Output Pagination with AWS Tools for PowerShell.

```
# List the aliases in this AWS account
$limit = 10

$result = Get-KMSAliasList | Select-Object -First $limit
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

To list only the aliases that are associated with a particular KMS key, use the KeyId parameter. Its value can be the key ID or key ARN of any KMS key in the region. You cannot specify an alias name or alias ARN.

Java

For details about the Java implementation, see the listAliases method in the *AWS SDK for Java API Reference*.

```
// List the aliases for one KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

ListAliasesRequest req = new ListAliasesRequest().withKeyId(keyId);
ListAliasesResult result = kmsClient.listAliases(req);
```

C#

For details, see the ListAliases method in the *AWS SDK for .NET*.

```
// List the aliases for one KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

ListAliasesRequest listAliasesRequest = new ListAliasesRequest()
{
    KeyId = keyId
};
```

```
ListAliasesResponse listAliasesResponse = kmsClient.ListAliases(listAliasesRequest);
```

Python

For details, see the list_aliases method in the AWS SDK for Python (Boto3).

```
# List the aliases for one KMS key

# Replace the following example key ARN with a valid key ID or key ARN
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kms_client.list_aliases(
    KeyId=key_id
)
```

Ruby

For details, see the list_aliases instance method in the AWS SDK for Ruby.

```
# List the aliases for one KMS key

# Replace the following example key ARN with a valid key ID or key ARN
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kmsClient.list_aliases({
  key_id: key_id
})
```

PHP

For details, see the List Aliases method in the *AWS SDK for PHP*.

```
// List the aliases for one KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';

$result = $KmsClient->listAliases([
    'KeyId' => $keyId,
]);
```

Node.js

For details, see the listAliases property in the *AWS SDK for JavaScript in Node.js*.

```
// List the aliases for one KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
const KeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
kmsClient.listAliases({ KeyId }, (err, data) => {
  ...
});
```

PowerShell

To list the aliases for a KMS key, use the `KeyId` parameter of the Get-KMSAliasList cmdlet.

```
# List the aliases for one KMS key

# Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

$response = Get-KmsAliasList -KeyId $keyId
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Updating an alias

To associate an existing alias with a different KMS key, use the UpdateAlias operation.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

Java

For details about the Java implementation, see the updateAlias method in the *AWS SDK for Java API Reference*.

```
// Updating an alias
//
String aliasName = "alias/projectKey1";
// Replace the following example key ARN with a valid key ID or key ARN
String targetKeyId = "arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321";

UpdateAliasRequest req = new UpdateAliasRequest()
     .withAliasName(aliasName)
     .withTargetKeyId(targetKeyId);

kmsClient.updateAlias(req);
```

C#

For details, see the UpdateAlias method in the *AWS SDK for .NET*.

```
// Updating an alias
//
String aliasName = "alias/projectKey1";
// Replace the following example key ARN with a valid key ID or key ARN
String targetKeyId = "arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321";

UpdateAliasRequest updateAliasRequest = new UpdateAliasRequest()
{
    AliasName = aliasName,
    TargetKeyId = targetKeyId
};

kmsClient.UpdateAlias(updateAliasRequest);
```

Python

For details, see the `update_alias method` in the AWS SDK for Python (Boto3).

```
# Updating an alias

alias_name = 'alias/projectKey1'
# Replace the following example key ARN with a valid key ID or key ARN
key_id = 'arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321'
```

```
response = kms_client.update_alias(
    AliasName=alias_name,
    TargetKeyID=key_id
)
```

Ruby

For details, see the `update_alias` instance method in the AWS SDK for Ruby.

```
# Updating an alias

alias_name = 'alias/projectKey1'
# Replace the following example key ARN with a valid key ID or key ARN
key_id = 'arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321'

response = kmsClient.update_alias({
  alias_name: alias_name,
  target_key_id: key_id
})
```

PHP

For details, see the UpdateAlias method in the *AWS SDK for PHP*.

```
// Updating an alias
//
$aliasName = "alias/projectKey1";

// Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321';

$result = $KmsClient->updateAlias([
    'AliasName' => $aliasName,
    'TargetKeyId' =>  $keyId,
]);
```

Node.js

For details, see the updateAlias property in the *AWS SDK for JavaScript in Node.js*.

```
// Updating an alias
```

```
//
const AliasName = 'alias/projectKey1';

// Replace the following example key ARN with a valid key ID or key ARN
const TargetKeyId = 'arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321';
kmsClient.updateAlias({ AliasName, TargetKeyId }, (err, data) => {
  ...
});
```

PowerShell

To change the KMS key that is associated with an alias, use the Update-KMSAlias cmdlet. The alias name is case-sensitive.

The `Update-KMSAlias` cmdlet does not return any output. To verify that the command worked, use the Get-KMSAliasList cmdlet.

```
# Updating an alias

$aliasName = 'alias/projectKey1'
# Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321'

Update-KMSAlias -AliasName $aliasName -TargetKeyID $keyId
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Deleting an alias

To delete an alias, use the DeleteAlias operation. Deleting an alias has no effect on the associated KMS key.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

Java

For details, see the deleteAlias method in the *AWS SDK for Java API Reference*.

```
// Delete an alias for a KMS key
//
String aliasName = "alias/projectKey1";

DeleteAliasRequest req  = new DeleteAliasRequest().withAliasName(aliasName);
kmsClient.deleteAlias(req);
```

C#

For details, see the DeleteAlias method in the *AWS SDK for .NET*.

```
// Delete an alias for a KMS key
//
String aliasName = "alias/projectKey1";

DeleteAliasRequest deleteAliasRequest = new DeleteAliasRequest()
{
    AliasName = aliasName
};
kmsClient.DeleteAlias(deleteAliasRequest);
```

Python

For details, see the `delete_alias` method in the AWS SDK for Python (Boto3).

```
# Delete an alias for a KMS key

alias_name = 'alias/projectKey1'

response = kms_client.delete_alias(
    AliasName=alias_name
)
```

Ruby

For details, see the `delete_alias` instance method in the AWS SDK for Ruby.

```
# Delete an alias for a KMS key

alias_name = 'alias/projectKey1'

response = kmsClient.delete_alias({
  alias_name: alias_name
})
```

PHP

For details, see the DeleteAlias method in the *AWS SDK for PHP*.

```
// Delete an alias for a KMS key
//
$aliasName = "alias/projectKey1";

$result = $KmsClient->deleteAlias([
    'AliasName' => $aliasName,
]);
```

Node.js

For details, see the deleteAlias property) in the *AWS SDK for JavaScript in Node.js*.

```
// Delete an alias for a KMS key
//
const AliasName = 'alias/projectKey1';
kmsClient.deleteAlias({ AliasName }, (err, data) => {
  ...
});
```

PowerShell

To delete an alias, use the Remove-KMSAlias cmdlet. The alias name is case-sensitive.

Because this cmdlet permanently deletes the alias, PowerShell prompts you to confirm the command. The `ConfirmImpact` is `High`, so you cannot use a `ConfirmPreference` to suppress this prompt. If you must suppress the confirmation prompt, add the `Confirm` common parameter with a value of `$false`, for example: `-Confirm:$false`.

The `Remove-KMSAlias` cmdlet doesn't return any output. To verify that the command was effective, use the Get-KMSAliasList cmdlet.

```
# Delete an alias for a KMS key

$aliasName = 'alias/projectKey1'
```

```
Remove-KMSAlias -AliasName $aliasName
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Encrypting and decrypting data keys

The examples in this topic use the Encrypt, Decrypt, and ReEncrypt operations in the AWS KMS API.

These operations are designed to encrypt and decrypt data keys. They use an AWS KMS keys in the encryption operations and they cannot accept more than 4 KB (4096 bytes) of data. Although you might use them to encrypt small amounts of data, such as a password or RSA key, they are not designed to encrypt application data.

To encrypt application data, use the server-side encryption features of an AWS service, or a client-side encryption library, such as the AWS Encryption SDK or the Amazon S3 encryption client.

Topics

Encrypting a data key

The Encrypt operation is designed to encrypt data keys, but it is not frequently used. The GenerateDataKey and GenerateDataKeyWithoutPlaintext operations return encrypted data keys. You might use this method when you are moving encrypted data to a different Region and want to encrypt its data key with a KMS key in the new Region.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

Java

For details, see the encrypt method in the *AWS SDK for Java API Reference*.

```
// Encrypt a data key
//
```

```
// Replace the following example key ARN with any valid key identfier
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
ByteBuffer plaintext = ByteBuffer.wrap(new byte[]{1,2,3,4,5,6,7,8,9,0});

EncryptRequest req = new EncryptRequest().withKeyId(keyId).withPlaintext(plaintext);
ByteBuffer ciphertext = kmsClient.encrypt(req).getCiphertextBlob();
```

C#

For details, see the Encrypt method in the *AWS SDK for .NET*.

```
// Encrypt a data key
//
// Replace the following example key ARN with any valid key identfier
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
MemoryStream plaintext = new MemoryStream();
plaintext.Write(new byte[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 }, 0, 10);

EncryptRequest encryptRequest = new EncryptRequest()
{
    KeyId = keyId,
    Plaintext = plaintext
};
MemoryStream ciphertext = kmsClient.Encrypt(encryptRequest).CiphertextBlob;
```

Python

For details, see the encrypt method in the AWS SDK for Python (Boto3).

```
# Encrypt a data key

# Replace the following example key ARN with any valid key identfier
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
plaintext = b'\x01\x02\x03\x04\x05\x06\x07\x08\x09\x00'

response = kms_client.encrypt(
    KeyId=key_id,
    Plaintext=plaintext
)
```

```
ciphertext = response['CiphertextBlob']
```

Ruby

For details, see the encrypt instance method in the AWS SDK for Ruby.

```
# Encrypt a data key

# Replace the following example key ARN with any valid key identfier
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
plaintext = "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x00"

response = kmsClient.encrypt({
  key_id: key_id,
  plaintext: plaintext
})

ciphertext = response.ciphertext_blob
```

PHP

For details, see the Encrypt method in the *AWS SDK for PHP*.

```
// Encrypt a data key
//
// Replace the following example key ARN with any valid key identfier
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$message = pack('c*',1,2,3,4,5,6,7,8,9,0);

$result = $KmsClient->encrypt([
    'KeyId' => $keyId,
    'Plaintext' => $message,
]);

$ciphertext = $result['CiphertextBlob'];
```

Node.js

For details, see the encrypt property in the AWS SDK for JavaScript in Node.js.

```
// Encrypt a data key
```

```
//
// Replace the following example key ARN with any valid key identfier
const KeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
const Plaintext = Buffer.from([1, 2, 3, 4, 5, 6, 7, 8, 9, 0]);
kmsClient.encrypt({ KeyId, Plaintext }, (err, data) => {
  if (err) console.log(err, err.stack); // an error occurred
  else {
    const { CiphertextBlob } = data;
      ...
  }
});
```

PowerShell

To encrypt a data key under a KMS key, use the Invoke-KMSEncrypt cmdlet. It returns the ciphertext as a `MemoryStream` (System.IO.MemoryStream) object. You can use the `MemoryStream` object as the input to the Invoke-KMSDecrypt cmdlet.

AWS KMS also returns data keys as `MemoryStream` objects. In this example, to simulate a plaintext data key, we create a byte array and write it to a `MemoryStream` object.

Note that the `Plaintext` parameter of `Invoke-KMSEncrypt` takes a byte array (`byte[]`); it does not require a `MemoryStream` object. Beginning in AWSPowerShell version 4.0, parameters in all AWSPowerShell modules that take byte arrays and `MemoryStream` objects accept byte arrays, `MemoryStream` objects, strings, string arrays, and `FileInfo` (System.IO.FileInfo) objects. You can pass any of these types to `Invoke-KMSEncrypt`.

```
# Encrypt a data key

# Replace the following example key ARN with any valid key identfier
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

# Simulate a data key
 # Create a byte array
[byte[]] $bytes = 1, 2, 3, 4, 5, 6, 7, 8, 9, 0

 # Create a MemoryStream
$plaintext = [System.IO.MemoryStream]::new()

 # Add the byte array to the MemoryStream
```

```
$plaintext.Write($bytes, 0, $bytes.length)

# Encrypt the simulated data key
$response = Invoke-KMSEncrypt -KeyId $keyId -Plaintext $plaintext

# Get the ciphertext from the response
$ciphertext = $response.CiphertextBlob
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Decrypting a data key

To decrypt a data key, use the Decrypt operation.

The `ciphertextBlob` that you specify must be the value of the `CiphertextBlob` field from a GenerateDataKey, GenerateDataKeyWithoutPlaintext, or Encrypt response, or the `PrivateKeyCiphertextBlob` field from a GenerateDataKeyPair or GenerateDataKeyPairWithoutPlaintext response. You can also use the `Decrypt` operation to decrypt data encrypted outside of AWS KMS by the public key in an asymmetric KMS key.

The `KeyId` parameter is not required when decrypting with symmetric encryption KMS keys. AWS KMS can get the KMS key that was used to encrypt the data from the metadata in the ciphertext blob. But it's always a best practice to specify the KMS key you are using. This practice ensures that you use the intended KMS key, and prevents you from inadvertently decrypting a ciphertext using a KMS key you do not trust.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

Java

For details, see the decrypt method in the *AWS SDK for Java API Reference*.

```
// Decrypt a data key
//
// Replace the following example key ARN with any valid key identfier
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
```

```
ByteBuffer ciphertextBlob = Place your ciphertext here;

DecryptRequest req = new
 DecryptRequest().withCiphertextBlob(ciphertextBlob).withKeyId(keyId);
ByteBuffer plainText = kmsClient.decrypt(req).getPlaintext();
```

C#

For details, see the Decrypt method in the *AWS SDK for .NET*.

```
// Decrypt a data key
//
// Replace the following example key ARN with any valid key identfier
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

MemoryStream ciphertextBlob = new MemoryStream();
// Write ciphertext to memory stream

DecryptRequest decryptRequest = new DecryptRequest()
{
    CiphertextBlob = ciphertextBlob,
    KeyId = keyId
};
MemoryStream plainText = kmsClient.Decrypt(decryptRequest).Plaintext;
```

Python

For details, see the decrypt method in the AWS SDK for Python (Boto3).

```
# Decrypt a data key

# Replace the following example key ARN with any valid key identfier
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
ciphertext = 'Place your ciphertext here'

response = kms_client.decrypt(
    CiphertextBlob=ciphertext,
    KeyId=key_id
)

plaintext = response['Plaintext']
```

Ruby

For details, see the decrypt instance method in the AWS SDK for Ruby.

```
# Decrypt a data key

# Replace the following example key ARN with any valid key identfier
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

ciphertext = 'Place your ciphertext here'
ciphertext_packed = [ciphertext].pack("H*")

response = kmsClient.decrypt({
  ciphertext_blob: ciphertext_packed,
  key_id: key_id
})

plaintext = response.plaintext
```

PHP

For details, see the Decrypt method in the *AWS SDK for PHP*.

```
// Decrypt a data key
//
// Replace the following example key ARN with any valid key identfier
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$ciphertext = 'Place your cipher text blob here';

$result = $KmsClient->decrypt([
     'CiphertextBlob' => $ciphertext,
     'KeyId' => $keyId,
]);

$plaintext = $result['Plaintext'];
```

Node.js

For details, see the decrypt property in the *AWS SDK for JavaScript in Node.js*.

```
// Decrypt a data key
```

```
//
// Replace the following example key ARN with any valid key identfier
const KeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
const CiphertextBlob = 'Place your cipher text blob here';
kmsClient.decrypt({ CiphertextBlob, KeyId }, (err, data) => {
  if (err) console.log(err, err.stack); // an error occurred
  else {
    const { Plaintext } = data;
    ...
  }
});
```

PowerShell

To decrypt a data key, use the Invoke-KMSEncrypt cmdlet.

This cmdlet returns the plaintext as a `MemoryStream` (System.IO.MemoryStream) object. To convert it to a byte array, use cmdlets or functions that convert `MemoryStream` objects to byte arrays, such as the functions in the Convert module.

Because this example uses the ciphertext that an AWS KMS encryption cmdlet returned, it uses a `MemoryStream` object for the value of the `CiphertextBlob` parameter. However, the `CiphertextBlob` parameter of `Invoke-KMSDecrypt` takes a byte array (`byte[]`); it does not require a `MemoryStream` object. Beginning in AWSPowerShell version 4.0, parameters in all AWSPowerShell modules that take byte arrays and `MemoryStream` objects accept byte arrays, `MemoryStream` objects, strings, string arrays, and `FileInfo` (System.IO.FileInfo) objects. You can pass any of these types to `Invoke-KMSDecrypt`.

```
# Decrypt a data key
# Replace the following example key ARN with any valid key identfier
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

[System.IO.MemoryStream]$ciphertext = Read-Host 'Place your cipher text blob here'

$response = Invoke-KMSDecrypt -CiphertextBlob $ciphertext -KeyId $keyId
$plaintext = $response.Plaintext
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Re-encrypting a data key under a different AWS KMS key

To decrypt an encrypted data key, and then immediately re-encrypt the data key under a different AWS KMS key, use the ReEncrypt operation. The operations are performed entirely on the server side within AWS KMS, so they never expose your plaintext outside of AWS KMS.

The `ciphertextBlob` that you specify must be the value of the `CiphertextBlob` field from a GenerateDataKey, GenerateDataKeyWithoutPlaintext, or Encrypt response, or the `PrivateKeyCiphertextBlob` field from a GenerateDataKeyPair or GenerateDataKeyPairWithoutPlaintext response. You can also use the `ReEncrypt` operation to re-encrypt data encrypted outside of AWS KMS by the public key in an asymmetric KMS key.

The `SourceKeyId` parameter is not required when re-encrypting with symmetric encryption KMS keys. AWS KMS can get the KMS key that was used to encrypt the data from the metadata in the ciphertext blob. But it's always a best practice to specify the KMS key you are using. This practice ensures that you use the intended KMS key, and prevents you from inadvertently decrypting a ciphertext using a KMS key you do not trust.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

Java

For details, see the reEncrypt method in the *AWS SDK for Java API Reference*.

```
// Re-encrypt a data key

ByteBuffer sourceCiphertextBlob = Place your ciphertext here;

// Replace the following example key ARNs with valid key identfiers
String sourceKeyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
String destinationKeyId = "arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321";

ReEncryptRequest req = new ReEncryptRequest();
req.setCiphertextBlob(sourceCiphertextBlob);
req.setSourceKeyId(sourceKeyId);
req.setDestinationKeyId(destinationKeyId);
ByteBuffer destinationCipherTextBlob = kmsClient.reEncrypt(req).getCiphertextBlob();
```

C#

For details, see the ReEncrypt method in the *AWS SDK for .NET*.

```
// Re-encrypt a data key

MemoryStream sourceCiphertextBlob = new MemoryStream();
// Write ciphertext to memory stream

// Replace the following example key ARNs with valid key identfiers
String sourceKeyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
String destinationKeyId = "arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321";

ReEncryptRequest reEncryptRequest = new ReEncryptRequest()
{
    CiphertextBlob = sourceCiphertextBlob,
    SourceKeyId = sourceKeyId,
    DestinationKeyId = destinationKeyId
};
MemoryStream destinationCipherTextBlob =
 kmsClient.ReEncrypt(reEncryptRequest).CiphertextBlob;
```

Python

For details, see the `re_encrypt` method in the AWS SDK for Python (Boto3).

```

# Re-encrypt a data key
ciphertext = 'Place your ciphertext here'

# Replace the following example key ARNs with valid key identfiers
source_key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
destination_key_id = 'arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321'

response = kms_client.re_encrypt(
    CiphertextBlob=ciphertext,
    SourceKeyId=source_key_id,
    DestinationKeyId=destination_key_id
)
```

```
destination_ciphertext_blob = response['CiphertextBlob']
```

Ruby

For details, see the `re_encrypt` instance method in the AWS SDK for Ruby.

```
# Re-encrypt a data key

ciphertext = 'Place your ciphertext here'
ciphertext_packed = [ciphertext].pack("H*")

# Replace the following example key ARNs with valid key identfiers
source_key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
destination_key_id = 'arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321'

response = kmsClient.re_encrypt({
  ciphertext_blob: ciphertext_packed,
  source_key_id: source_key_id,
  destination_key_id: destination_key_id
})

destination_ciphertext_blob = response.ciphertext_blob.unpack('H*')
```

PHP

For details, see the ReEncrypt method in the *AWS SDK for PHP*.

```
// Re-encrypt a data key

$ciphertextBlob = 'Place your ciphertext here';

// Replace the following example key ARNs with valid key identfiers
$sourceKeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$destinationKeyId = 'arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321';

$result = $KmsClient->reEncrypt([
    'CiphertextBlob' => $ciphertextBlob,
    'SourceKeyId' => $sourceKeyId,
    'DestinationKeyId' => $destinationKeyId,
]);
```

Node.js

For details, see the reEncrypt property in the *AWS SDK for JavaScript in Node.js*.

```
// Re-encrypt a data key
const CiphertextBlob = 'Place your cipher text blob here';
// Replace the following example key ARNs with valid key identfiers
const SourceKeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
const DestinationKeyId = 'arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321';

kmsClient.reEncrypt({ CiphertextBlob, SourceKeyId, DestinationKeyId }, (err, data)
 => {
  ...
});
```

PowerShell

To re-encrypt a ciphertext under the same or a different KMS key, use the Invoke-KMSReEncrypt cmdlet.

Because this example uses the ciphertext that an AWS KMS encryption cmdlet returned, it uses a `MemoryStream` object for the value of the `CiphertextBlob` parameter. However, the `CiphertextBlob` parameter of `Invoke-KMSReEncrypt` takes a byte array (`byte[]`); it does not require a `MemoryStream` object. Beginning in AWSPowerShell version 4.0, parameters in all AWSPowerShell modules that take byte arrays and `MemoryStream` objects accept byte arrays, `MemoryStream` objects, strings, string arrays, and `FileInfo` (System.IO.FileInfo) objects. You can pass any of these types to `Invoke-KMSReEncrypt`.

```
# Re-encrypt a data key

[System.IO.MemoryStream]$ciphertextBlob = Read-Host 'Place your cipher text blob
 here'

# Replace the following example key ARNs with valid key identfiers
$sourceKeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
```

```
$destinationKeyId = 'arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321'

$response = Invoke-KMSReEncrypt -Ciphertext $ciphertextBlob -SourceKeyId
 $sourceKeyId -DestinationKeyId $destinationKeyId
$reEncryptedCiphertext = $response.CiphertextBlob
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Working with key policies

The examples in this topic use the AWS KMS API to view and change the key policies of AWS KMS keys.

For details about how to use key policies, IAM policies, and grants to manage access to your KMS keys, see Authentication and access control for AWS KMS. For help writing and formatting a JSON policy document, see the IAM JSON Policy Reference in the *IAM User Guide*.

Topics

- Listing key policy names
- Getting a key policy
- Setting a key policy

Listing key policy names

To get the names of key policies for an AWS KMS key, use the ListKeyPolicies operation. The only key policy name it returns is **default**.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

Java

For details about the Java implementation, see the listKeyPolicies method in the *AWS SDK for Java API Reference*.

```
// List key policies
```

```
//
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

ListKeyPoliciesRequest req = new ListKeyPoliciesRequest().withKeyId(keyId);
ListKeyPoliciesResult result = kmsClient.listKeyPolicies(req);
```

C#

For details, see the ListKeyPolicies method in the *AWS SDK for .NET*.

```
// List key policies
//
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

ListKeyPoliciesRequest listKeyPoliciesRequest = new ListKeyPoliciesRequest()
{
    KeyId = keyId
};
ListKeyPoliciesResponse listKeyPoliciesResponse =
 kmsClient.ListKeyPolicies(listKeyPoliciesRequest);
```

Python

For details, see the `list_key_policies method` in the AWS SDK for Python (Boto3).

```
# List key policies

# Replace the following example key ARN with a valid key ID or key ARN
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kms_client.list_key_policies(
    KeyId=key_id
)
```

Ruby

For details, see the `list_key_policies` instance method in the AWS SDK for Ruby.

```
# List key policies

# Replace the following example key ARN with a valid key ID or key ARN
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kmsClient.list_key_policies({
  key_id: key_id
})
```

PHP

For details, see the ListKeyPolicies method in the *AWS SDK for PHP*.

```
// List key policies
//
// Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';

$result = $KmsClient->listKeyPolicies([
    'KeyId' => $keyId
]);
```

Node.js

For details, see the listKeyPolicies property in the *AWS SDK for JavaScript in Node.js*.

```
// List key policies
//
// Replace the following example key ARN with a valid key ID or key ARN
const KeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';

kmsClient.listKeyPolicies({ KeyId }, (err, data) => {
  ...
});
```

PowerShell

To list the name of the default key policy, use the Get-KMSKeyPolicyList cmdlet.

```
# List key policies

# Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
$response = Get-KMSKeyPolicyList -KeyId $keyId
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Getting a key policy

To get the key policy for an AWS KMS key, use the GetKeyPolicy operation.

GetKeyPolicy requires a policy name. The only valid policy name is **default**.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

Java

For details, see the getKeyPolicy method in the *AWS SDK for Java API Reference*.

```
// Get the policy for a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
String policyName = "default";

GetKeyPolicyRequest req = new
 GetKeyPolicyRequest().withKeyId(keyId).withPolicyName(policyName);
GetKeyPolicyResult result = kmsClient.getKeyPolicy(req);
```

C#

For details, see the GetKeyPolicy method in the *AWS SDK for .NET*.

```
// Get the policy for a KMS key
//
```

```
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
String policyName = "default";

GetKeyPolicyRequest getKeyPolicyRequest = new GetKeyPolicyRequest()
{
    KeyId = keyId,
    PolicyName = policyName
};
GetKeyPolicyResponse getKeyPolicyResponse =
 kmsClient.GetKeyPolicy(getKeyPolicyRequest);
```

Python

For details, see the get_key_policy method in the AWS SDK for Python (Boto3).

```
# Get the policy for a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
policy_name = 'default'

response = kms_client.get_key_policy(
    KeyId=key_id,
    PolicyName=policy_name
)
```

Ruby

For details, see the get_key_policy instance method in the AWS SDK for Ruby.

```
# Get the policy for a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
policy_name = 'default'

response = kmsClient.get_key_policy({
  key_id: key_id,
  policy_name: policy_name
})
```

PHP

For details, see the GetKeyPolicy method in the *AWS SDK for PHP*.

```
// Get the policy for a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$policyName = "default";

$result = $KmsClient->getKeyPolicy([
    'KeyId' => $keyId,
    'PolicyName' => $policyName
]);
```

Node.js

For details, see the getKeyPolicy property in the *AWS SDK for JavaScript in Node.js*.

```
// Get the policy for a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
const KeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
const PolicyName = 'default';
kmsClient.getKeyPolicy({ KeyId, PolicyName }, (err, data) => {
  ...
});
```

PowerShell

To get the key policy for a KMS key, use the Get-KMSKeyPolicy cmdlet. This cmdlet returns the key policy as a string (System.String) that you can use in a Write-KMSKeyPolicy (`PutKeyPolicy`) command. To convert the policies in the JSON string to `PSCustomObject` objects, use the ConvertFrom-JSON cmdlet.

```
# Get the policy for a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
```

```
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
$policyName = 'default'

$response = Get-KMSKeyPolicy -KeyId $keyId -PolicyName $policyName
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Setting a key policy

To create or replace the key policy for a KMS key, use the PutKeyPolicy operation.

`PutKeyPolicy` requires a policy name. The only valid policy name is **default**.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

Java

For details, see the putKeyPolicy method in the *AWS SDK for Java API Reference*.

```
// Set a key policy for a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
String policyName = "default";
String policy = "{" +
                "  \"Version\": \"2012-10-17\"," +
                "  \"Statement\": [{" +
                "    \"Sid\": \"Allow access for ExampleRole\"," +
                "    \"Effect\": \"Allow\"," +
                // Replace the following example user ARN with a valid one
                "    \"Principal\": {\"AWS\": \"arn:aws:iam::111122223333:role/
ExampleKeyUserRole\"}," +
                "    \"Action\": [" +
                "      \"kms:Encrypt\"," +
                "      \"kms:GenerateDataKey*\"," +
                "      \"kms:Decrypt\"," +
                "      \"kms:DescribeKey\"," +
                "      \"kms:ReEncrypt*\"" +
                "    ]," +
                "    \"Resource\": \"*\"" +
                "  }]" +
                "}";

PutKeyPolicyRequest req = new
 PutKeyPolicyRequest().withKeyId(keyId).withPolicy(policy).withPolicyName(policyName);
kmsClient.putKeyPolicy(req);
```

C#

For details, see the PutKeyPolicy method in the *AWS SDK for .NET*.

```
// Set a key policy for a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
String policyName = "default";
String policy = "{" +
                "  \"Version\": \"2012-10-17\"," +
                "  \"Statement\": [{" +
                "    \"Sid\": \"Allow access for ExampleUser\"," +
                "    \"Effect\": \"Allow\"," +
                // Replace the following example user ARN with a valid one
                "    \"Principal\": {\"AWS\": \"arn:aws:iam::111122223333:role/
ExampleKeyUserRole\"}," +
                "    \"Action\": [" +
                "      \"kms:Encrypt\"," +
                "      \"kms:GenerateDataKey*\"," +
                "      \"kms:Decrypt\"," +
                "      \"kms:DescribeKey\"," +
                "      \"kms:ReEncrypt*\"" +
                "    ]," +
                "    \"Resource\": \"*\"" +
                "  }]" +
                "}";

PutKeyPolicyRequest putKeyPolicyRequest = new PutKeyPolicyRequest()
{
    KeyId = keyId,
    Policy = policy,
    PolicyName = policyName
};
```

```
kmsClient.PutKeyPolicy(putKeyPolicyRequest);
```

Python

For details, see the put_key_policy method in the AWS SDK for Python (Boto3).

```
# Set a key policy for a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
policy_name = 'default'
policy = """
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "Allow access for ExampleUser",
        "Effect": "Allow",
        "Principal": {"AWS": "arn:aws:iam::111122223333:role/ExampleKeyUserRole"},
        "Action": [
            "kms:Encrypt",
            "kms:GenerateDataKey*",
            "kms:Decrypt",
            "kms:DescribeKey",
            "kms:ReEncrypt*"
        ],
        "Resource": "*"
    }]
}"""

response = kms_client.put_key_policy(
    KeyId=key_id,
    Policy=policy,
    PolicyName=policy_name
)
```

Ruby

For details, see the put_key_policy instance method in the AWS SDK for Ruby.

```
# Set a key policy for a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
```

```
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
policy_name = 'default'
policy = "{" +
  "  \"Version\": \"2012-10-17\"," +
  "  \"Statement\": [{" +
  "    \"Sid\": \"Allow access for ExampleUser\"," +
  "    \"Effect\": \"Allow\"," +
  # Replace the following example user ARN with a valid one
  "    \"Principal\": {\"AWS\": \"arn:aws:iam::111122223333:role/ExampleKeyUserRole
\"}," +
  "    \"Action\": [" +
  "      \"kms:Encrypt\"," +
  "      \"kms:GenerateDataKey*\"," +
  "      \"kms:Decrypt\"," +
  "      \"kms:DescribeKey\"," +
  "      \"kms:ReEncrypt*\"" +
  "    ]," +
  "    \"Resource\": \"*\"" +
  "  }]" +
  "}"

response = kmsClient.put_key_policy({
  key_id: key_id,
  policy: policy,
  policy_name: policy_name
})
```

PHP

For details, see the PutKeyPolicy method in the *AWS SDK for PHP*.

```
// Set a key policy for a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$policyName = "default";

$result = $KmsClient->putKeyPolicy([
    'KeyId' => $keyId,
    'PolicyName' => $policyName,
    'Policy' => '{
        "Version": "2012-10-17",
        "Id": "custom-policy-2016-12-07",
        "Statement": [
            { "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal":
               { "AWS": "arn:aws:iam::111122223333:user/root" },
            "Action": [ "kms:*" ],
            "Resource": "*" },
            { "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal":
               { "AWS": "arn:aws:iam::111122223333:role/ExampleKeyUserRole" },
            "Action": [
                "kms:Encrypt*",
                "kms:GenerateDataKey*",
                "kms:Decrypt*",
                "kms:DescribeKey*",
                "kms:ReEncrypt*"
            ],
            "Resource": "*" }
        ]
    } '
]);
```

Node.js

For details, see the putKeyPolicy property in the *AWS SDK for JavaScript in Node.js*.

```
// Set a key policy for a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
const KeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
const PolicyName = 'default';
const Policy = `{
    "Version": "2012-10-17",
    "Id": "custom-policy-2016-12-07",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/ExampleKeyUserRole"
            },
            "Action": [
                "kms:Encrypt*",
                "kms:GenerateDataKey*",
                "kms:Decrypt*",
                "kms:DescribeKey*",
                "kms:ReEncrypt*"
            ],
            "Resource": "*"
        }
    ]
}`; // The key policy document

kmsClient.putKeyPolicy({ KeyId, Policy, PolicyName }, (err, data) => {
  ...
});
```

PowerShell

To set a key policy for a KMS key, use the Write-KMSKeyPolicy cmdlet. This cmdlet doesn't return any output. To verify that the command was effective, use the Get-KMSKeyPolicy cmdlet.

The `Policy` parameter takes a string. Enclose the string in single quotes to make it a literal string. You don't have to use continuation characters or escape characters in the literal string.

```
# Set a key policy for a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
$policyName = 'default'
$policy = '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/ExampleKeyUserRole"
            },
            "Action": [
                "kms:Encrypt*",
                "kms:GenerateDataKey*",
                "kms:Decrypt*",
                "kms:DescribeKey*",
                "kms:ReEncrypt*"
            ],
            "Resource": "*"
        }]
    }'

Write-KMSKeyPolicy -KeyId $keyId -PolicyName $policyName -Policy $policy
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Working with grants

The examples in this topic use the AWS KMS API to create, view, retire, and revoke grants on AWS KMS keys. For more details about using grants in AWS KMS, see Grants in AWS KMS.

Topics

- Creating a grant
- Viewing a grant
- Retiring a grant
- Revoking a grant

Creating a grant

To create a grant for an AWS KMS key, use the CreateGrant operation. The response includes only the grant ID and grant token. To get detailed information about the grant, use the ListGrants operation, as shown in Viewing a grant.

These examples create a grant that allows users who can assume the `ExampleKeyUser` role to call the GenerateDataKey operation on the KMS key identified by the `KeyId` parameter.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

Java

For details, see the createGrant method in the *AWS SDK for Java API Reference*.

```
// Create a grant
//
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
String granteePrincipal = "arn:aws:iam::111122223333:role/ExampleKeyUser";
String operation = GrantOperation.GenerateDataKey.toString();

CreateGrantRequest request = new CreateGrantRequest()
    .withKeyId(keyId)
    .withGranteePrincipal(granteePrincipal)
    .withOperations(operation);

CreateGrantResult result = kmsClient.createGrant(request);
```

C#

For details, see the CreateGrant method in the *AWS SDK for .NET*.

```
// Create a grant
//
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
String granteePrincipal = "arn:aws:iam::111122223333:role/ExampleKeyUser";
String operation = GrantOperation.GenerateDataKey;
```

```
CreateGrantRequest createGrantRequest = new CreateGrantRequest()
{
    KeyId = keyId,
    GranteePrincipal = granteePrincipal,
    Operations = new List<string>() { operation }
};

CreateGrantResponse createGrantResult = kmsClient.CreateGrant(createGrantRequest);
```

Python

For details, see the create_grant method in the AWS SDK for Python (Boto3).

```
# Create a grant

# Replace the following example key ARN with a valid key ID or key ARN
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
grantee_principal = 'arn:aws:iam::111122223333:role/ExampleKeyUser'
operation = ['GenerateDataKey']

response = kms_client.create_grant(
    KeyId=key_id,
    GranteePrincipal=grantee_principal,
    Operations=operation
)
```

Ruby

For details, see the create_grant instance method in the AWS SDK for Ruby.

```
# Create a grant

# Replace the following example key ARN with a valid key ID or key ARN
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
grantee_principal = 'arn:aws:iam::111122223333:role/ExampleKeyUser'
operation = ['GenerateDataKey']

response = kmsClient.create_grant({
  key_id: key_id,
  grantee_principal: grantee_principal,
  operations: operation
})
```

PHP

For details, see the CreateGrant method in the *AWS SDK for PHP*.

```
// Create a grant
//
// Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$granteePrincipal = "arn:aws:iam::111122223333:role/ExampleKeyUser";
$operation = ['GenerateDataKey']

$result = $KmsClient->createGrant([
    'GranteePrincipal' => $granteePrincipal,
    'KeyId' => $keyId,
    'Operations' => $operation
]);
```

Node.js

For details, see the createGrant property in the *AWS SDK for JavaScript in Node.js*.

```
// Create a grant
//
// Replace the following example key ARN with a valid key ID or key ARN
const KeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
const GranteePrincipal = 'arn:aws:iam::111122223333:role/ExampleKeyUser';
const Operations: ["GenerateDataKey"];
kmsClient.createGrant({ KeyId, GranteePrincipal, Operations }, (err, data) => {
  ...
});
```

PowerShell

To create a grant, use the New-KMSGrant cmdlet.

```
# Create a grant

# Replace the following example key ARN with a valid key ID or key ARN
```

```
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
$granteePrincipal = 'arn:aws:iam::111122223333:role/ExampleKeyUser'
$operation = 'GenerateDataKey'

$response = New-KMSGrant -GranteePrincipal $granteePrincipal -KeyId $keyId -
Operation $operation
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Viewing a grant

To get detailed information about the grants on a KMS key, use the ListGrants operation.

> **Note**
>
> The `GranteePrincipal` field in the `ListGrants` response usually contains the grantee principal of the grant. However, when the grantee principal in the grant is an AWS service, the `GranteePrincipal` field contains the service principal, which might represent several different grantee principals.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

These examples use the optional `Limits` parameter, which determines how many grants the operation returns.

Java

For details about the Java implementation, see the listGrants method in the *AWS SDK for Java API Reference*.

```
// Listing grants on a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
Integer limit = 10;
```

```
ListGrantsRequest req = new ListGrantsRequest().withKeyId(keyId).withLimit(limit);
ListGrantsResult result = kmsClient.listGrants(req);
```

C#

For details, see the ListGrants method in the *AWS SDK for .NET*.

```
// Listing grants on a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
int limit = 10;

ListGrantsRequest listGrantsRequest = new ListGrantsRequest()
{
    KeyId = keyId,
    Limit = limit
};
ListGrantsResponse listGrantsResponse = kmsClient.ListGrants(listGrantsRequest);
```

Python

For details, see the `list_grants` method in the AWS SDK for Python (Boto3).

```
# Listing grants on a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kms_client.list_grants(
    KeyId=key_id,
    Limit=10
)
```

Ruby

For details, see the `list_grants` instance method in the AWS SDK for Ruby.

```
# Listing grants on a KMS key
# Replace the following example key ARN with a valid key ID or key ARN
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kmsClient.list_grants({
  key_id: key_id,
  limit: 10
})
```

PHP

For details, see the ListGrants method in the *AWS SDK for PHP*.

```
// Listing grants on a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$limit = 10;

$result = $KmsClient->listGrants([
    'KeyId' => $keyId,
    'Limit' => $limit,
]);
```

Node.js

For details, see the listGrants property in the *AWS SDK for JavaScript in Node.js*.

```
// Listing grants on a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
const KeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
const Limit = 10;
kmsClient.listGrants({ KeyId, Limit }, (err, data) => {
  ...
});
```

PowerShell

To view the details of all AWS KMS grants for a KMS key, use the Get-KMSGrantList cmdlet.

To limit the number of output objects, this example uses the Select-Object cmdlet, instead of the Limit parameter, which is being deprecated in list cmdlets. For help with paginating output in AWS Tools for PowerShell, see Output Pagination with AWS Tools for PowerShell.

```
# Listing grants on a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
$limit = 10

$response = Get-KMSGrantList -KeyId $keyId | Select-Object -First $limit
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

You must specify the KMS key in every `ListGrants` operations. However, you can further filter the grant list by specifying the grant ID or a grantee principal. The following examples get only the grants for a KMS key where the `test-engineer` role is the grantee principal.

Java

For details about the Java implementation, see the listGrants method in the *AWS SDK for Java API Reference*.

```
// Listing grants on a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
String grantee = "arn:aws:iam::111122223333:role/test-engineer";

ListGrantsRequest req = new
 ListGrantsRequest().withKeyId(keyId).withGranteePrincipal(grantee);
ListGrantsResult result = kmsClient.listGrants(req);
```

C#

For details, see the ListGrants method in the *AWS SDK for .NET*.

```
// Listing grants on a KMS key
```

```
//
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
String grantee = "arn:aws:iam::111122223333:role/test-engineer";

ListGrantsRequest listGrantsRequest = new ListGrantsRequest()
{
    KeyId = keyId,
    GranteePrincipal = grantee
};
ListGrantsResponse listGrantsResponse = kmsClient.ListGrants(listGrantsRequest);
```

Python

For details, see the `list_grants` method in the AWS SDK for Python (Boto3).

```
# Listing grants on a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
grantee = 'arn:aws:iam::111122223333:role/test-engineer'

response = kms_client.list_grants(
    KeyId=key_id,
    GranteePrincipal=grantee
)
```

Ruby

For details, see the `list_grants` instance method in the AWS SDK for Ruby.

```
# Listing grants on a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
grantee = 'arn:aws:iam::111122223333:role/test-engineer'

response = kmsClient.list_grants({
  key_id: keyId,
  grantee_principal: grantee
})
```

PHP

For details, see the ListGrants method in the *AWS SDK for PHP*.

```
// Listing grants on a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$grantee = 'arn:aws:iam::111122223333:role/test-engineer';

$result = $KmsClient->listGrants([
    'KeyId' => $keyId,
    'GranteePrincipal' => $grantee,
]);
```

Node.js

For details, see the listGrants property in the *AWS SDK for JavaScript in Node.js*.

```
// Listing grants on a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
const KeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
const Grantee = 'arn:aws:iam::111122223333:role/test-engineer';

kmsClient.listGrants({ KeyId, Grantee }, (err, data) => {
  ...
});
```

PowerShell

To view the details of all AWS KMS grants for a KMS key, use the Get-KMSGrantList cmdlet.

```
# Listing grants on a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
```

```
$grantee = 'arn:aws:iam::111122223333:role/test-engineer'
$response = Get-KMSGrantList -KeyId $keyId -GranteePrincipal $grantee
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Retiring a grant

To retire a grant for a KMS key, use the RetireGrant operation. You should retire a grant to clean up after you are done using it.

To retire a grant, provide the grant token, or both the grant ID and KMS key ID. For this operation, the KMS key ID must be Amazon Resource Name (ARN) of the KMS key. The grant token is returned by the CreateGrant operation. The grant ID is returned by the CreateGrant and ListGrants operations.

RetireGrant doesn't return a response. To verify that it was effective, use the ListGrants operation.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

Java

For details, see the retireGrant method in the *AWS SDK for Java API Reference*.

```
// Retire a grant
//
String grantToken = Place your grant token here;

RetireGrantRequest req = new RetireGrantRequest().withGrantToken(grantToken);
kmsClient.retireGrant(req);
```

C#

For details, see the RetireGrant method in the *AWS SDK for .NET*.

```
// Retire a grant
//
String grantToken = "Place your grant token here";

```

```
RetireGrantRequest retireGrantRequest = new RetireGrantRequest()
{
    GrantToken = grantToken
};
kmsClient.RetireGrant(retireGrantRequest);
```

Python

For details, see the `retire_grant` method in the AWS SDK for Python (Boto3).

```
# Retire a grant

grant_token = Place your grant token here

response = kms_client.retire_grant(
    GrantToken=grant_token
)
```

Ruby

For details, see the `retire_grant` instance method in the AWS SDK for Ruby.

```
# Retire a grant

grant_token = Place your grant token here

response = kmsClient.retire_grant({
  grant_token: grant_token
})
```

PHP

For details, see the RetireGrant method in the *AWS SDK for PHP*.

```
// Retire a grant
//
$grantToken = 'Place your grant token here';

$result = $KmsClient->retireGrant([
    'GrantToken' => $grantToken,
]);
```

Node.js

For details, see the retireGrant property in the *AWS SDK for JavaScript in Node.js*.

```
// Retire a grant
//
const GrantToken = 'Place your grant token here';
kmsClient.retireGrant({ GrantToken }, (err, data) => {
  ...
});
```

PowerShell

To retire a grant, use the Disable-KMSGrant cmdlet. To get the grant token, use the New-KMSGrant cmdlet. The `GrantToken` parameter takes a string, so you don't need to convert output that the Read-Host cmdlet returns.

```
# Retire a grant

$grantToken = Read-Host -Message Place your grant token here
Disable-KMSGrant -GrantToken $grantToken
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Revoking a grant

To revoke a grant to a KMS key, use the RevokeGrant operation. You can revoke a grant to explicitly deny operations that depend on it.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client.

Java

For details, see the revokeGrant method in the *AWS SDK for Java API Reference*.

```
// Revoke a grant on a KMS key
//
```

```
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

// Replace the following example grant ID with a valid one
String grantId = "grant1";

RevokeGrantRequest req = new
 RevokeGrantRequest().withKeyId(keyId).withGrantId(grantId);
kmsClient.revokeGrant(req);
```

C#

For details, see the RevokeGrant method in the *AWS SDK for .NET*.

```
// Revoke a grant on a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

// Replace the following example grant ID with a valid one
String grantId = "grant1";

RevokeGrantRequest revokeGrantRequest = new RevokeGrantRequest()
{
    KeyId = keyId,
    GrantId = grantId
};
kmsClient.RevokeGrant(revokeGrantRequest);
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Python

For details, see the `revoke_grant` method in the AWS SDK for Python (Boto3).

```
# Revoke a grant on a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
```

```
# Replace the following example grant ID with a valid one
grant_id = 'grant1'

response = kms_client.revoke_grant(
    KeyId=key_id,
    GrantId=grant_id
)
```

Ruby

For details, see the `revoke_grant` instance method in the AWS SDK for Ruby.

```
# Revoke a grant on a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

# Replace the following example grant ID with a valid one
grant_id = 'grant1'

response = kmsClient.revoke_grant({
  key_id: key_id,
  grant_id: grant_id
})
```

PHP

For details, see the RevokeGrant method in the *AWS SDK for PHP*.

```
// Revoke a grant on a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';

// Replace the following example grant ID with a valid one
$grantId = "grant1";

$result = $KmsClient->revokeGrant([
    'KeyId' => $keyId,
    'GrantId' => $grantId,
]);
```

Node.js

For details, see the revokeGrant property in the *AWS SDK for JavaScript in Node.js*.

```
// Revoke a grant on a KMS key
//
// Replace the following example key ARN with a valid key ID or key ARN
const KeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';

// Replace the following example grant ID with a valid one
const GrantId = 'grant1';
kmsClient.revokeGrant({ GrantId, KeyId }, (err, data) => {
  ...
});
```

PowerShell

To revoke a grant, use the Revoke-KMSGrant cmdlet.

```
# Revoke a grant on a KMS key

# Replace the following example key ARN with a valid key ID or key ARN
$keyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

# Replace the following example grant ID with a valid one
$grantId = 'grant1'

Revoke-KMSGrant -KeyId $keyId -GrantId $grantId
```

To use the AWS KMS PowerShell cmdlets, install the AWS.Tools.KeyManagementService module. For more information, see the AWS Tools for Windows PowerShell User Guide.

Testing your AWS KMS API calls

To use AWS KMS, you must have credentials that AWS can use to authenticate your API requests. The credentials must include the permission to access KMS keys and aliases. The permissions are determined by key policies, IAM policies, grants, and cross-account access controls. In addition to controlling access to KMS keys, you can control access to your CloudHSM, and to your custom key stores.

You can specify the `DryRun` API parameter to verify that you have the necessary permissions to use AWS KMS keys. You can also use `DryRun` to verify that the request parameters in a AWS KMS API call are correctly specified.

Topics

What is the DryRun parameter?

`DryRun` is an optional API parameter that you specify to verify that AWS KMS API calls will succeed. Use `DryRun` to test your API call, before actually making the call to AWS KMS. You can verify the following.

- That you have the necessary permissions to use AWS KMS keys.
- That you have specified the parameters in the call correctly.

AWS KMS supports using the `DryRun` parameter in certain API actions:

- CreateGrant
- Decrypt
- DeriveSharedSecret
- Encrypt
- GenerateDataKey
- GenerateDataKeyPair
- GenerateDataKeyPairWithoutPlaintext
- GenerateDataKeyWithoutPlaintext
- GenerateMac
- ReEncrypt
- RetireGrant
- RevokeGrant

- Sign
- Verify
- VerifyMac

Using the DryRun parameter will incur charges and will be billed as a standard API request. For more information about AWS KMS pricing, see AWS Key Management Service Pricing.

All API requests using the DryRun parameter apply to the request quota of the API and can result in a throttling exception if you exceed an API request quota. For example, calling Decrypt with DryRun or without DryRun counts against the same cryptographic operations quota. See Throttling AWS KMS requests to learn more.

Every call to an AWS KMS API operation is captured as an event and recorded in an AWS CloudTrail log. The output of any operations that specify the DryRun parameter appear in your CloudTrail log. For more information, see Logging AWS KMS API calls with AWS CloudTrail.

Specifying DryRun with the API

To use DryRun, specify the —dry-run parameter in AWS CLI commands and AWS KMS API calls that support the parameter. When you do, AWS KMS will verify whether your call will succeed. AWS KMS calls that use DryRun will always fail and return a message with information about reason why the call failed. The message can include the following exceptions:

- DryRunOperationException - The request would succeed if DryRun wasn't specified.
- ValidationException - The request failed from specifying an incorrect API parameter.
- AccessDeniedException - You do not have permissions to perform the specified API action on the KMS resource.

For example, the following command uses the CreateGrant operation and creates a grant that allows users who are authorized to assume the keyUserRole role to call the Decrypt operation on a specified symmetric KMS key. The DryRun parameter is specified.

```
$  aws kms create-grant \
    --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
    --grantee-principal arn:aws:iam::111122223333:role/keyUserRole \
    --operations Decrypt \
    --dry-run
```

AWS KMS eventual consistency

The AWS KMS API follows an eventual consistency model due to the distributed nature of the system. As a result, changes to AWS KMS resources might not be immediately visible to the subsequent commands you run.

When you perform AWS KMS API calls, there might be a brief delay before the change is available throughout AWS KMS. It typically takes less than a few seconds for the change to propagate throughout the system, but in some cases it can take several minutes. You might get unexpected errors, such as a `NotFoundException` or an `InvalidStateException`, during this time. For example, AWS KMS might return a `NotFoundException` if you call `GetParametersForImport` immediately after calling `CreateKey`.

We recommend that you configure a retry strategy on your AWS KMS clients to automatically retry operations after a brief waiting period. For more information, see Retry behavior in the AWS SDKs and Tools Reference Guide.

For grant related API calls, you can use a grant token to avoid any potential delay and use the permissions in a grant immediately. For more information, see Eventual consistency (for grants).

References

The following references provide useful information about using and managing KMS keys.

- Key type reference. Lists the type of KMS key that supports each AWS KMS API operation.

  To find: Can I enable and disable an RSA signing KMS key?
- Key state table. Shows how the key state of a KMS key affects its use in AWS KMS API operations.

  To find: Can I change the alias of a KMS key that is pending deletion?
- AWS KMS API permissions reference. Provides information about the permissions required for each AWS KMS API operation.

  To find: Can I run GetKeyPolicy on a key in a different AWS account? Can I allow `kms:Decrypt` permission in an IAM policy?
  - ViaService reference. Lists the AWS services that support the `kms:ViaService` condition key.

    To find: Can I use the `kms:ViaService` condition key to allow a permission only when it comes from Amazon ElastiCache? What about Amazon Neptune?
- AWS KMS pricing. Lists and explains the price of KMS keys.

  To find: How much does it cost to use my asymmetric keys?
- AWS KMS request quotas. Lists the per-second quotas for AWS KMS API requests in each account and Region.

  To find: How many Decrypt requests can I run in each second? How many Decrypt requests can I run on KMS keys in my custom key store?
- AWS KMS resource quotas. Lists the quotas on AWS KMS resources.

  To find: How many KMS key can I have in each Region of my account? How many aliases can I have on each KMS key?
- AWS services integrated with AWS KMS. Lists the AWS services that use KMS keys to protect the resources that they create, store, and manage.

  To find: Does Amazon Connect use KMS keys to protect my Connect resources?

Document history

This topic describes significant updates to the *AWS Key Management Service Developer Guide*.

Topics

- Recent updates
- Earlier updates

Recent updates

The following table describes significant changes to this documentation since January 2018. In addition to major changes listed here, we also update the documentation frequently to improve the descriptions and examples, and to address the feedback that you send to us. To be notified about significant changes, subscribe to the RSS feed.

You might need to scroll horizontally or vertically to see all of the data in this table.

Change	Description	Date
New feature	Added new `KeyUsage` type `KEY_AGREEMENT` for NIST-recommended elliptic curve (ECC) and SM2 (China Regions only) KMS keys and added support to derive shared secrets.	June 13, 2024
Updates to key rotation	Added support for custom rotation periods for automatic key rotations, on-demand key rotations, and visibility into your key material rotations.	April 12, 2024
Updates to managed policy	Added new permissions to `AWSKeyManagementServiceCustomKeyStor`	November 10, 2023

	`esServiceRolePolicy` that allow AWS KMS to monitor changes in the VPC that contains your AWS CloudHSM cluster so that AWS KMS can provide clear error messages in the case of failures.	
Feature update	Added support for the `DryRun` API parameter.	July 5, 2023
Feature update	Added support for importing key material for all types of AWS KMS keys, except custom key stores.	June 5, 2023
Feature update	Updates to AWS KMS APIs for Nitro Enclaves	March 10, 2023
Feature update	The `RSAES_PKCS1_V1_5` wrapping algorithm is deprecated. AWS KMS will end all support for `RSAES_PKCS1_V1_5` by October 1, 2023 pursuant to cryptographic key management guidance from the National Institute of Standards and Technology (NIST). We recommend that you begin using a different wrapping algorithm immediately.	February 28, 2023

Feature update	Added support for External key stores, a feature that lets you protect your AWS resources using cryptographic keys outside of AWS.	November 29, 2022
Quota change	Increased the AWS KMS keys resource quota to 100,000 KMS keys in each account and Region.	July 8, 2022
Feature update	Added support for HMAC KMS keys in more AWS Regions	July 8, 2022
New topic	Added the Resilience in AWS Key Management Service topic to the Security chapter of the AWS KMS Developer Guide.	June 14, 2022
New feature	Added support for AWS KMS keys and API operations that generate and verify HMAC codes.	April 19, 2022
Documentation change	Replace the term *customer master key (CMK)* with *AWS KMS key* and *KMS key*.	August 30, 2021
New feature	Added support for multi-Region keys, a set of interoperable KMS keys in different Regions that have the same key ID and key material. You can use multi-Region keys to encrypt data in one Region and decrypt data in a different Region.	June 8, 2021

New feature	Added support for attribute based access control (ABAC). You can use tags and aliases to control access to your AWS KMS keys.	December 17, 2020
New feature	Added support for VPC endpoint policies.	July 9, 2020
New content	Explains the security properties of AWS KMS.	June 18, 2020
New feature	Added support for asymmetric AWS KMS keys and asymmetric data keys.	November 25, 2019
Updated feature	You can view the key policy of AWS managed keys in the AWS KMS console. This feature used to be limited to customer managed keys.	November 15, 2019
New feature	Explains how to use hybrid post-quantum key exchange algorithms in TLS for your calls to AWS KMS.	November 4, 2019
Quota change	Increased the resource quotas for some APIs that manage KMS keys.	September 18, 2019
Quota change	Changed the resource quotas for KMS keys, aliases, and grants per KMS key.	March 27, 2019

Quota change	Changed the shared per-second request quota for cryptographic operations that use AWS KMS keys in a custom key store.	March 7, 2019
New feature	Explains how to create and manage AWS KMS custom key stores. Each key store is backed by an AWS CloudHSM cluster that you own and control.	November 26, 2018
New console	Explains how to use the new AWS KMS console, which is independent of the IAM console. The original console, and instructions for using it, will remain available for a brief period to give you time to familiarize yourself with the new console.	November 7, 2018
Quota change	Changed the shared request quota for use of AWS KMS keys.	August 21, 2018
New content	Explains how AWS Secrets Manager uses AWS KMS keys to encrypt the secret value in a secret.	July 13, 2018
New content	Explains how DynamoDB uses AWS KMS AWS KMS keys to support its server-side encryption option.	May 23, 2018

New feature	Explains how to use a private endpoint in your VPC to connect directly to AWS KMS, instead of connecting over the internet.	January 22, 2018

Earlier updates

The following table describes the important changes to the AWS Key Management Service Developer Guide prior to 2018.

You might need to scroll horizontally or vertically to see all of the data in this table.

Change	Description	Date
New content	Added documentation about Tagging keys.	February 15, 2017
New content	Added documentation about Monitoring AWS KMS keys and Monitoring with Amazon CloudWatch.	August 31, 2016
New content	Added documentation about Imported key material.	August 11, 2016
New content	Added the following documentation: IAM policies, Permissions reference, and Condition keys.	July 5, 2016
Update	Updated portions of the documentation in the Authentication and access control chapter.	July 5, 2016

Change	Description	Date
Update	Updated the Quotas page to reflect new default quotas.	May 31, 2016
Update	Updated the Quotas page to reflect new default quotas, and updated the grant token documentation to improve clarity and accuracy.	April 11, 2016
New content	Added documentation about Allowing multiple IAM principals to access a KMS key and Using the IP address condition.	February 17, 2016
Update	Updated the Key policies in AWS KMS and Changing a key policy pages to improve clarity and accuracy.	February 17, 2016
Update	Updated the Managing keys topic pages to improve clarity.	January 5, 2016
New content	Added documentation about How AWS CloudTrail uses AWS KMS.	November 18, 2015
New content	Added instructions for Changing a key policy.	November 18, 2015
Update	Updated the documentation about How Amazon Relational Database Service (Amazon RDS) uses AWS KMS.	November 18, 2015

Change	Description	Date
New content	Added documentation about How WorkSpaces uses AWS KMS.	November 6, 2015
Update	Updated the Key policies in AWS KMS page to improve clarity.	October 22, 2015
New content	Added documentation about Deleting AWS KMS keys, including supportin g documentation about Creating an alarm and Determining past usage of a KMS key.	October 15, 2015
New content	Added documentation about Determining access to AWS KMS keys.	October 15, 2015
New content	Added documentation about Key states of AWS KMS keys.	October 15, 2015
New content	Added documentation about How Amazon Simple Email Service (Amazon SES) uses AWS KMS.	October 1, 2015
Update	Updated the Quotas page to explain the new request quotas.	August 31, 2015
New content	Added information about the charges for using AWS KMS. See AWS KMS Pricing.	August 14, 2015

Change	Description	Date
New content	Added request quotas to the AWS KMS Quotas.	June 11, 2015
New content	Added a new Java code sample demonstrating use of the `UpdateAlias` operation . See Updating an alias.	June 1, 2015
Update	Moved the AWS Key Management Service regions table to the *AWS General Reference*.	May 29, 2015
New content	Added documentation about How Amazon EMR uses AWS KMS.	January 28, 2015
New content	Added documentation about How Amazon WorkMail uses AWS KMS.	January 28, 2015
New content	Added documentation about How Amazon Relational Database Service (Amazon RDS) uses AWS KMS.	January 6, 2015
New content	Added documentation about How Amazon Elastic Transcoder uses AWS KMS.	November 24, 2014
New guide	Introduced the *AWS Key Management Service Developer Guide*.	November 12, 2014