AWS KMS Cryptographic Details

AWS Key Management Service

AWS Key Management Service: AWS KMS Cryptographic Details

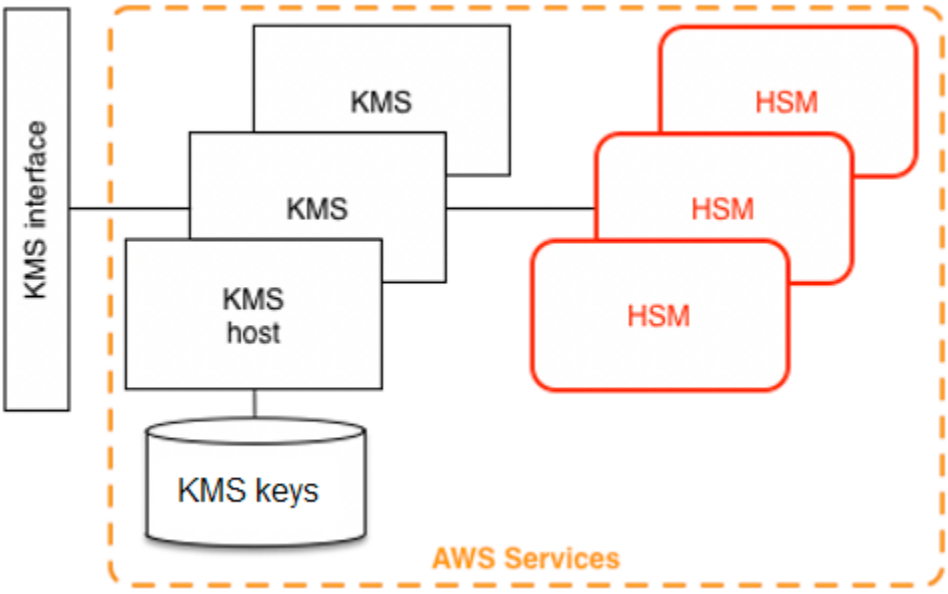
Table of Contents

Introduction to the cryptographic details of AWS KMS

AWS Key Management Service (AWS KMS) provides a web interface to generate and manage cryptographic keys and operates as a cryptographic service provider for protecting data. AWS KMS offers traditional key management services integrated with AWS services to provide a consistent view of customers' keys across AWS, with centralized management and auditing. This whitepaper provides a detailed description of the cryptographic operations of AWS KMS to assist you in evaluating the features offered by the service.

AWS KMS includes a web interface through the AWS Management Console, command line interface, and RESTful API operations to request cryptographic operations of a distributed fleet of FIPS 140-2 validated hardware security modules (HSMs)[1]. The AWS KMS HSM is a multichip standalone hardware cryptographic appliance designed to provide dedicated cryptographic functions to meet the security and scalability requirements of AWS KMS. You can establish your own HSM-based cryptographic hierarchy under keys that you manage as AWS KMS keys. These keys are made available only on the HSMs and only in memory for the necessary time needed to process your cryptographic request. You can create multiple KMS keys, each represented by its key ID. Only under AWS IAM roles and accounts administered by each customer can customer KMS keys be created, deleted, or used to encrypt, decrypt, sign, or verify data. You can define access controls on who can manage and/or use KMS keys by creating a policy that is attached to the key. Such policies allow you to define application-specific uses for your keys for each API operation.

In addition, most AWS services support encryption of data at rest using KMS keys. This capability allows customers to control how and when AWS services can access encrypted data by controlling how and when KMS keys can be accessed.



AWS KMS is a tiered service consisting of web-facing AWS KMS hosts and a tier of HSMs. The grouping of these tiered hosts forms the AWS KMS stack. All requests to AWS KMS must be made over the Transport Layer Security protocol (TLS) and terminate on an AWS KMS host. AWS KMS hosts only allow TLS with a ciphersuite that provides perfect forward secrecy. AWS KMS authenticates and authorizes your requests using the same credential and policy mechanisms of AWS Identity and Access Management (IAM) that are available for all other AWS API operations.

Basic concepts

Learning some basic terms and concepts will help you get the most out of AWS Key Management Service.

AWS KMS key

> **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.

A logical key that represents the top of your key hierarchy. A KMS key is given an Amazon Resource Name (ARN) that includes a unique key identifier, or key ID. AWS KMS keys have three types:

- **Customer managed key** – Customers create and control the lifecycle and key policies of customer managed keys. All requests made against these keys are logged as CloudTrail events.
- **AWS managed keys** – AWS creates and controls the lifecycle and key policies of AWS managed keys, which are resources in a customer's AWS account. Customers can view access policies and CloudTrail events for AWS managed keys, but cannot manage any aspect of these keys. All requests made against these keys are logged as CloudTrail events.
- **AWS owned keys** – These keys are created and exclusively used by AWS for internal encryption operations across different AWS services. Customers do not have visibility into key policies or AWS owned key usage in CloudTrail.

Alias

A user-friendly name that is associated with a KMS key. The alias can be used interchangeably with key ID in many of the AWS KMS API operations.

Permissions

A policy attached to a KMS key that defines permissions on the key. The default policy allows any principals that you define, as well as allowing the AWS account to add IAM policies that reference the key.

Grants

The delegated permission to use a KMS key when the intended IAM principals or duration of usage is not known at the outset and therefore cannot be added to a key or IAM policy. One use of grants is to define scoped-down permissions for how an AWS service can use a KMS key. The service may need to use your key to do asynchronous work on your behalf on encrypted data in the absence of a direct-signed API call from you.

Data keys

Cryptographic keys generated on HSMs, protected by a KMS key. AWS KMS allows authorized entities to obtain data keys protected by a KMS key. They can be returned both as plaintext (unencrypted) data keys and as encrypted data keys. Data keys can be symmetric or asymmetric (with both the public and private portions returned).

Ciphertexts

The encrypted output of AWS KMS, sometimes referred to as customer ciphertext to eliminate confusion. Ciphertext contains encrypted data with additional information that identifies the KMS key to use in the decryption process. Encrypted data keys are one common example of

ciphertext produced when using a KMS key, but any data under 4 KB in size can be encrypted under a KMS key to produce a ciphertext.

Encryption context

A key–value pair map of additional information that is associated with AWS KMS–protected information. AWS KMS uses authenticated encryption to protect data keys. The encryption context is incorporated into the AAD of the authenticated encryption in AWS KMS–encrypted ciphertexts. This context information is optional and not returned when requesting a key (or an encryption operation). But if used, this context value is required to successfully complete a decryption operation. An intended use of the encryption context is to provide additional authenticated information. This information can help you enforce policies and be included in the AWS CloudTrail logs. For example, you could use a key–value pair of {"key name":"satellite uplink key"} to name the data key. Subsequent use of the key creates an AWS CloudTrail entry that includes “key name”: “satellite uplink key.” This additional information can provide useful context to understand why a given KMS key was used.

Public key

When using asymmetric ciphers (RSA or elliptic curve), the public key is the “public component” of a public-private key pair. The public key can be shared and distributed to entities that need to encrypt data for the owner of the public-private key pair. For digital signature operations, the public key is used to verify the signature.

Private key

When using asymmetric ciphers (RSA or elliptic curve), the private key is the “private component” of a public-private key pair. The private key is used to decrypt data or create digital signatures. Similar to symmetric KMS keys, private keys are encrypted in HSMs. They are decrypted only into the short term memory of the HSM and only for the time needed to process your cryptographic request.

AWS KMS design goals

AWS KMS is designed to meet the following requirements.

Durability

The durability of cryptographic keys is designed to equal that of the highest durability services in AWS. A single cryptographic key can encrypt large volumes of your data that has accumulated over a long time.

Trustworthy

Use of keys is protected by access control policies that you define and manage. There is no mechanism to export plaintext KMS keys. The confidentiality of your cryptographic keys is crucial. Multiple Amazon employees with role-specific access to quorum-based access controls are required to perform administrative actions on the HSMs.

Low-latency and high throughput

AWS KMS provides cryptographic operations at latency and throughput levels suitable for use by other services in AWS.

Independent Regions

AWS provides independent Regions for customers who need to restrict data access in different Regions. Key usage can be isolated within an AWS Region.

Secure source of random numbers

Because strong cryptography depends on truly unpredictable random number generation, AWS KMS provides a high-quality and validated source of random numbers.

Audit

AWS KMS records the use and management of cryptographic keys in AWS CloudTrail logs. You can use AWS CloudTrail logs to inspect use of your cryptographic keys, including the use of keys by AWS services on your behalf.

To achieve these goals, the AWS KMS system includes a set of AWS KMS operators and service host operators (collectively, “operators”) that administer “domains.” A domain is a Regionally defined set of AWS KMS servers, HSMs, and operators. Each AWS KMS operator has a hardware token that contains a private and public key pair that is used to authenticate its actions. The HSMs have an additional private and public key pair to establish encryption keys that protect HSM state synchronization.

This paper illustrates how AWS KMS protects your keys and other data that you want to encrypt. Throughout this document, encryption keys or data that you want to encrypt are referred to as “secrets” or “secret material.”

AWS Key Management Service foundations

The topics in this chapter describe the cryptographic primitives of AWS Key Management Service and where they are used. They also introduce the basic elements of AWS KMS.

Topics

Cryptographic primitives

AWS KMS uses configurable cryptographic algorithms so that the system can quickly migrate from one approved algorithm, or mode, to another. The initial default set of cryptographic algorithms has been selected from Federal Information Processing Standard (FIPS-approved) algorithms for their security properties and performance.

Entropy and random number generation

AWS KMS key generation is performed on the AWS KMS HSMs. The HSMs implement a hybrid random number generator that uses the NIST SP800-90A Deterministic Random Bit Generator (DRBG) CTR_DRBG using AES-256. It is seeded with a nondeterministic random bit generator with 384-bits of entropy and updated with additional entropy to provide prediction resistance on every call for cryptographic material.

Symmetric key operations (encryption only)

All symmetric key encrypt commands used within HSMs use the Advanced Encryption Standards (AES), in Galois Counter Mode (GCM) using 256-bit keys. The analogous calls to decrypt use the inverse function.

AES-GCM is an authenticated encryption scheme. In addition to encrypting plaintext to produce ciphertext, it computes an authentication tag over the ciphertext and any additional data for which authentication is required (additionally authenticated data, or AAD). The authentication tag helps ensure that the data is from the purported source and that the ciphertext and AAD have not been modified.

Frequently, AWS omits the inclusion of the AAD in our descriptions, especially when referring to the encryption of data keys. It is implied by surrounding text in these cases that the structure to

be encrypted is partitioned between the plaintext to be encrypted and the cleartext AAD to be protected.

AWS KMS provides an option for you to import key material into an AWS KMS key instead of relying on AWS KMS to generate the key material. This imported key material can be encrypted using RSAES-OAEP or RSAES-PKCS1-v1_5 to protect the key during transport to the AWS KMS HSM. The RSA key pairs are generated on AWS KMS HSMs. The imported key material is decrypted on an AWS KMS HSM and re-encrypted under AES-GCM before being stored by the service.

Asymmetric key operations (encryption, digital signing and signature verification)

AWS KMS supports the use of asymmetric key operations for both encryption and digital signature operations. Asymmetric key operations rely on a mathematically related public key and private key pair that you can use for encryption and decryption or signing and signature verification, but not both. 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.

AWS KMS supports two types of asymmetric ciphers.

- **RSA-OAEP (for encryption) & RSA-PSS and RSA-PKCS-#1-v1_5 (for signing and verification)** – Supports RSA key lengths (in bits): 2048, 3072, and 4096 for different security requirements.
- **Elliptic Curve (ECC)** – Used exclusively for signing and verification. Supports ECC curves: NIST P256, P384, P521, SECP 256k1.

Key derivation functions

A key derivation function is used to derive additional keys from an initial secret or key. AWS KMS uses a key derivation function (KDF) to derive per-call keys for every encryption under an AWS KMS key. All KDF operations use the KDF in counter mode using HMAC [FIPS197] with SHA256 [FIPS180]. The 256-bit derived key is used with AES-GCM to encrypt or decrypt customer data and keys.

AWS KMS internal use of digital signatures

Digital signatures are also used to authenticate commands and communications between AWS KMS entities. All service entities have an elliptic curve digital signature algorithm (ECDSA) key pair. They

perform ECDSA as defined in Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS) and X9.62-2005: *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*. The entities use the secure hash algorithm defined in Federal Information Processing Standards Publications, FIPS PUB 180-4, known as SHA384. The keys are generated on the curve secp384r1 (NIST-P384).

Envelope encryption

A basic construction used within many cryptographic systems is envelope encryption. Envelope encryption uses two or more cryptographic keys to secure a message. Typically, one key is derived from a longer-term static key *k*, and another key is a per-message key, *msgKey*, which is generated to encrypt the message. The envelope is formed by encrypting the message: *ciphertext = Encrypt(msgKey, message)* . Then the message key is encrypted with the long-term static key: *encKey = Encrypt(k, msgKey)* . Finally, the two values *(encKey, ciphertext)* are packaged into a single structure, or envelope encrypted message.

The recipient, with access to *k*, can open the enveloped message by first decrypting the encrypted key and then decrypting the message.

AWS KMS provides the ability to manage these longer-term static keys and automate the process of envelope encryption of your data.

In addition to the encryption capabilities provided within the AWS KMS service, the AWS Encryption SDK provides client-side envelope encryption libraries. You can use these libraries to protect your data and the encryption keys that are used to encrypt that data.

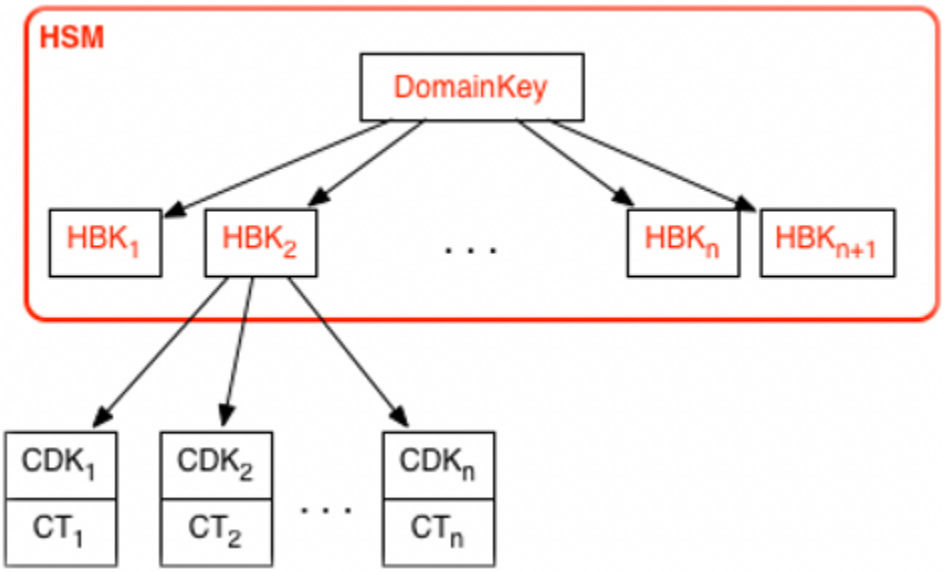
AWS KMS key hierarchy

Your key hierarchy starts with a top-level logical key, an AWS KMS key. A KMS key represents a container for top-level key material and is uniquely defined within the AWS service namespace with an Amazon Resource Name (ARN). The ARN includes a uniquely generated key identifier, a *key ID.* A KMS key is created based on a user-initiated request through AWS KMS. Upon reception, AWS KMS requests the creation of an initial HSM backing key (HBK) to be placed into the KMS key container. The HBK is generated on an HSM in the domain and is designed never to be exported from the HSM in plaintext. Instead, the HBK is exported encrypted under HSM-managed domain keys. These exported HBKs are referred to as exported key tokens (EKTs).

The EKT is exported to a highly durable, low-latency storage. For example, suppose you receive an ARN to the logical KMS key. This represents the top of a key hierarchy, or cryptographic context, for

you. You can create multiple KMS keys within your account and set policies on your KMS keys like any other AWS named resource.

Within the hierarchy of a specific KMS key, the HBK can be thought of as a version of the KMS key. When you want to rotate the KMS key through AWS KMS, a new HBK is created and associated with the KMS key as the active HBK for the KMS key. The older HBKs are preserved and can be used to decrypt and verify previously protected data. But only the active cryptographic key can be used to protect new information.



You can make requests through AWS KMS to use your KMS keys to directly protect information or request additional HSM-generated keys that are protected under your KMS key. These keys are called customer data keys, or CDKs. CDKs can be returned encrypted as ciphertext (CT), in plaintext, or both. All objects encrypted under a KMS key (either customer-supplied data or HSM-generated keys) can be decrypted only on an HSM via a call through AWS KMS.

The returned ciphertext, or the decrypted payload, is never stored within AWS KMS. The information is returned to you over your TLS connection to AWS KMS. This also applies to calls made by AWS services on your behalf.

The key hierarchy and the specific key properties appear in the following table.

Key	Description	Lifecycle
Domain key	A 256-bit AES-GCM key only in memory of an HSM used to wrap	Rotated daily[1]

Key	Description	Lifecycle
	versions of the KMS keys, the HSM backing keys.	
HSM backing key	A 256-bit symmetric key or RSA or elliptic curve private key, used to protect customer data and keys and stored encrypted under domain keys. One or more HSM backing keys comprise the KMS key, represented by the keyId.	Rotated yearly[2] (optional config.)
Derived encryption key	A 256-bit AES-GCM key only in memory of an HSM used to encrypt customer data and keys. Derived from an HBK for each encryption.	Used once per encrypt and regenerated on decrypt
Customer data key	User-defined symmetric or asymmetric key exported from HSM in plaintext and ciphertext. Encrypted under an HSM backing key and returned to authorized users over TLS channel.	Rotation and use controlled by application

[1] AWS KMS might from time to time relax domain key rotation to at most weekly to account for domain administration and configuration tasks.

[2] Default AWS managed keys created and managed by AWS KMS on your behalf are automatically rotated annually.

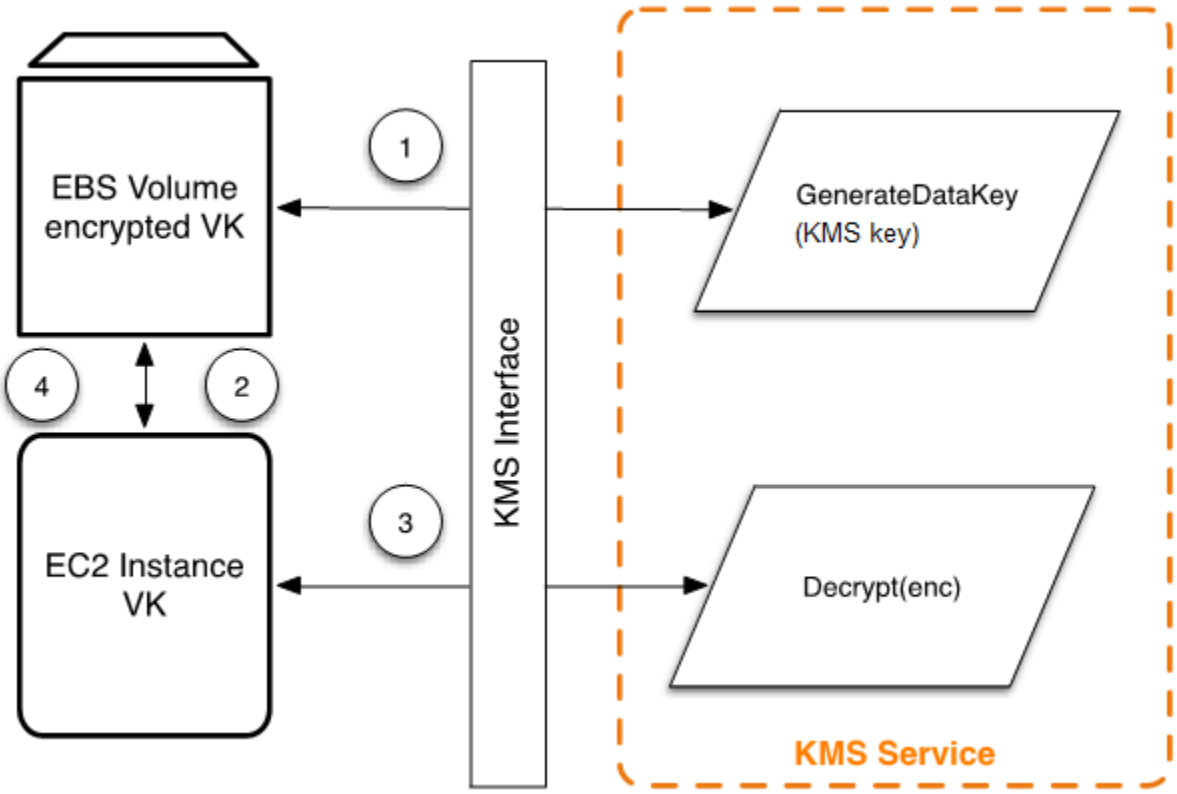
AWS KMS use cases

Use cases can help you get the most out of AWS Key Management Service. The first demonstrates how AWS KMS performs server-side encryption with AWS KMS keys on an Amazon Elastic Block Store (Amazon EBS) volume. The second is a client-side application that demonstrates how you can use envelope encryption to protect content with AWS KMS.

Topics

Amazon EBS volume encryption

Amazon EBS offers volume encryption capability. Each volume is encrypted using AES-256-XTS. This requires two 256-bit volume keys, which you can think of as one 512-bit volume key. The volume key is encrypted under a KMS key in your account. For Amazon EBS to encrypt a volume for you, it must have access to generate a volume key (VK) under a KMS key in the account. You do this by providing a grant for Amazon EBS to the KMS key to create data keys and to encrypt and decrypt these volume keys. Now Amazon EBS uses AWS KMS with a KMS key to generate AWS KMS encrypted volume keys.



The following workflow encrypts data that is being written to an Amazon EBS volume:

1. Amazon EBS obtains an encrypted volume key under a KMS key through AWS KMS over a TLS session and stores the encrypted key with the volume metadata.
2. When the Amazon EBS volume is mounted, the encrypted volume key is retrieved.
3. A call to AWS KMS over TLS is made to decrypt the encrypted volume key. AWS KMS identifies the KMS key and makes an internal request to an HSM in the fleet to decrypt the encrypted volume key. AWS KMS then returns the volume key back to the Amazon Elastic Compute Cloud (Amazon EC2) host that contains your instance over the TLS session.
4. The volume key is used to encrypt and decrypt all data going to and from the attached Amazon EBS volume. Amazon EBS retains the encrypted volume key for later use in case the volume key in memory is no longer available.

For more information about encrypting Amazon EBS volumes with KMS keys, see How Amazon Elastic Block Store uses AWS KMS in the *AWS Key Management Service Developer Guide* and **Amazon EBS encryption** in the Amazon EC2 User Guide and Amazon EC2 User Guide.

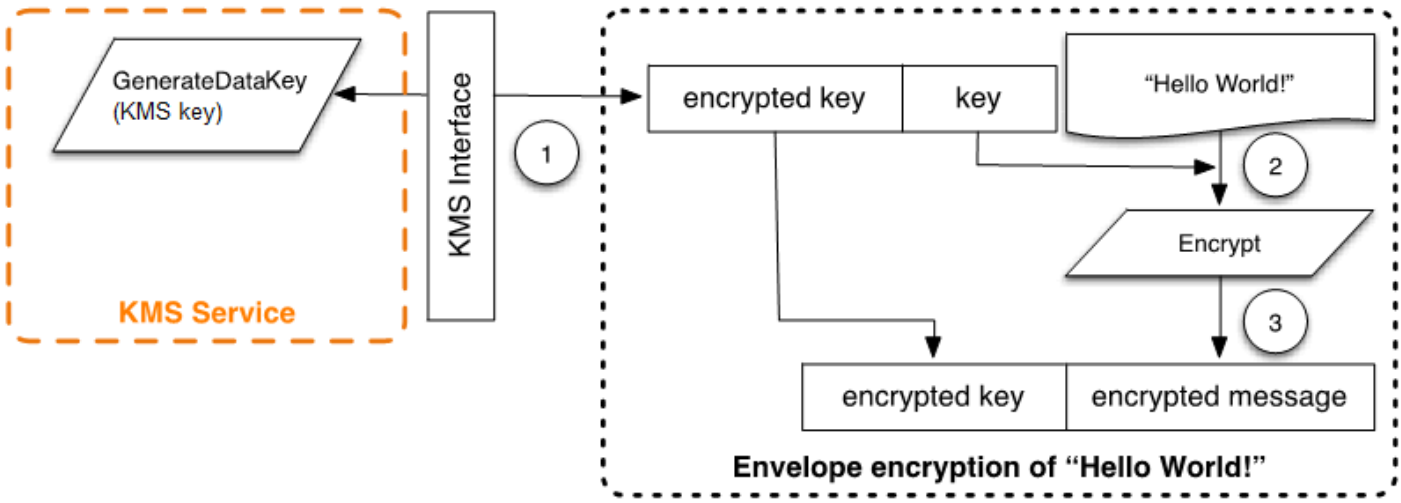
Client-side encryption

The AWS Encryption SDK includes an API operation for performing envelope encryption using a KMS key. For complete recommendations and usage details see the related documentation. Client applications can use the AWS Encryption SDK to perform envelope encryption using AWS KMS.

```
// Instantiate the SDK
final AwsCrypto crypto = new AwsCrypto();
// Set up the KmsMasterKeyProvider backed by the default credentials
final KmsMasterKeyProvider prov = new KmsMasterKeyProvider(keyId);
// Do the encryption
final byte[] ciphertext = crypto.encryptData(prov, message);
```

The client application can run the following steps:

1. A request is made under a KMS key for a new data key. An encrypted data key and a plaintext version of the data key are returned.
2. Within the AWS Encryption SDK, the plaintext data key is used to encrypt the message. The plaintext data key is then deleted from memory.
3. The encrypted data key and encrypted message are combined into a single ciphertext byte array.

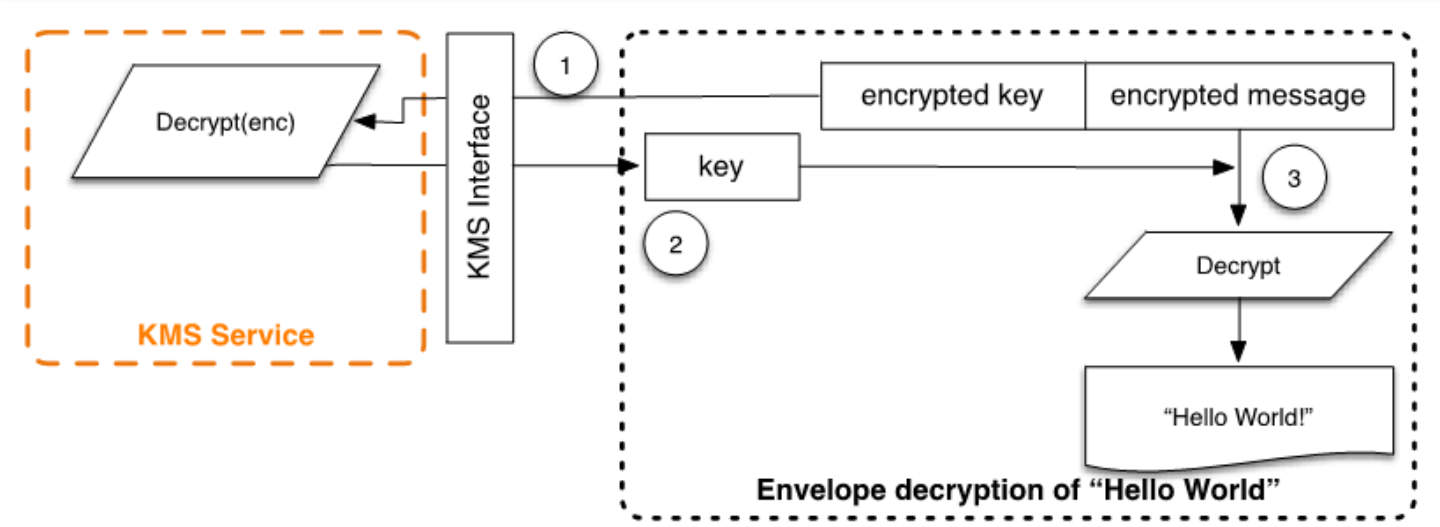


The envelope-encrypted message can be decrypted using the decrypt functionality to obtain the originally encrypted message.

```
final AwsCrypto crypto = new AwsCrypto();
```

```
final KmsMasterKeyProvider prov = new KmsMasterKeyProvider(keyId);
// Decrypt the data
final CryptoResult<byte[], KmsMasterKey> res = crypto.decryptData(prov, ciphertext);
// We need to check the KMS key to ensure that the
// assumed key was used
if (!res.getMasterKeyIds().get(0).equals(keyId)) {
     throw new IllegalStateException("Wrong key id!");
}
byte[] plaintext = res.getResult();
```

1. The AWS Encryption SDK parses the envelope-encrypted message to obtain the encrypted data key and make a request to AWS KMS to decrypt the data key.
2. The AWS Encryption SDK receives the plaintext data key from AWS KMS.
3. The data key is then used to decrypt the message, returning the initial plaintext.

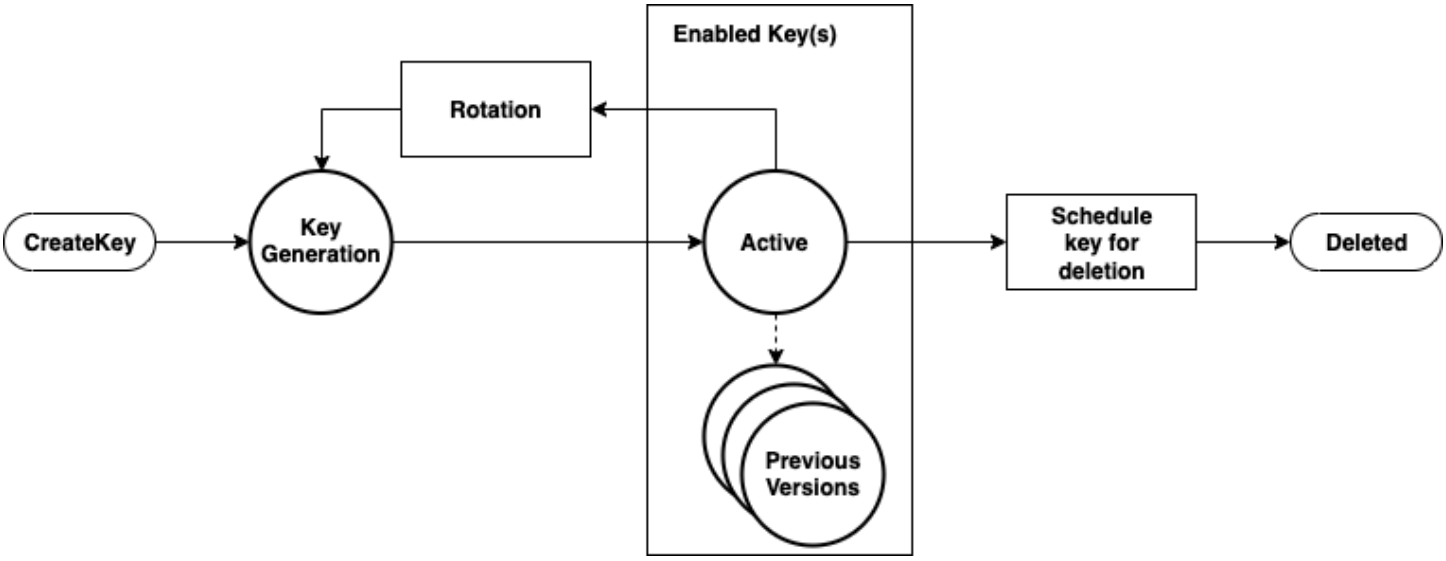


Working with AWS KMS keys

An AWS KMS key refers to a logical key that might refer to one or more hardware security module (HSM) backing keys (HBKs). This topic explains how to create a KMS key, import key material, and how to enable, disable, rotate, and delete KMS keys.

> **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.

This chapter discusses the lifecycle of a KMS key from creation to deletion, as shown in the following image.



Topics

Calling CreateKey

An AWS KMS key is generated as a result of a call to the `CreateKey` API call.

The following is a subset of the `CreateKey` request syntax.

```
{
  "Description": "string",
  "KeySpec": "string",
  "KeyUsage": "string",
  "Origin": "string";
  "Policy": "string"
}
```

The request accepts the following data in JSON format.

Description

(Optional) Description of the key. We recommend that you choose a description that helps you decide whether the key is appropriate for a task.

KeySpec

Specifies the type of KMS key to create. The default value, SYMMETRIC_DEFAULT, creates a symmetric encryption KMS key. This parameter is optional for symmetric encryption keys, and is required for all other key specs.

KeyUsage

Specifies the use of the key. Valid values are `ENCRYPT_DECRYPT`, `SIGN_VERIFY`, or `GENERATE_VERIFY_MAC`. The default value is `ENCRYPT_DECRYPT`. This parameter is optional for symmetric encryption keys, and is required for all other key specs.

Origin

(Optional) Specifies the source of the key material for the KMS key. The default value is `AWS_KMS`, which indicates that AWS KMS generates and manages the key material for the KMS key. Other valid values include `EXTERNAL`, which represents a KMS key created without key material for imported key material, and `AWS_CLOUDHSM` which creates a KMS key in a custom key store backed by an AWS CloudHSM cluster that you control.

Policy

(Optional) Policy to attach to the key. If the policy is omitted, the key is created with the default policy (following) that allows the root account and IAM principals with AWS KMS permissions to manage it.

For details on the policy, see Key policies in AWS KMS and Default key policy in the *AWS Key Management Service Developer Guide*.

The `CreateKey` request returns a response that includes a key ARN.

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

If the `Origin` is `AWS_KMS`, after the ARN is created, a request to an AWS KMS HSM is made over an authenticated session to provision a hardware security module (HSM) backing key (HBK). The HBK is a 256-bit key that is associated with this key ID of the KMS key. It can be generated only on an HSM and is designed never to be exported outside of the HSM boundary in cleartext. The HBK is encrypted under the current domain key, DK_0. These encrypted HBKs are referred to as encrypted key tokens (EKTs). Although the HSMs can be configured to use a variety of key wrapping methods, the current implementation uses AES-256 in Galois Counter Mode (GCM), an authenticated encryption scheme. This authenticated encryption mode allows us to protect some cleartext exported key token metadata.

This is stylistically represented as:

$$EKT = Encrypt(DK_0, HBK)$$

Two fundamental forms of protection are provided to your KMS keys and the subsequent HBKs: authorization policies set on your KMS keys and the cryptographic protections on your associated HBKs. The remaining sections describe the cryptographic protections and the security of the management functions in AWS KMS.

In addition to the ARN, you can create a user-friendly name and associate it with the KMS key by creating an *alias* for the key. Once an alias has been associated with a KMS key, the alias can be used to identify the KMS key in cryptographic operations. For detailed information, see Using aliases in the *AWS Key Management Service Developer Guide*.

Multiple levels of authorizations surround the use of KMS keys. AWS KMS enables separate authorization policies between the encrypted content and the KMS key. For instance, an AWS KMS

envelope-encrypted Amazon Simple Storage Service (Amazon S3) object inherits the policy on the Amazon S3 bucket. However, access to the necessary encryption key is determined by the access policy on the KMS key. For information about authorization of KMS keys, see Authentication and access control for AWS KMS in the *AWS Key Management Service Developer Guide*.

Importing key material

AWS KMS provides a mechanism for importing the cryptographic material used for an HBK. As described in Calling CreateKey, when the CreateKey command is used with `Origin` set to `EXTERNAL`, a logical KMS key is created that contains no underlying HBK. The cryptographic material must be imported using the `ImportKeyMaterial` API call. You can use this feature to control the key creation and durability of the cryptographic material. If you use this feature, we recommend that you take significant caution in the handling and durability of these keys in your environment. For complete details and recommendations for importing key material, see Importing key material in the *AWS Key Management Service Developer Guide*.

Calling ImportKeyMaterial

The `ImportKeyMaterial` request imports the necessary cryptographic material for the HBK. The cryptographic material must be a 256-bit symmetric key. It must be encrypted using the algorithm specified in `WrappingAlgorithm` under the returned public key from a recent `GetParametersForImport` request.

An `ImportKeyMaterial` request takes the following arguments.

```
{
  "EncryptedKeyMaterial": blob,
  "ExpirationModel": "string",
  "ImportToken": blob,
  "KeyId": "string",
  "ValidTo": number
}
```

EncryptedKeyMaterial

The imported key material encrypted with the public key returned in a `GetParametersForImport` request using the wrapping algorithm specified in that request.

ExpirationModel

Specifies whether the key material expires. When this value is `KEY_MATERIAL_EXPIRES`, the `ValidTo` parameter must contain an expiration date. When this value is `KEY_MATERIAL_DOES_NOT_EXPIRE`, do not include the `ValidTo` parameter. The valid values are `"KEY_MATERIAL_EXPIRES"` and `"KEY_MATERIAL_DOES_NOT_EXPIRE"`.

ImportToken

The import token returned by the same `GetParametersForImport` request that provided the public key.

KeyId

The KMS key that will be associated with the imported key material. The `Origin` of the KMS key must be `EXTERNAL`.

You can delete and reimport the *same* imported key material into the specified KMS key, but you cannot import or associate the KMS key any other key material.

ValidTo

(Optional) The time at which the imported key material expires. When the key material expires, AWS KMS deletes the key material and the KMS key becomes unusable. This parameter is required when the value of the `ExpirationModel` is `KEY_MATERIAL_EXPIRES`. Otherwise it is invalid.

When the request succeeds, the KMS key is available for use within AWS KMS until the specified expiration date, if one is provided. After the imported key material expires, the EKT is deleted from the AWS KMS storage layer.

Enabling and disabling keys

Disabling a KMS key prevents the key from being used in cryptographic operations. It suspends the ability to use all HBKs that are associated with the KMS key. Enabling restores use of the HBKs and the KMS key. Enable and Disable are simple requests that take only the key ID or key ARN of the KMS key.

Deleting keys

Authorized users can use the ScheduleKeyDeletion API to schedule the deletion of a KMS key and all associated HBKs. This is an inherently destructive operation, and you should exercise caution when deleting keys from AWS KMS. AWS KMS enforces a minimal wait time of seven days when deleting KMS keys. During the waiting period the key is placed in a disabled state with a key state of **Pending Deletion**. All calls to use the key for cryptographic operations will fail. ScheduleKeyDeletion takes the following arguments.

```
{
  "KeyId": "string",
  "PendingWindowInDays": number
}
```

KeyId

The unique identifier for the KMS key to delete. To specify this value, use the unique key ID or the key ARN of the KMS key.

PendingWindowInDays

(Optional) The waiting period, in number of days. This value is optional. The range is 7-30 days and the default value is 30 days. After the waiting period ends, AWS KMS deletes the KMS key and all associated HBKs.

Rotating key material

Authorized users can enable automatic annual rotation of their customer managed KMS keys. AWS managed keys are always rotated every year.

When a KMS key is rotated, a new HBK is created and marked as the current version of the key material for all new encrypt requests. All previous versions of the HBK remain available for use in perpetuity to decrypt any ciphertexts that were encrypted using this HBK version. Because AWS KMS does not store any ciphertext encrypted under a KMS key, ciphertexts encrypted under an older, rotated HBK require that HBK to decrypt. You can use the ReEncrypt API to reencrypt any ciphertext under the new HBK for the KMS key or under a different KMS key without exposing the plaintext.

For information about enabling and disabling key rotation, see Rotating AWS KMS keys in the *AWS Key Management Service Developer Guide*.

Customer data operations

After you have established a KMS key, it can be used to perform cryptographic operations. Whenever data is encrypted under a KMS key, the resulting object is a customer ciphertext. The ciphertext contains two sections: an unencrypted header (or cleartext) portion, protected by the authenticated encryption scheme as the additional authenticated data, and an encrypted portion. The cleartext portion includes the HBK identifier (HBKID). These two immutable fields of the ciphertext value help ensure that AWS KMS can decrypt the object in the future.

Topics

Generating data keys

Authorized users can use the GenerateDataKey API (and related APIs) to request a specific type of data key or a random key of arbitrary length. This topic provides a simplified view of this API operation. For details, see the GenerateDataKey APIs in the *AWS Key Management Service API Reference*.

- GenerateDataKey
- GenerateDataKeyWithoutPlaintext
- GenerateDataKeyPair
- GenerateDataKeyPairWithoutPlaintext

The following is the `GenerateDataKey` request syntax.

```
{
 “EncryptionContext”: {“string” : “string”},
 “GrantTokens”: [“string”],
 “KeyId”: “string”,
 “NumberOfBytes”: “number”
```

```
}
```

The request accepts the following data in JSON format.

KeyId

Key identifier of the key used to encrypt the data key. This value must identify a symmetric encryption KMS key.

This parameter is required.

NumberOfBytes

An integer that contains the number of bytes to generate. This parameter is required.

Caller must provide either `KeySpec` or `NumberOfBytes`, but not both.

EncryptionContext

(Optional) Name-value pair that contains additional data to authenticate during the encryption and decryption processes that use the key.

GrantTokens

(Optional) A list of grant tokens that represent grants that provide permissions to generate or use a key. For more information on grants and grant tokens, see Authentication and access control for AWS KMS in the *AWS Key Management Service Developer Guide*.

After authenticating the command, AWS KMS, acquires the current active EKT associated with the KMS key. It passes the EKT along with your provided request and any encryption context to an HSM over a protected session between the AWS KMS host and an HSM in the domain.

The HSM does the following:

1. Generates the requested secret material and hold it in volatile memory.
2. Decrypts the *EKT* matching the key ID of the KMS key that is defined in the request to obtain the active $HBK = Decrypt(DK_i, EKT)$.
3. Generates a random nonce *N*.
4. Generates a 256-bit AES-GCM derived encryption key *K* from *HBK* and *N*.
5. Encrypts the secret material *ciphertext = Encrypt(K, context, secret)*.

`GenerateDataKey` returns the plaintext secret material and the ciphertext to you over the secure channel between the AWS KMS host and the HSM. AWS KMS then sends it to you over the TLS session. AWS KMS does not retain the plaintext or ciphertext. Without possession of the *ciphertext*, the encryption context, and the authorization to use the *KMS key*, the underlying secret cannot be returned.

The following is the response syntax.

```
{
 "CiphertextBlob": "blob",
 "KeyId": "string",
 "Plaintext": "blob"
}
```

The management of data keys is left to you as the application developer. For best practice client-side encryption with AWS KMS data keys (but not data key pairs), you can use the AWS Encryption SDK.

Data keys can be rotated at any frequency. Further, the data key can be reencrypted under a different KMS key or a rotated KMS key using the `ReEncrypt` API operation. For details, see ReEncrypt in the *AWS Key Management Service API Reference*.

Encrypt

A basic function of AWS KMS is to encrypt an object under a KMS key. By design, AWS KMS provides low latency cryptographic operations on HSMs. Thus there is a limit of 4 KB on the amount of plaintext that can be encrypted in a direct call to the encrypt function. The AWS Encryption SDK can be used to encrypt larger messages. AWS KMS, after authenticating the command, acquires the current active EKT pertaining to the KMS key. It passes the EKT, along with the plaintext and encryption context, to any available HSM in the Region. These are sent over an authenticated session between the AWS KMS host and an HSM in the domain.

The HSM runs the following:

1. Decrypts the EKT to obtain the *HBK = Decrypt(DK_i, EKT)* .
2. Generates a random nonce *N*.
3. Derives a 256-bit AES-GCM derived encryption key *K* from *HBK* and *N*.
4. Encrypts the plaintext *ciphertext = Encrypt(K, context, plaintext)*.

The ciphertext value is returned to you, and neither the plaintext data or ciphertext is retained anywhere in the AWS infrastructure. Without possession of the *ciphertext* and the encryption context, and the authorization to use the KMS key, the underlying plaintext cannot be returned.

Decrypt

A call to AWS KMS to decrypt a ciphertext value accepts an encrypted value ciphertext and an encryption context. AWS KMS authenticates the call using AWS signature version 4 signed requests and extracts the HBKID for the wrapping key from the ciphertext. The HBKID is used to obtain the *EKT* required to decrypt the ciphertext, the key ID, and the policy for the key ID. The request is authorized based on the key policy, grants that may be present, and any associated IAM policies that reference the key ID. The `Decrypt` function is analogous to the encryption function.

The following is the `Decrypt` request syntax.

```
{
 "CiphertextBlob": "blob",
 "EncryptionContext": { "string" : "string" }
 "GrantTokens": ["string"]
}
```

The following are the request parameters.

CiphertextBlob

Ciphertext including metadata.

EncryptionContext

(Optional) The encryption context. If this was specified in the `Encrypt` function, it must be specified here or the decryption operation fails. For more information, see Encryption context in the *AWS Key Management Service Developer Guide*.

GrantTokens

(Optional) A list of grant tokens that represent grants that provide permissions to perform decryption.

The *ciphertext* and the *EKT* are sent, along with the encryption context, over an authenticated session to an HSM for decryption.

The HSM runs the following:

1. Decrypts the *EKT* to obtain the *HBK = Decrypt(DK_i, EKT)* .
2. Extracts the nonce *N* from the *ciphertext* structure.
3. Regenerates a 256-bit AES-GCM derived encryption key *K* from *HBK* and *N*.
4. Decrypts the *ciphertext* to obtain *plaintext = Decrypt(K, context, ciphertext)* .

The resulting key ID and plaintext are returned to the AWS KMS host over the secure session and then back to the calling customer application over a TLS connection.

The following is the response syntax.

```
{
    "KeyId": "string",
    "Plaintext": blob
}
```

If the calling application wants to ensure that the authenticity of the plaintext, it must verify that the key ID returned is the one expected.

Reencrypting an encrypted object

An existing customer ciphertext encrypted under one KMS key can be reencrypted to another KMS key through a reencrypt command. Reencrypt encrypts data on the server side with a new KMS key without exposing the plaintext of the key on the client side. The data is first decrypted and then encrypted.

The following is the request syntax.

```
{
 "CiphertextBlob": "blob",
 "DestinationEncryptionContext": { "string" : "string" },
 "DestinationKeyId": "string",
 "GrantTokens": ["string"],
       "SourceKeyId": "string",
 "SourceEncryptionContext": { "string" : "string"}
}
```

The request accepts the following data in JSON format.

CiphertextBlob

Ciphertext of the data to reencrypt.

DestinationEncryptionContext

(Optional) Encryption context to be used when the data is reencrypted.

DestinationKeyId

Key identifier of the key used to reencrypt the data.

GrantTokens

(Optional) A list of grant tokens that represent grants that provide permissions to perform decryption.

SourceKeyId

(Optional) Key identifier of the key used to decrypt the data.

SourceEncryptionContext

(Optional) Encryption context used to encrypt and decrypt the data specified in the `CiphertextBlob` parameter.

The process combines the decrypt and encrypt operations of the previous descriptions: The customer ciphertext is decrypted under the initial HBK referenced by the customer ciphertext to the current HBK under the intended KMS key. When the KMS keys used in this command are the same, this command moves the customer ciphertext from an old version of an HBK to the latest version of an HBK.

The following is the response syntax.

```
{
    "CiphertextBlob": blob,
    "DestinationEncryptionAlgorithm": "string",
    "KeyId": "string",
    "SourceEncryptionAlgorithm": "string",
    "SourceKeyId": "string"
}
```

If the calling application wants to ensure the authenticity of the underlying plaintext, it must verify the SourceKeyId returned is the one expected.

AWS KMS internal operations

AWS KMS internals are required to scale and secure HSMs for a globally distributed key management service.

Topics

Domains and domain state

A cooperative collection of trusted internal AWS KMS entities within an AWS Region is referred to as a domain. A domain includes a set of trusted entities, a set of rules, and a set of secret keys, called domain keys. The domain keys are shared among HSMs that are members of the domain. A domain state consists of the following fields.

Name

A domain name to identify this domain.

Members

A list of HSMs that are members of the domain, including their public signing key and public agreement keys.

Operators

A list of entities, public signing keys, and a role (AWS KMS operator or service host) that represents the operators of this service.

Rules

A list of quorum rules for each command that must be satisfied to run a command on the HSM.

Domain keys

A list of domain keys (symmetric keys) currently in use within the domain.

The full domain state is available only on the HSM. The domain state is synchronized between HSM domain members as an exported domain token.

Domain keys

All the HSMs in a domain share a set of domain keys, $\{DK_r\}$. These keys are shared through a domain state export routine. The exported domain state can be imported into any HSM that is a member of the domain.

The set of domain keys, $\{DK_r\}$, always includes one active domain key, and several deactivated domain keys. Domain keys are rotated daily to ensure that AWS complies with Recommendation for Key Management - Part 1. During domain key rotation, all existing KMS keys encrypted under the outgoing domain key are re-encrypted under the new active domain key. The active domain key is used to encrypt any new EKTs. The expired domain keys can be used only to decrypt previously encrypted EKTs for a number of days equivalent to the number of recently rotated domain keys.

Exported domain tokens

There is a regular need to synchronize state between domain participants. This is accomplished through exporting the domain state whenever a change is made to the domain. The domain state is exported as an exported domain token.

Name

A domain name to identify this domain.

Members

A list of HSMs that are members of the domain, including their signing and agreement public keys.

Operators

A list of entities, public signing keys, and a role that represents the operators of this service.

Rules

A list of quorum rules for each command that must be satisfied to run a command on an HSM domain member.

Encrypted domain keys

Envelope-encrypted domain keys. The domain keys are encrypted by the signing member for each of the members listed above, enveloped to their public agreement key.

Signature

A signature on the domain state produced by an HSM, necessarily a member of the domain that exported the domain state.

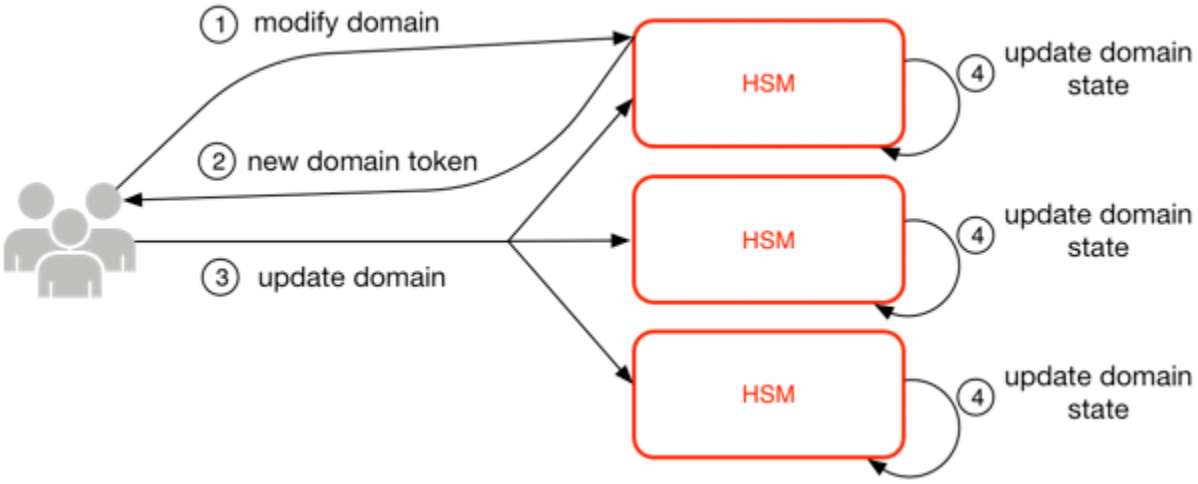
The exported domain token forms the fundamental source of trust for entities operating within the domain.

Managing domain states

The domain state is managed through quorum-authenticated commands. These changes include modifying the list of trusted participants in the domain, modifying the quorum rules for running HSM commands, and periodically rotating the domain keys. These commands are authenticated on a per-command basis as opposed to authenticated session operations, as shown in the following image.

In its initialized and operational state, an HSM contains a set of self-generated asymmetric identity keys, a signing key pair, and a key-establishment key pair. Through a manual process, an AWS KMS operator can establish an initial domain to be created on a first HSM in a Region. This initial domain consists of a full domain state as defined previously in this topic. It is installed through a join command to each of the defined HSM members in the domain.

After an HSM has joined an initial domain, it is bound to the rules that are defined in that domain. These rules govern the commands that use customer cryptographic keys or make changes to the host or domain state. The authenticated session API operations that use your cryptographic keys have been defined earlier.



The foregoing image depicts how a domain state gets modified. The process consists of four steps:

1. A quorum-based command is sent to an HSM to modify the domain.
2. A new domain state is generated and exported as a new exported domain token. The state on the HSM is not modified, meaning that the change is not enacted on the HSM.
3. A second command is sent to each of the HSMs in the newly exported domain token to update their domain state with the new domain token.
4. The HSMs listed in the new exported domain token can authenticate the command and the domain token. They can also unpack the domain keys to update the domain state on all HSMs in the domain.

HSMs do not communicate directly with one another. Instead, a quorum of operators requests a change to the domain state that results in a new exported domain token. A service host member of the domain is used to distribute the new domain state to every HSM in the domain.

The leaving and joining of a domain are done through the HSM management functions. The modification of the domain state is done through the domain management functions.

Leave domain

Causes an HSM to leave a domain, deleting all remnants and keys of that domain from memory.

Join domain

Causes an HSM to join a new domain or update its current domain state to the new domain state. The existing domain is used as source of the initial set of rules to authenticate this message.

Create domain

Causes a new domain to be created on an HSM. Returns a first domain token that can be distributed to member HSMs of the domain.

Modify operators

Adds or removes operators from the list of authorized operators and their roles in the domain.

Modify members

Adds or removes an HSM from the list of authorized HSMs in the domain.

Modify rules

Modifies the set of quorum rules that are required to run commands on an HSM.

Rotate domain keys

Causes a new domain key to be created and marked as the active domain key. This moves the existing active key to a deactivated key and removes the oldest deactivated key from the domain state.

Internal communication security

Commands between the service hosts or AWS KMS operators and the HSMs are secured through two mechanisms depicted in Authenticated sessions: a quorum-signed request method and an authenticated session using an HSM-service host protocol.

The quorum-signed commands are designed so that no single operator can modify the critical security protections that the HSMs provide. The commands that run over the authenticated sessions help ensure that only authorized service operators can perform operations involving KMS keys. All customer-bound secret information is secured across the AWS infrastructure.

Key establishment

To secure internal communications, AWS KMS uses two different key establishment methods. The first is defined as C(1, 2, ECC DH) in Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revision 2). This scheme has an initiator with a static signing key. The initiator generates and signs an ephemeral elliptic curve Diffie-Hellman (ECDH) key, intended for a recipient with a static ECDH agreement key. This method uses one ephemeral key and two static keys using ECDH. That is the derivation of the label C(1, 2, ECC DH). This method is sometimes called one-pass ECDH.

The second key establishment method is C(2, 2, ECC, DH). In this scheme, both parties have a static signing key, and they generate, sign, and exchange an ephemeral ECDH key. This method uses two static keys and two ephemeral keys, each using ECDH. That is the derivation of the label C(2, 2, ECC, DH). This method is sometimes called ECDH ephemeral or ECDHE. All ECDH keys are generated on the curve secp384r1 (NIST-P384).

HSM security boundary

The inner security boundary of AWS KMS is the HSM. The HSM has a proprietary interface and no other active physical interfaces in its operational state. An operational HSM is provisioned during initialization with the necessary cryptographic keys to establish its role in the domain. Sensitive cryptographic materials of the HSM are only stored in volatile memory and erased when the HSM moves out of the operational state, including intended or unintended shutdowns or resets.

The HSM API operations are authenticated either by individual commands or over a mutually authenticated confidential session established by a service host.

Quorum-signed commands

Quorum-signed commands are issued by operators to HSMs. This section describes how quorum-based commands are created, signed, and authenticated. These rules are fairly simple. For example, command *Foo* requires two members from role *Bar* to be authenticated. There are three steps in the creation and verification of a quorum-based command. The first step is the initial command creation; the second is the submission to additional operators to sign; and the third is the verification and execution.

For the purpose of introducing the concepts, assume that there is an authentic set of operator's public keys and roles $\{QOS_s\}$, and a set of quorum-rules $QR = \{Command_i, Rule_{\{i,\ t\}}\}$ where each *Rule* is a set of roles and minimum number $N\ \{Role_t, N_t\}$. For a command to satisfy the quorum rule, the command dataset must be signed by a set of operators listed in $\{QOS_s\}$ such that they meet one of the rules listed for that command. As mentioned earlier, the set of quorum rules and operators are stored in the domain state and the exported domain token.

In practice, an initial signer signs the command $Sig_1 = Sign(dO_{p1}, Command)$. A second operator also signs the command $Sig_2 = Sign(dO_{p2}, Command)$. The doubly signed message is sent to an HSM for execution. The HSM performs the following:

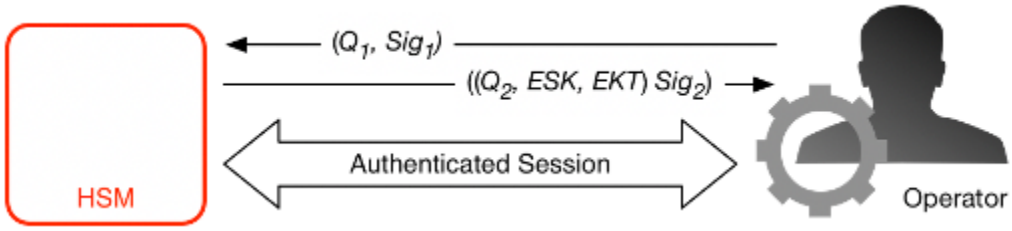
1. For each signature, it extracts the signer's public key from the domain state and verifies the signature on the command.
2. It verifies that the set of signers satisfies a rule for the command.

Authenticated sessions

Your key operations run between the externally facing AWS KMS hosts and the HSMs. These commands pertain to the creation and use of cryptographic keys and secure random number generation. The commands run over a session-authenticated channel between the service hosts and the HSMs. In addition to the need for authenticity, these sessions require confidentiality. Commands running over these sessions include the returning of cleartext data keys and decrypted messages intended for you. To ensure that these sessions cannot be subverted through man-in-the-middle attacks, sessions are authenticated.

This protocol performs a mutually authenticated ECDHE key agreement between the HSM and the service host. The exchange is initiated by the service host and completed by the HSM. The HSM also returns a session key (SK) encrypted by the negotiated key and an exported key token that contains the session key. The exported key token contains a validity period, after which the service host must renegotiate a session key.

A service host is a member of the domain and has an identity-signing key pair $(dHOS_i, QHOS_i)$ and an authentic copy of the HSMs' identity public keys. It uses its set of identity-signing keys to securely negotiate a session key that can be used between the service host and any HSM in the domain. The exported key tokens have a validity period associated with them, after which a new key must be negotiated.



The process begins with the service host recognition that it requires a session key to send and receive sensitive communication flows between itself and an HSM member of the domain.

1. A service host generates an ECDH ephemeral key pair (d_1, Q_1) and signs it with its identity key $Sig1 = Sign(dOS,Q_1)$.
2. The HSM verifies the signature on the received public key using its current domain token and creates an ECDH ephemeral key pair (d_2, Q_2). It then completes the ECDH-key-exchange according to Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised) to form a negotiated 256-bit AES-GCM key. The HSM generates a fresh 256-bit AES-GCM session key. It encrypts the session key with the negotiated key to form the encrypted session key (ESK). It also encrypts the session key under the domain key as an exported key token *EKT*. Finally, it signs a return value with its identity key pair $Sig_2 = Sign(dHSK, (Q_2, ESK, EKT))$.
3. The service host verifies the signature on the received keys using its current domain token. The service host then completes the ECDH key exchange according to Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised). It next decrypts the ESK to obtain the session key SK.

During the validity period in the *EKT*, the service host can use the negotiated session key *SK* to send envelope-encrypted commands to the HSM. Every service-host-initiated command over this authenticated session includes the *EKT*. The HSM responds using the same negotiated session key SK.

Replication process for multi-Region keys

AWS KMS uses a cross-Region replication mechanism to copy the key material in a KMS key from an HSM in one AWS Region to an HSM in a different AWS Region. For this mechanism to work, the KMS key that is being replicated must be a multi-Region key. When replicating a KMS key from one Region to another, the HSMs in the Regions cannot communicate directly, because they're in isolated networks. Instead, the messages exchanged during the cross-Region replication are delivered by a proxy service.

During cross-Region replication, every message generated by an AWS KMS HSM is cryptographically signed using a *replication signing key*. Replication signing keys (RSKs) are ECDSA keys on the NIST P-384 curve. Every Region owns at least one RSK, and the public component of each RSK is shared with every other Region in the same AWS partition.

The cross-Region replication process to copy key material from Region A to Region B works as follows:

1. The HSM in Region B generates an ephemeral ECDH key on the NIST P-384 curve, *Replication Agreement Key B* (RAKB). The public component of RAKB is sent to an HSM in Region A by the proxy service.
2. The HSM in Region A receives the public component of RAKB and then generates another ephemeral ECDH key on the NIST P-384 curve, *Replication Agreement Key A* (RAKA). The HSM runs the ECDH key establishment scheme on RAKA and the public component of RAKB, and derives a symmetric key from the output, the *Replication Wrapping Key* (RWK). The RWK is used to encrypt the key material of the multi-Region KMS key that is being replicated.
3. The public component of RAKA and the key material encrypted with the RWK are sent to the HSM in Region B through the proxy service.
4. The HSM in Region B receives the public component of RAKA and the key material encrypted using the RWK. The HSM derives by RWK by running the ECDH key establishment scheme on RAKB and the public component of RAKA.
5. The HSM in Region B use the RWK to decrypt the key material from Region A.

Durability protection

Additional service durability for keys generated by the service is provided by the use of offline HSMs, multiple nonvolatile storage of exported domain tokens, and redundant storage of encrypted KMS keys. The offline HSMs are members of the existing domains. With the exception of not being online and participating in the regular domain operations, the offline HSMs appear identically in the domain state as the existing HSM members.

The durability design is intended to protect all KMS keys in a Region should AWS experience a wide-scale loss of either the online HSMs or the set of KMS keys stored within our primary storage system. AWS KMS keys with imported key material are not included under the durability protections afforded other KMS keys. In the event of a Regionwide failure in AWS KMS, imported key material may need to be reimported into a KMS key.

The offline HSMs, and the credentials to access them, are stored in safes within monitored safe rooms in multiple independent geographical locations. Each safe requires at least one AWS security officer and one AWS KMS operator, from two independent teams in AWS, to obtain these materials. The use of these materials is governed by internal policy requiring a quorum of AWS KMS operators to be present.

Reference

Use the following reference material to get information about abbreviations, keys, contributors, and sources cited in this document.

Topics

Abbreviations

The following list illuminates abbreviations referenced in this document.

AES

Advanced Encryption Standard

CDK

customer data key

DK

domain key

ECDH

Elliptic Curve Diffie-Hellman

ECDHE

Elliptic Curve Diffie-Hellman Ephemeral

ECDSA

Elliptic Curve Digital Signature Algorithm

EKT

exported key token

ESK

encrypted session key

GCM

Galois Counter Mode

HBK

HSM backing key

HBKID

HSM backing key identifier

HSM

hardware security module

RSA

Rivest Shamir and Adleman (cryptologic)

secp384r1

Standards for Efficient Cryptography prime 384-bit random curve 1

SHA256

Secure Hash Algorithm of digest length 256-bits

Keys

The following list defines the keys referenced in this document.

HBK

HSM backing key: HSM backing keys are 256-bit root keys, from which specific use keys are derived.

DK

Domain key: A domain key is a 256-bit AES-GCM key. It is shared among all the members of a domain and is used to protect HSM backing keys material and HSM-service host session keys.

DKEK

Domain key encryption key: A domain key encryption Key is an AES-256-GCM key generated on a host and used for encrypting the current set of domain keys synchronizing domain state across the HSM hosts.

(dHAK,QHAK)

HSM agreement key pair: Every initiated HSM has a locally generated Elliptic Curve Diffie-Hellman agreement key pair on the curve secp384r1 (NIST-P384).

(dE, QE)

Ephemeral agreement key pair: HSM and service hosts generate ephemeral agreement keys. These are Elliptic Curve Diffie-Hellman keys on the curve secp384r1 (NIST-P384). These are generated in two use cases: to establish a host-to-host encryption key to transport domain key encryption keys in domain tokens and to establish HSM-service host session keys to protect sensitive communications.

(dHSK,QHSK)

HSM signature key pair: Every initiated HSM has a locally generated Elliptic Curve Digital Signature key pair on the curve secp384r1 (NIST-P384).

(dOS,QOS)

Operator signature key pair: Both the service host operators and AWS KMS operators have an identity signing key used to authenticate itself to other domain participants.

K

Data encryption key: A 256-bit AES-GCM key derived from an HBK using the NIST SP800-108 KDF in counter mode using HMAC with SHA256.

SK

Session key: A session key is created as a result of an authenticated Elliptic Curve Diffie-Hellman key exchanged between a service host operator and an HSM. The purpose of the exchange is to secure communication between the service host and the members of the domain.

Contributors

The following individuals and organizations contributed to this document:

- Ken Beer, General Manager - KMS, AWS Cryptography
- Matthew Campagna, Principal Security Engineer, AWS Cryptography

Bibliography

For information about the AWS Key Management Service HSMs, go to the NIST Computer Security Resource Center Cryptographic Module Validation Program search page and search for **AWS Key Management Service HSM**.

Amazon Web Services, General Reference (Version 1.0), "Signing AWS API Request," http://docs.aws.amazon.com/general/latest/gr/signing_aws_api_requests.html.

Amazon Web Services, "What is the AWS Encryption SDK," http://docs.aws.amazon.com/encryption-sdk/latest/developer-guide/introduction.html.

Federal Information Processing Standards Publications, FIPS PUB 180-4. *Secure Hash Standard*, August 2012. Available from https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf.

Federal Information Processing Standards Publication 197, *Announcing the Advanced Encryption Standard (AES)*, November 2001. Available from http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

Federal Information Processing Standards Publication 198-1, *The Keyed-Hash Message Authentication Code (HMAC)*, July 2008. Available from http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf.

NIST Special Publication 800-52 Revision 2, Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations, August 2019. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf.

PKCS#1 v2.2: *RSA Cryptography Standard* (RFC 8017), Internet Engineering Task Force (IETF), November 2016. https://tools.ietf.org/html/rfc8017.

Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, NIST Special Publication 800-38D, November 2007. Available from http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf.

Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices, NIST Special Publication 800-38E, January 2010. Available from https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38e.pdf.

Recommendation for Key Derivation Using Pseudorandom Functions, NIST Special Publication 800-108, October 2009, Available from https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-108.pdf.

Recommendation for Key Management - Part 1: General (Revision 5), NIST Special Publication 800-57A, May 2020, Available from https://doi.org/10.6028/NIST.SP.800-57pt1r5.

Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised), NIST Special Publication 800-56A Revision 3, April 2018. Available from https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf.

Recommendation for Random Number Generation Using Deterministic Random Bit Generators, NIST Special Publication 800-90A Revision 1, June 2015, Available from https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf.

SEC 2: *Recommended Elliptic Curve Domain Parameters*, Standards for Efficient Cryptography Group, Version 2.0, 27 January 2010.

Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS), Brown, D., Turner, S., Internet Engineering Task Force, July 2010, http://tools.ietf.org/html/rfc5753/.

X9.62-2005: Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), American National Standards Institute, 2005.

Document History for AWS KMS Cryptographic Details

The following table describes important changes to the documentation for AWS Key Management Service Cryptographic Details. We also update the documentation frequently to address the feedback that you send to us.

Change	Description	Date
Updated content	Added details about the implementation of the AWS KMS `ReplicateKey` operation.	October 28, 2021
Documentation change	Replace the term *customer master key (CMK)* with *AWS KMS key* and *KMS key*.	August 30, 2021
Initial release	Created this guide from the KMS Cryptographic Details technical paper	December 30, 2020