



Entwicklerhandbuch

AWS Encryption SDK



AWS Encryption SDK: Entwicklerhandbuch

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

Die Handelsmarken und Handelsaufmachung von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, durch die Kunden irregeführt werden könnten oder Amazon in schlechtem Licht dargestellt oder diskreditiert werden könnte. Alle anderen Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

Was ist der AWS Encryption SDK?	1
Entwickelt in Open-Source-Repositories	3
Kompatibilität mit Verschlüsselungsbibliotheken und -services	3
Support und Wartung	4
Mehr lernen	5
Senden von Feedback	6
Konzepte	6
Envelope-Verschlüsselung	7
Datenschlüssel	9
Schlüssel zum Umschließen	10
Schlüsselanhänger und Hauptschlüsselanbieter	11
Verschlüsselungskontext	12
Verschlüsselte Nachricht	14
Algorithmen-Paket	14
Manager von kryptographischen Materialien	15
Symmetrische und asymmetrische Verschlüsselung	16
Wichtiges Engagement	16
Verpflichtungspolitik	18
Digitale Signaturen	20
Wie das SDK funktioniert	21
Wie AWS Encryption SDK verschlüsselt der Daten	21
Wie AWS Encryption SDK entschlüsselt der eine verschlüsselte Nachricht	22
Unterstützte Algorithmen-Pakete	23
Empfohlen: AES-GCM mit Schlüsselableitung, Signatur und Schlüsselverpflichtung	23
Andere unterstützte Algorithmen-Pakete	24
Interaktion mit AWS KMS	26
Bewährte Methoden	28
Konfiguration der SDK	33
Auswahl einer Programmiersprache	33
Auswahl von Schlüsseln zum Umbrechen	34
Verwenden Sie mehrere Regionen AWS KMS keys	35
Auswahl einer Algorithmus-Suite	57
Beschränkung verschlüsselter Datenschlüssel	66
Einen Discovery-Filter erstellen	71

Festlegung einer Verpflichtungspolitik	73
Arbeiten mit Streaming-Daten	74
Zwischenspeichern von Datenschlüsseln	74
Verwenden von Schlüsselbunden	75
Funktionsweise von Schlüsselbunden	76
Schlüsselbund-Kompatibilität	78
Unterschiedliche Anforderungen für Verschlüsselungsschlüsselringe	78
Kompatible Schlüsselbunde und Masterschlüssel-Anbieter	78
Einen Schlüsselbund wählen	80
AWS KMS Schlüsselanhänger	81
AWS KMS Hierarchische Schlüsselanhänger	102
AWS KMS ECDH-Schlüsselbünde	127
Unformatierte Schlüsselanhänger AES	134
RSARaw-Schlüsselringe	138
Rohe Schlüsselanhänger ECDH	144
Multi-Schlüsselbunde	153
Programmiersprachen	159
C	159
Installation	160
Verwenden des C SDK	161
Beispiele	166
.NET	175
Installieren und Erstellen	177
Debugging	177
AWS KMS-Schlüsselbünde	178
Erforderlicher Verschlüsselungskontext (CMM)	181
Beispiele	183
Java	192
Voraussetzungen	193
Installation	194
AWS KMS Schlüsselringe	195
Erforderlicher Verschlüsselungskontext CMM	198
Beispiele	201
JavaScript	214
Kompatibilität	215
Installation	217

Module	218
Beispiele	221
Python	228
Voraussetzungen	228
Installation	229
Beispiele	230
Befehlszeilenschnittstelle	241
Installieren der -CLI	243
Die CLI verwenden	246
Beispiele	261
Syntax und Parameterreferenz	286
Versionen	301
Datenschlüssel-Caching	305
Das Datenschlüssel-Caching verwenden	306
Verwenden von Datenschlüssel-Caching: S tep-by-step	307
Beispiel für das Datenschlüssel-Caching: Verschlüsseln einer Zeichenfolge	315
Festlegen von Cache-Sicherheitsschwellenwerten	331
Weitere Informationen zum Datenschlüssel-Caching	333
Wie das Datenschlüssel-Caching funktioniert	333
Erstellen eines Cache für kryptografische Materialien	337
Erstellen eines Managers von kryptographischen Materialien, der Caching verwendet	338
Was befindet sich in einem Datenschlüssel-Cache-Eintrag?	339
Verschlüsselungskontext: Wie Cache-Einträge ausgewählt werden	340
Benutzt meine Anwendung zwischengespeicherte Datenschlüssel?	340
Beispiel für das Datenschlüssel-Caching	341
Lokale Cache-Ergebnisse	342
Beispiel-Code	343
AWS CloudFormation-Vorlage	355
Versionen der AWS Encryption SDK	370
C	371
C#/. NET	372
Befehlszeilenschnittstelle (CLI)	372
Java	375
JavaScript	377
Python	379
Einzelheiten zur Version	380

Versionen vor 1.7. x	381
Version 1.7. x	381
Version 2.0. x	384
Version 2.2. x	386
Version 2.3. x	387
Migrieren Sie Ihre AWS Encryption SDK	388
So migrieren und implementieren Sie	390
Phase 1: Aktualisieren Sie Ihre Anwendung auf den neuesten Stand 1.X-xAusführung	390
Phase 2: Aktualisieren Sie Ihre Anwendung auf die neueste Version	392
Aktualisierung der AWS KMS Hauptschlüsselanbieter	393
Umstellung auf den strikten Modus	394
In den Discovery-Modus migrieren	398
Wird aktualisiert AWS KMS Schlüsselringe	401
Festlegung Ihrer Verpflichtungspolitik	404
Wie legen Sie Ihre Verpflichtungsrichtlinie fest	405
Problembehandlung bei der Migration auf die neuesten Versionen	413
Veraltete oder entfernte Objekte	414
Konfigurationskonflikt: Commitment Policy- und Algorithmus	414
Konfigurationskonflikt: Verpflichtungspolitik und Chiffretext	415
Validierung der wichtigsten Verpflichtungen fehlgeschlagen	416
Andere Verschlüsselungsfehler	416
Andere Fehler bei der Entschlüsselung	416
Überlegungen zum Rollback	417
Häufig gestellte Fragen	418
Referenz	424
Nachrichtenformat – Referenz	424
Header-Struktur	425
Textstruktur	434
Footer-Struktur	439
Nachrichtenformat – Beispiele	440
Gerahmte Daten (Nachrichtenformat, Version 1)	441
Frame-Daten (Nachrichtenformat, Version 2)	444
Daten ohne Frames (Nachrichtenformat, Version 1)	446
Referenz auf den Körper AAD	450
Algorithmen – Referenz	452
Initialisierungsvektor – Referenz	457

AWS KMS Technische Details zum hierarchischen Schlüsselbund	458
Dokumentverlauf	460
Neueste Aktualisierungen	460
Frühere Aktualisierungen	463
.....	cdlxv

Was ist der AWS Encryption SDK?

Dabei AWS Encryption SDK handelt es sich um eine clientseitige Verschlüsselungsbibliothek, die entwickelt wurde, um jedem das Verschlüsseln und Entschlüsseln von Daten mithilfe von Industriestandards und bewährten Methoden zu erleichtern. Es ermöglicht Ihnen, sich auf die Kernfunktionalität Ihrer Anwendung zu konzentrieren und nicht darauf, wie Sie Ihre Daten am besten verschlüsseln und entschlüsseln können. Die AWS Encryption SDK wird kostenlos unter der Apache 2.0-Lizenz zur Verfügung gestellt.

Das AWS Encryption SDK beantwortet Fragen wie die folgenden für Sie:

- Welche Verschlüsselungsalgorithmus sollte ich verwenden?
- Wie oder in welchem Modus sollte ich diesen Algorithmus verwenden?
- Wie kann ich den Verschlüsselungsschlüssel generieren?
- Wie kann ich den Verschlüsselungsschlüssel schützen, und wo sollte ich ihn speichern?
- Wie kann ich meine verschlüsselten Daten portierbar machen?
- Wie kann ich sicherstellen, dass der beabsichtigte Empfänger meine verschlüsselten Daten lesen kann?
- Wie kann ich sicherstellen, dass meine verschlüsselten Daten zwischen dem Schreiben und dem Lesen nicht verändert werden?
- Wie verwende ich die Datenschlüssel, die AWS KMS zurückgegeben werden?

Mit dem AWS Encryption SDK definieren Sie einen [Hauptschlüsselanbieter](#) (Python) oder einen [Schlüsselbund](#) (C, C#, .NET, Java und JavaScript), der bestimmt, welche Wrapping-Schlüssel Sie zum Schutz Ihrer Daten verwenden. Anschließend verschlüsseln und entschlüsseln Sie Ihre Daten mit einfachen Methoden, die von der bereitgestellt werden. AWS Encryption SDK Das AWS Encryption SDK macht den Rest.

Ohne das AWS Encryption SDK investieren Sie möglicherweise mehr Aufwand in die Entwicklung einer Verschlüsselungslösung als in die Kernfunktionalität Ihrer Anwendung. The AWS Encryption SDK beantwortet diese Fragen, indem es die folgenden Informationen bereitstellt.

Eine Standard-Implementierung, die die bewährten Methoden der Kryptografie befolgt

Standardmäßig AWS Encryption SDK generiert der einen eindeutigen Datenschlüssel für jedes Datenobjekt, das er verschlüsselt. Dies entspricht den bewährten Methoden der Kryptografie, eindeutige Datenschlüssel für jede Verschlüsselungsoperation zu verwenden.

Der AWS Encryption SDK verschlüsselt Ihre Daten mithilfe eines sicheren, authentifizierten, symmetrischen Schlüsselalgorithmus. Weitere Informationen finden Sie unter [the section called “Unterstützte Algorithmen-Pakete”](#).

Ein Framework zum Schutz von Datenschlüsseln durch Wrapping Keys

Das AWS Encryption SDK schützt die Datenschlüssel, die Ihre Daten verschlüsseln, indem es sie unter einem oder mehreren Umschließungsschlüsseln verschlüsselt. Durch die Bereitstellung eines Frameworks zum Verschlüsseln von Datenschlüsseln mit mehr als einem Umschließungsschlüssel AWS Encryption SDK trägt das dazu bei, dass Ihre verschlüsselten Daten portabel sind.

Verschlüsseln Sie beispielsweise Daten mit einem Eingang AWS KMS und AWS KMS key einem Schlüssel aus Ihrer lokalen Umgebung. HSM Sie können einen der Wrapping-Schlüssel verwenden, um die Daten zu entschlüsseln, falls einer nicht verfügbar ist oder der Anrufer nicht berechtigt ist, beide Schlüssel zu verwenden.

Eine formatierte Nachricht, die verschlüsselte Datenschlüssel mit den verschlüsselten Daten speichert

Der AWS Encryption SDK speichert die verschlüsselten Daten und den verschlüsselten Datenschlüssel zusammen in einer [verschlüsselten Nachricht](#), die ein definiertes Datenformat verwendet. Das bedeutet, dass Sie die Datenschlüssel, die Ihre Daten verschlüsseln, nicht nachverfolgen oder schützen müssen, da dies für Sie AWS Encryption SDK erledigt wird.

Einige Sprachimplementierungen von AWS Encryption SDK erfordern eine AWS SDK, aber die benötigen AWS Encryption SDK keine AWS-Konto und sie hängen von keinem AWS Dienst ab. Sie benötigen eine AWS-Konto nur, wenn Sie sie [AWS KMS keys](#) zum Schutz Ihrer Daten verwenden möchten.

Entwickelt in Open-Source-Repositories

Das AWS Encryption SDK wurde in Open-Source-Repositories entwickelt. Sie können diese Repositories verwenden, um den Code einzusehen, Probleme zu lesen und einzureichen sowie Informationen zu finden, die für Ihre Sprachimplementierung spezifisch sind.

- AWS-Verschlüsselungs-SDK for C — [aws-encryption-sdk-c](#)
- AWS Encryption SDK für .NET — [aws-encryption-sdk-net](#) Verzeichnis des aws-encryption-sdk-dafny Repositories.
- AWS Verschlüsselung CLI — [aws-encryption-sdk-cli](#)
- AWS-Verschlüsselungs-SDK for Java — [aws-encryption-sdk-java](#)
- AWS-Verschlüsselungs-SDK for JavaScript — [aws-encryption-sdk-javascript](#)
- AWS-Verschlüsselungs-SDK for Python — [aws-encryption-sdk-python](#)

Kompatibilität mit Verschlüsselungsbibliotheken und -services

Das AWS Encryption SDK wird in mehreren [Programmiersprachen](#) unterstützt. Alle Sprachimplementierungen sind interoperabel. Sie können mit einer Sprachimplementierung verschlüsseln und mit einer anderen entschlüsseln. Die Interoperabilität ist möglicherweise von Spracheinschränkungen abhängig. Wenn dies der Fall ist, werden diese Einschränkungen im Thema zur Sprachimplementierung beschrieben. Außerdem müssen Sie beim Verschlüsseln und Entschlüsseln kompatible Schlüsselbünde oder Masterschlüssel und Masterschlüsselanbieter verwenden. Details hierzu finden Sie unter [the section called “Schlüsselbund-Kompatibilität”](#).

Sie können jedoch AWS Encryption SDK nicht mit anderen Bibliotheken zusammenarbeiten. Da jede Bibliothek verschlüsselte Daten in einem anderen Format zurückgibt, können Sie nicht mit einer Bibliothek verschlüsseln und mit einer anderen entschlüsseln.

DynamoDB Encryption Client und clientseitige Amazon S3 S3-Verschlüsselung

Die Daten, die mit dem [DynamoDB Encryption Client oder der clientseitigen Amazon S3 S3-Verschlüsselung](#) verschlüsselt wurden, AWS Encryption SDK können nicht entschlüsselt werden. [Diese Bibliotheken können die von ihnen zurückgegebene verschlüsselte Nachricht nicht entschlüsseln](#). AWS Encryption SDK

AWS Key Management Service (AWS KMS)

Sie AWS Encryption SDK können [Datenschlüssel verwenden AWS KMS keys, um Ihre Daten](#) zu schützen, einschließlich Schlüssel für mehrere Regionen KMS. Sie können das beispielsweise so konfigurieren, AWS Encryption SDK dass Ihre Daten unter einem oder mehreren AWS KMS keys in Ihrem verschlüsselt werden. AWS-Konto Sie müssen jedoch den verwenden, AWS Encryption SDK um diese Daten zu entschlüsseln.

Der Chiffretext, den die Verschlüsselungs- oder Operationen zurückgeben, AWS Encryption SDK kann nicht AWS KMS [entschlüsselt](#) werden. [ReEncrypt Ebenso kann der AWS KMS Decrypt-Vorgang die zurückgesendete verschlüsselte Nachricht nicht entschlüsseln.](#) AWS Encryption SDK

Der AWS Encryption SDK unterstützt nur [symmetrische](#) Verschlüsselungsschlüssel. KMS Sie können keinen [asymmetrischen KMS Schlüssel](#) für die Verschlüsselung oder Anmeldung in verwenden. AWS Encryption SDK Der AWS Encryption SDK generiert seine eigenen ECDSA Signaturschlüssel für [Algorithmus-Suites](#), die Nachrichten signieren.

Hilfe bei der Entscheidung, welche Bibliothek oder welcher Dienst verwendet werden soll, finden Sie unter [So wählen Sie ein Verschlüsselungstool oder einen Verschlüsselungsdienst](#) aus in AWS Kryptografische Dienste und Tools.

Support und Wartung

Die AWS Encryption SDK verwendet dieselbe [Wartungsrichtlinie](#) wie die Tools, einschließlich ihrer Versionierung AWS SDK - und Lebenszyklusphasen. Als [bewährte Methode](#) empfehlen wir, dass Sie die neueste verfügbare Version von AWS Encryption SDK für Ihre Programmiersprache verwenden und ein Upgrade durchführen, sobald neue Versionen veröffentlicht werden. Wenn für eine Version wesentliche Änderungen erforderlich sind, z. B. das Upgrade von AWS Encryption SDK Versionen vor 1.7. x auf Versionen 2.0. x und später stellen wir [detaillierte Anweisungen](#) zur Verfügung, um Ihnen zu helfen.

Jede Programmiersprachenimplementierung von AWS Encryption SDK wird in einem separaten GitHub Open-Source-Repository entwickelt. Der Lebenszyklus und die Supportphase der einzelnen Versionen variieren wahrscheinlich je nach Repositorium. Beispielsweise AWS Encryption SDK könnte sich eine bestimmte Version von in einer Programmiersprache in der Phase der allgemeinen Verfügbarkeit (vollständiger Support) befinden, die end-of-support Phase jedoch in einer anderen Programmiersprache. Wir empfehlen, wann immer möglich eine vollständig unterstützte Version zu verwenden und Versionen zu vermeiden, die nicht mehr unterstützt werden.

Informationen zur Lebenszyklusphase von AWS Encryption SDK Versionen für Ihre Programmiersprache finden Sie in der `SUPPORT_POLICY.rst` Datei in den einzelnen AWS Encryption SDK Repositories.

- AWS-Verschlüsselungs-SDK for C — [SUPPORT_POLICY.rst](#)
- AWS Encryption SDK für .NET — [SUPPORT_POLICY.rst](#)
- AWS [Verschlüsselung CLI](#) — [SUPPORT_POLICY.rst](#)
- AWS-Verschlüsselungs-SDK for Java — [SUPPORT_POLICY.rst](#)
- AWS-Verschlüsselungs-SDK for JavaScript — [SUPPORT_POLICY.rst](#)
- AWS-Verschlüsselungs-SDK for Python — [SUPPORT_POLICY.rst](#)

Weitere Informationen finden Sie unter [Versionen der AWS Encryption SDK](#) und [AWS SDKs und in den Wartungsrichtlinien für Tools](#) im AWS SDKs Tools-Referenzhandbuch.

Mehr lernen

Weitere Informationen zur Verschlüsselung AWS Encryption SDK und zur clientseitigen Verschlüsselung finden Sie in diesen Quellen.

- Hilfe zu den hier verwendeten Begriffen und Konzepten finden Sie SDK unter [Konzepte in der AWS Encryption SDK](#)
- Richtlinien für bewährte Verfahren finden Sie unter [Bewährte Methoden für AWS Encryption SDK](#).
- Informationen darüber, wie das SDK funktioniert, finden Sie unter [Wie das SDK funktioniert](#).
- Beispiele, die zeigen, wie Optionen in der konfiguriert werden AWS Encryption SDK, finden Sie unter [Konfiguration der AWS Encryption SDK](#).
- Detaillierte technische Informationen finden Sie unter [Referenz](#).
- Die technischen Spezifikationen für finden Sie in der [AWS Encryption SDK Spezifikation](#) unter GitHub. AWS Encryption SDK
- Antworten auf Ihre Fragen zur Nutzung des AWS Encryption SDK Crypto Tools Discussion Forum finden Sie im [AWS Crypto Tools Discussion Forum](#).

Für Informationen über Implementierungen von AWS Encryption SDK in verschiedenen Programmiersprachen.

- C: Siehe [AWS-Verschlüsselungs-SDK for C](#) die AWS Encryption SDK [C-Dokumentation](#) und das [aws-encryption-sdk-c](#) Projektarchiv unter GitHub.
- C#. NET: Siehe [AWS Encryption SDK für .NET](#) und das [aws-encryption-sdk-net](#) Verzeichnis des `aws-encryption-sdk-dafny` Repositories ist aktiviert. GitHub
- Befehlszeilenschnittstelle: siehe [AWS Encryption SDK-Befehlszeilenschnittstelle](#), [Lesen Sie die Dokumente](#) zur AWS Verschlüsselung CLI und das [aws-encryption-sdk-cli](#) Repository auf GitHub.
- Java: Siehe [AWS-Verschlüsselungs-SDK for Java](#), das AWS Encryption SDK [Javadoc](#) und das [aws-encryption-sdk-java](#) Repository ist aktiviert. GitHub

JavaScript: Siehe [the section called "JavaScript"](#) und das [aws-encryption-sdk-javascript](#) Repository ist aktiviert. GitHub

- Python: Siehe [AWS-Verschlüsselungs-SDK for Python](#) die AWS Encryption SDK [Python-Dokumentation](#) und das [aws-encryption-sdk-python](#) Repository unter GitHub.

Senden von Feedback

Wir freuen uns über Ihr Feedback! Wenn Sie eine Frage oder einen Kommentar haben oder ein Problem melden möchten, verwenden Sie bitte die folgenden Ressourcen.

- Wenn Sie eine potenzielle Sicherheitslücke in der entdecken AWS Encryption SDK, [benachrichtigen Sie bitte den AWS Sicherheitsdienst](#). Erstellen Sie kein öffentliches GitHub Problem.
- Um Feedback zu geben AWS Encryption SDK, reichen Sie ein Problem im GitHub Repository für die von Ihnen verwendete Programmiersprache ein.
- Verwenden Sie die Feedback-Links auf dieser Seite, um Feedback zu dieser Dokumentation zu geben. Sie können auch ein Problem melden oder zu [aws-encryption-sdk-docs](#) dem Open-Source-Repository für diese Dokumentation beitragen. GitHub

Konzepte in der AWS Encryption SDK

In diesem Abschnitt werden die in der AWS Encryption SDK verwendeten Konzepte vorgestellt und ein Glossar sowie eine Referenz bereitgestellt. Es soll Ihnen helfen, zu verstehen, wie das AWS Encryption SDK funktioniert und mit welchen Begriffen wir es beschreiben.

Benötigen Sie Hilfe?

- Erfahren Sie, wie die [Umschlagverschlüsselung AWS Encryption SDK](#) verwendet, um Ihre Daten zu schützen.
- Erfahren Sie mehr über die Elemente der Umschlagverschlüsselung: die [Datenschlüssel](#), die Ihre Daten schützen, und die [Umhüllungsschlüssel](#), die Ihre Datenschlüssel schützen.
- Erfahren Sie mehr über die [Schlüsselanhänger](#) und [Hauptschlüsselanbieter](#), die bestimmen, welche Wrap-Schlüssel Sie verwenden.
- Erfahren Sie mehr über den [Verschlüsselungskontext](#), der Ihrem Verschlüsselungsprozess Integrität verleiht. Es ist optional, aber es ist eine bewährte Methode, die wir empfehlen.
- Erfahren Sie mehr über die [verschlüsselte Nachricht](#), die von den Verschlüsselungsmethoden zurückgegeben wird.
- Dann sind Sie bereit, das AWS Encryption SDK in Ihrer bevorzugten [Programmiersprache](#) zu verwenden.

Themen

- [Envelope-Verschlüsselung](#)
- [Datenschlüssel](#)
- [Schlüssel zum Umschließen](#)
- [Schlüsselanhänger und Hauptschlüsselanbieter](#)
- [Verschlüsselungskontext](#)
- [Verschlüsselte Nachricht](#)
- [Algorithmen-Paket](#)
- [Manager von kryptographischen Materialien](#)
- [Symmetrische und asymmetrische Verschlüsselung](#)
- [Wichtiges Engagement](#)
- [Verpflichtungspolitik](#)
- [Digitale Signaturen](#)

Envelope-Verschlüsselung

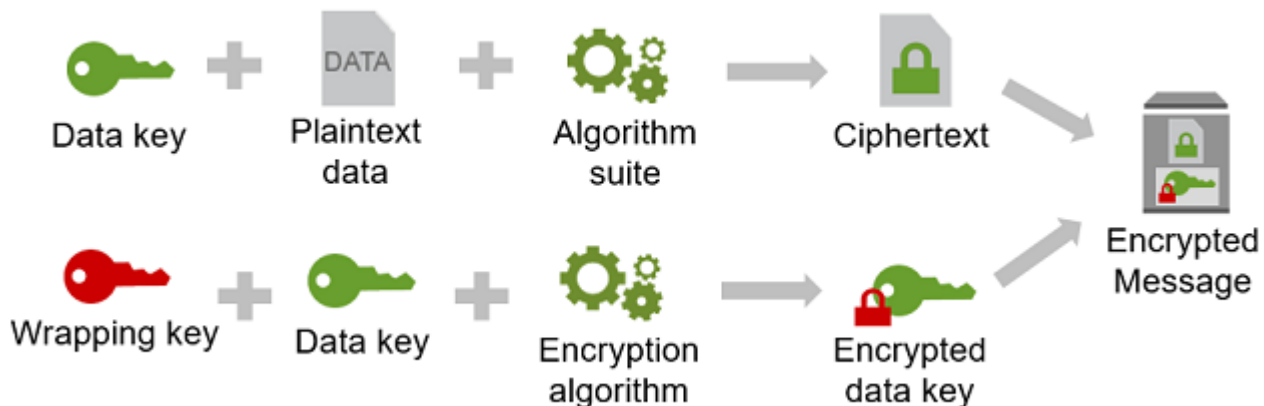
Die Sicherheit Ihrer verschlüsselten Daten hängt teilweise vom Schutz des Datenschlüssels ab, der sie entschlüsseln kann. Eine akzeptierte bewährte Methode zum Schutz des Datenschlüssels

ist seine Verschlüsselung. Dazu benötigen Sie einen weiteren Verschlüsselungsschlüssel, der als Schlüsselverschlüsselungsschlüssel oder [Wrapping-Schlüssel](#) bezeichnet wird. Die Praxis, Datenschlüssel mit einem Wrapping-Schlüssel zu verschlüsseln, wird als Envelope-Verschlüsselung bezeichnet.

Schutz von Datenschlüsseln

Die AWS Encryption SDK verschlüsselt jede Nachricht mit einem eindeutigen Datenschlüssel. Anschließend verschlüsselt es den Datenschlüssel unter dem von Ihnen angegebenen Wrapping-Schlüssel. Es speichert den verschlüsselten Datenschlüssel zusammen mit den verschlüsselten Daten in der verschlüsselten Nachricht, die es zurückgibt.

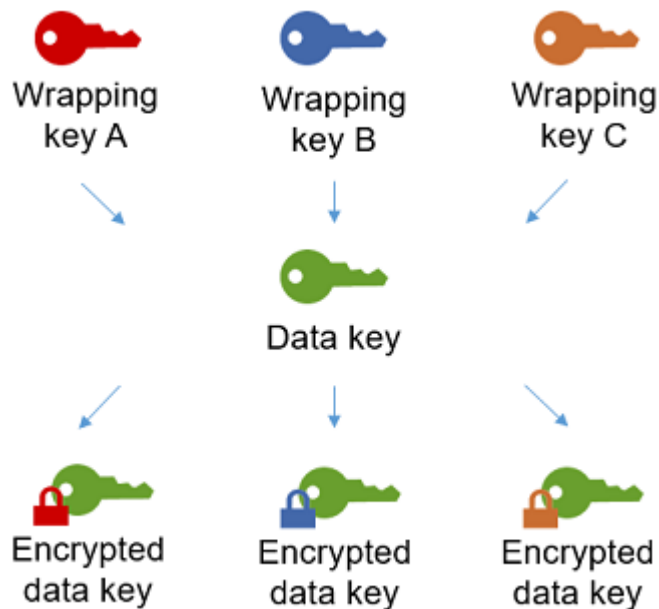
Um Ihren Wrapping-Schlüssel anzugeben, verwenden Sie einen [Schlüsselbund](#) oder einen [Master-Key-Anbieter](#).



Verschlüsseln derselben Daten unter mehreren Wrapping Keys

Sie können den Datenschlüssel unter mehreren Umschließungsschlüsseln verschlüsseln. Möglicherweise möchten Sie unterschiedliche Umschließungsschlüssel für verschiedene Benutzer oder Umschließungsschlüssel unterschiedlichen Typs oder an verschiedenen Speicherorten bereitstellen. Jeder der Umschließungsschlüssel verschlüsselt denselben Datenschlüssel. Der AWS Encryption SDK speichert alle verschlüsselten Datenschlüssel zusammen mit den verschlüsselten Daten in der verschlüsselten Nachricht.

Um die Daten zu entschlüsseln, müssen Sie einen Wrapping-Schlüssel angeben, der einen der verschlüsselten Datenschlüssel entschlüsseln kann.



Kombination der Stärken mehrerer Algorithmen

Um Ihre Daten zu verschlüsseln, AWS Encryption SDK verwendet der standardmäßig eine ausgeklügelte [AESAlgorithmus-Suite](#) mit GCM symmetrischer Verschlüsselung, einer Schlüsselableitungsfunktion (HKDF) und Signierung. Um den Datenschlüssel zu verschlüsseln, können Sie einen [symmetrischen oder asymmetrischen Verschlüsselungsalgorithmus](#) angeben, der Ihrem Wrapping-Schlüssel entspricht.

Im Allgemeinen sind symmetrische Schlüsselverschlüsselungsalgorithmen schneller und erzeugen kleinere Verschlüsselungstexte als eine asymmetrische Verschlüsselung oder eine Verschlüsselung mit öffentlichem Schlüssel. Algorithmen mit öffentlichem Schlüssel unterstützen jedoch eine inhärente Rollentrennung und eine einfachere Schlüsselverwaltung. Um beide Stärken zu kombinieren, können Sie Ihre Daten mit symmetrischer Schlüsselverschlüsselung verschlüsseln und anschließend den Datenschlüssel mit Public-Key-Verschlüsselung verschlüsseln.

Datenschlüssel

Ein Datenschlüssel ist ein Verschlüsselungsschlüssel, den das AWS Encryption SDK verwendet, um Ihre Daten zu verschlüsseln. Jeder Datenschlüssel ist ein Byte-Array, das die Anforderungen für kryptografische Schlüssel erfüllt. Sofern Sie nicht das [Zwischenspeichern von Datenschlüsseln](#) verwenden, AWS Encryption SDK verwendet der einen eindeutigen Datenschlüssel, um jede Nachricht zu verschlüsseln.

Sie müssen Datenschlüssel nicht spezifizieren, generieren, implementieren, erweitern, schützen oder verwenden. Das AWS Encryption SDK übernimmt diese Aufgabe für Sie, wenn Sie die Ver- und Entschlüsselungsoperationen aufrufen.

Um Ihre Datenschlüssel zu schützen, werden sie mit einem oder mehreren AWS Encryption SDK Schlüsselschlüsseln verschlüsselt, die als Wrapping Keys [oder Master Keys](#) bezeichnet werden. Nachdem der Ihre Klartext-Datenschlüssel AWS Encryption SDK verwendet hat, um Ihre Daten zu verschlüsseln, werden sie so schnell wie möglich aus dem Speicher entfernt. Die verschlüsselten Datenschlüssel werden dann mit den verschlüsselten Daten in der [verschlüsselten Nachricht](#) gespeichert, die die Verschlüsselungsoperationen zurückgibt. Details hierzu finden Sie unter [the section called "Wie das SDK funktioniert"](#).

Tip

In der AWS Encryption SDK unterscheiden wir Datenschlüssel von Datenverschlüsselungsschlüsseln. Mehrere der unterstützten [Algorithmen-Pakete](#), einschließlich des Standardpakets, verwenden eine [Schlüsselableitungsfunktion](#), die verhindert, dass der Datenschlüssel seine kryptografische Grenze erreicht. Die Schlüsselableitungsfunktion nimmt den Datenschlüssel als Eingabe entgegen und gibt einen Datenverschlüsselungsschlüssel zurück, mit dem die Daten letztlich verschlüsselt werden. Aus diesem Grund sagen wir oft, dass die Daten „unter“ einem Datenschlüssel verschlüsselt werden, statt „von“ dem Datenschlüssel.

Jeder verschlüsselte Datenschlüssel enthält Metadaten, einschließlich der Kennung des Wrapping-Schlüssels, mit dem er verschlüsselt wurde. Diese Metadaten erleichtern es den, gültige Wrapping-Schlüssel beim Entschlüsseln AWS Encryption SDK zu identifizieren.

Schlüssel zum Umschließen

Ein Wrapping Key ist ein Schlüssel zur AWS Encryption SDK Verschlüsselung des [Datenschlüssels, der Ihre Daten](#) verschlüsselt. Jeder Klartext-Datenschlüssel kann mit einem oder mehreren Umschließungsschlüsseln verschlüsselt werden. Sie legen fest, welche Umschließungsschlüssel zum Schutz Ihrer Daten verwendet werden, wenn Sie einen [Schlüsselbund](#) oder einen [Hauptschlüsselanbieter](#) konfigurieren.

Note

Wrapping Key bezieht sich auf die Schlüssel in einem Schlüsselbund oder einem Hauptschlüsselanbieter. Der Hauptschlüssel ist normalerweise der `MasterKey` Klasse zugeordnet, die Sie instanziiieren, wenn Sie einen Hauptschlüsselanbieter verwenden.

Der AWS Encryption SDK unterstützt mehrere häufig verwendete Schlüssel zum Umschließen von Schlüsseln, wie z. B. AWS Key Management Service (AWS KMS) symmetrische Schlüssel [AWS KMS keys](#) (einschließlich Schlüssel mit [mehreren Regionen](#)), [KMS unformatierte Schlüssel](#) GCM (Advanced Encryption Standard/Galois Counter Mode) und AES Rohschlüssel. RSA Sie können auch Ihre eigenen Wrapping Keys erweitern oder implementieren.

Wenn Sie die Envelope-Verschlüsselung verwenden, müssen Sie Ihre Wrapping-Schlüssel vor unbefugtem Zugriff schützen. Sie können dies auf eine der folgenden Arten tun:

- Verwenden Sie einen Webservice, der für diesen Zweck vorgesehen ist, wie z. B. [AWS Key Management Service \(AWS KMS\)](#).
- Verwenden Sie ein [Hardware-Sicherheitsmodul \(HSM\)](#), wie es von angeboten wird [AWS CloudHSM](#).
- Verwenden Sie andere wichtige Verwaltungstools und -dienste.

Wenn Sie kein Schlüsselverwaltungssystem haben, empfehlen wir AWS KMS. Das AWS Encryption SDK lässt sich integrieren AWS KMS , um Ihnen zu helfen, Ihre Verpackungsschlüssel zu schützen und zu verwenden. Für das AWS Encryption SDK ist jedoch kein AWS Service erforderlich AWS .

Schlüsselanbieter und Hauptschlüsselanbieter

Um die Wrapping-Schlüssel anzugeben, die Sie für die Verschlüsselung und Entschlüsselung verwenden, verwenden Sie einen Schlüsselbund (C, C#, .NET, und JavaScript) oder ein Hauptschlüsselanbieter (Java, Python, CLI). Sie können die von ihm bereitgestellten Schlüsselringe und Hauptschlüsselanbieter verwenden oder Ihre eigenen Implementierungen entwerfen. AWS Encryption SDK Der AWS Encryption SDK stellt Schlüsselringe und Hauptschlüsselanbieter bereit, die je nach Spracheinschränkungen miteinander kompatibel sind. Details hierzu finden Sie unter [Schlüsselbund-Kompatibilität](#).

Ein Schlüsselbund generiert, verschlüsselt und entschlüsselt Datenschlüssel. Wenn Sie einen Schlüsselbund definieren, können Sie die [Umschließungsschlüssel angeben, mit denen Ihre](#)

[Datenschlüssel](#) verschlüsselt werden. Die meisten Schlüsselbunde spezifizieren mindestens einen Umschließungsschlüssel oder einen Dienst, der Schlüssel zum Umschließen bereitstellt und schützt. Sie können auch einen Schlüsselbund ohne Umschließungsschlüssel oder einen komplexeren Schlüsselbund mit zusätzlichen Konfigurationsoptionen definieren. Hilfe bei der Auswahl und Verwendung der von definierten Schlüsselbunden AWS Encryption SDK finden Sie unter [Verwenden von Schlüsselbunden](#). Schlüsselringe werden in C, C#, JavaScript, und Version 3. x der AWS-Verschlüsselungs-SDK for Java.

Ein Hauptschlüsselanbieter ist eine Alternative zu einem Schlüsselbund. Der Hauptschlüsselanbieter gibt die von Ihnen angegebenen Wrapping-Schlüssel (oder Hauptschlüssel) zurück. Jeder Masterschlüssel ist einem Masterschlüssel-Anbieter zugeordnet, aber ein Masterschlüssel-Anbieter stellt in der Regel mehrere Masterschlüssel bereit. Master-Key-Anbieter werden in Java, Python und der AWS Verschlüsselung unterstütztCLI.

Sie müssen einen Schlüsselbund (oder einen Hauptschlüsselanbieter) für die Verschlüsselung angeben. Sie können denselben Schlüsselbund (oder Hauptschlüsselanbieter) oder einen anderen für die Entschlüsselung angeben. Bei der Verschlüsselung AWS Encryption SDK verwendet der alle von Ihnen angegebenen Umschließungsschlüssel, um den Datenschlüssel zu verschlüsseln. Beim Entschlüsseln AWS Encryption SDK verwendet der nur die von Ihnen angegebenen Umschließungsschlüssel, um einen verschlüsselten Datenschlüssel zu entschlüsseln. [Die Angabe von Umschließungsschlüsseln für die Entschlüsselung ist optional, hat sich aber bewährt. AWS Encryption SDK](#)

Einzelheiten zum Angeben von Schlüsseln zum Umschließen von Schlüsseln finden Sie unter [Auswahl von Schlüsseln zum Umbrechen](#).

Verschlüsselungskontext

Um die Sicherheit Ihrer kryptographischen Operationen zu verbessern, fügen Sie einen [Verschlüsselungskontext](#) in alle Anfragen zur Verschlüsselung von Daten ein. Die Verwendung eines Verschlüsselungskontexts ist optional, aber wir empfehlen dies als bewährte Methode für die Kryptografie.

Ein Verschlüsselungskontext ist eine Gruppe von Name-Wert-Paaren mit willkürlichen, nicht geheimen, zusätzlich authentifizierten Daten. Der Verschlüsselungskontext kann beliebige Daten enthalten, aber er besteht in der Regel aus Daten, die für die Protokollierung und Verfolgung nützlich sind, wie z. B. Daten über den Dateityp, den Zweck oder das Eigentum. Wenn Sie Daten verschlüsseln, wird der Verschlüsselungskontext kryptografisch an die verschlüsselten Daten

gebunden, sodass derselbe Verschlüsselungskontext zur Entschlüsselung der Daten benötigt wird. Das AWS Encryption SDK enthält den Verschlüsselungskontext als Klartext im Header der [verschlüsselten Nachricht](#), die es zurückgibt.

Der Verschlüsselungskontext, den der AWS Encryption SDK verwendet, besteht aus dem Verschlüsselungskontext, den Sie angeben, und einem öffentlichen key pair, das der [Cryptographic Materials Manager](#) (CMM) hinzufügt. Insbesondere wenn Sie einen [Verschlüsselungsalgorithmus mit Signatur](#) verwenden, CMM fügt der dem Verschlüsselungskontext ein Name-Wert-Paar hinzu, das aus einem reservierten Namen und einem Wert besteht `aws-crypto-public-key`, der den öffentlichen Bestätigungsschlüssel darstellt. Der `aws-crypto-public-key` Name im Verschlüsselungskontext ist für den reserviert AWS Encryption SDK und kann nicht als Name in einem anderen Paar im Verschlüsselungskontext verwendet werden. Einzelheiten finden Sie [AAD](#) in der Referenz zum Nachrichtenformat.

Der folgende Beispielvechlüsselungskontext besteht aus zwei Verschlüsselungskontextpaaren, die in der Anfrage angegeben sind, und dem öffentlichen key pair, das der CMM hinzufügt.

```
"Purpose"="Test", "Department"="IT", aws-crypto-public-key=<public key>
```

Zum Entschlüsseln der Daten übergeben Sie die verschlüsselte Nachricht. Da der den Verschlüsselungskontext aus dem verschlüsselten Nachrichtenheader extrahieren AWS Encryption SDK kann, müssen Sie den Verschlüsselungskontext nicht separat angeben. Der Verschlüsselungskontext kann jedoch helfen zu bestätigen, dass Sie die richtige verschlüsselte Nachricht entschlüsselt wurde.

- Wenn Sie in der [AWS Encryption SDK Befehlszeilenschnittstelle](#) (CLI) einen Verschlüsselungskontext in einem Entschlüsselungsbefehl angeben, CLI wird überprüft, ob die Werte im Verschlüsselungskontext der verschlüsselten Nachricht vorhanden sind, bevor die Klartextdaten zurückgegeben werden.
- In anderen Programmiersprachenimplementierungen umfasst die Entschlüsselungsantwort den Verschlüsselungskontext und die Klartextdaten. Die Entschlüsselungsfunktion in Ihrer Anwendung sollte immer überprüfen, ob der Verschlüsselungskontext in der decrypt-Antwort den Verschlüsselungskontext in der Verschlüsselungsanfrage (oder einer Teilmenge) enthält, bevor sie die Klartextdaten zurückgibt.

Note

Mit [Version 4. x der AWS Encryption SDK Form. NET](#) und [Version 3. x von AWS-Verschlüsselungs-SDK for Java](#), Sie können für alle Verschlüsselungsanfragen mit dem erforderlichen Verschlüsselungskontext einen Verschlüsselungskontext CMM angeben.

Wenn Sie einen Verschlüsselungskontext wählen, denken Sie daran, dass dies kein Geheimnis ist. Der Verschlüsselungskontext wird im Klartext im Header der [verschlüsselten Nachricht](#) angezeigt, die zurückgegeben wird AWS Encryption SDK . Wenn Sie verwenden AWS Key Management Service, wird der Verschlüsselungskontext möglicherweise auch im Klartext in Prüfaufzeichnungen und Protokollen angezeigt, z. B. AWS CloudTrail

Beispiele für das Einreichen und Überprüfen eines Verschlüsselungskontextes in Ihrem Code finden Sie in den Beispielen für Ihre bevorzugte [Programmiersprache](#).

Verschlüsselte Nachricht

Wenn Sie Daten mit dem verschlüsseln AWS Encryption SDK, wird eine verschlüsselte Nachricht zurückgegeben.

Eine verschlüsselte Nachricht ist eine übertragbare, [formatierte Datenstruktur](#), die die verschlüsselten Daten zusammen mit verschlüsselten Kopien der Datenschlüssel, der Algorithmus-ID und optional einem [Verschlüsselungskontext](#) und einer [digitalen](#) Signatur enthält. Verschlüsselungsoperationen im AWS Encryption SDK geben eine verschlüsselte Nachricht zurück, und Entschlüsselungsoperationen nehmen eine verschlüsselte Nachricht als Eingabe entgegen.

Die Kombination der verschlüsselten Daten und ihrer verschlüsselten Datenschlüssel rationalisiert den Entschlüsselungsvorgang und befreit Sie von der Notwendigkeit, verschlüsselte Datenschlüssel unabhängig von den verschlüsselten Daten zu speichern und zu verwalten.

Technische Informationen über die verschlüsselte Nachricht finden Sie unter [Verschlüsseltes Nachrichtenformat](#).

Algorithmen-Paket

Der AWS Encryption SDK verwendet eine Algorithmus-Suite, um die Daten in der [verschlüsselten Nachricht](#) zu verschlüsseln und zu signieren, die bei den Verschlüsselungs- und Entschlüsselungsvorgängen zurückgegeben werden. Das AWS Encryption SDK unterstützt mehrere

[Algorithmen-Pakete](#). Alle unterstützten Suiten verwenden Advanced Encryption Standard (AES) als primären Algorithmus und kombinieren ihn mit anderen Algorithmen und Werten.

Die AWS Encryption SDK legt eine empfohlene Algorithmus-Suite als Standard für alle Verschlüsselungsvorgänge fest. Die Vorgabe kann sich ändern, wenn sich Standards und bewährte Methoden verbessern. Sie können bei Anfragen zur Verschlüsselung von Daten oder bei der Erstellung eines [Cryptographic Materials Managers \(CMM\)](#) eine alternative Algorithmus-Suite angeben. Sofern für Ihre Situation jedoch keine Alternative erforderlich ist, empfiehlt es sich, die Standardeinstellung zu verwenden. Die aktuelle Standardeinstellung ist AES - GCM mit einer HMAC basierten extract-and-expand [Schlüsselableitungsfunktion \(HKDF\)](#), einer [Schlüsselzusage](#), einer [Signatur mit einem Algorithmus für digitale Signaturen mit elliptischen Kurven \(ECDSA\)](#) und einem 256-Bit-Verschlüsselungsschlüssel.

Wenn Ihre Anwendung eine hohe Leistung erfordert und die Benutzer, die Daten verschlüsseln, und die Benutzer, die Daten entschlüsseln, gleichermaßen vertrauenswürdig sind, sollten Sie erwägen, eine Algorithmus-Suite ohne digitale Signatur anzugeben. Wir empfehlen jedoch dringend eine Algorithmensuite, die Schlüsselzusage und eine Funktion zur Schlüsselableitung umfasst. Algorithmen-Suiten ohne diese Funktionen werden nur aus Gründen der Abwärtskompatibilität unterstützt.

Manager von kryptographischen Materialien

Der Cryptographic Materials Manager (CMM) stellt die kryptografischen Materialien zusammen, die zum Verschlüsseln und Entschlüsseln von Daten verwendet werden. Die kryptografischen Materialien umfassen Klartext- und verschlüsselte Datenschlüssel und einen optionalen Nachrichten-Signaturschlüssel. Sie interagieren nie direkt mit dem CMM. Die Ver- und Entschlüsselungsmethoden übernehmen das für Sie.

Sie können die Standardeinstellung CMM oder das vom bereitgestellte [Caching CMM](#) verwenden oder ein benutzerdefiniertes CMM Caching schreiben. AWS Encryption SDK Und Sie können eine angeben CMM, aber das ist nicht erforderlich. Wenn Sie einen Schlüsselbund oder einen Hauptschlüsselanbieter angeben, AWS Encryption SDK erstellt dieser einen Standard CMM für Sie. Die Standardeinstellung CMM bezieht die Ver- oder Entschlüsselungsmaterialien von dem Schlüsselbund oder Hauptschlüsselanbieter, den Sie angeben. Dabei könnte es sich um einen Aufruf eines kryptographischen Dienstes handeln, z. B. [AWS Key Management Service](#) (AWS KMS).

Da der als Bindeglied zwischen dem AWS Encryption SDK und einem Schlüsselbund (oder Hauptschlüsselanbieter) CMM fungiert, ist er ein idealer Ort für Anpassungen und Erweiterungen, z. B. zur Unterstützung von Richtlinienumsetzung und Zwischenspeicherung. [Das AWS Encryption](#)

[SDK bietet eine Zwischenspeicherung zur Unterstützung der Zwischenspeicherung CMM von Datenschlüsseln.](#)

Symmetrische und asymmetrische Verschlüsselung

Bei der symmetrischen Verschlüsselung wird derselbe Schlüssel zum Verschlüsseln und Entschlüsseln von Daten verwendet.

Asymmetrische Verschlüsselung verwendet ein mathematisch verwandtes Datenschlüsselpaar. Ein Schlüssel des Paares verschlüsselt die Daten; nur der andere Schlüssel im Paar kann die Daten entschlüsseln. Einzelheiten finden Sie unter [Kryptografische Algorithmen](#) im AWS Cryptographic Services and Tools Guide.

Der AWS Encryption SDK verwendet [Envelope-Verschlüsselung](#). Es verschlüsselt Ihre Daten mit einem symmetrischen Datenschlüssel. Es verschlüsselt den symmetrischen Datenschlüssel mit einem oder mehreren symmetrischen oder asymmetrischen Wrapping-Schlüsseln. Es gibt eine [verschlüsselte Nachricht](#) zurück, die die verschlüsselten Daten und mindestens eine verschlüsselte Kopie des Datenschlüssels enthält.

Verschlüsseln Ihrer Daten (symmetrische Verschlüsselung)

Um Ihre Daten zu verschlüsseln, AWS Encryption SDK verwendet der einen symmetrischen [Datenschlüssel](#) und eine [Algorithmussuite, die einen symmetrischen Verschlüsselungsalgorithmus](#) enthält. Um die Daten zu entschlüsseln, AWS Encryption SDK verwendet der denselben Datenschlüssel und dieselbe Algorithmussuite.

Verschlüsselung Ihres Datenschlüssels (symmetrische oder asymmetrische Verschlüsselung)

Der [Schlüsselbund](#) oder [Hauptschlüsselanbieter, den Sie für einen Verschlüsselungs- und Entschlüsselungsvorgang angeben, bestimmt, wie der symmetrische Datenschlüssel](#) ver- und entschlüsselt wird. Sie können einen Schlüsselbund- oder Hauptschlüsselanbieter wählen, der symmetrische Verschlüsselung verwendet, z. B. einen AWS KMS Schlüsselbund, oder einen Anbieter, der asymmetrische Verschlüsselung verwendet, z. B. einen Rohschlüsselbund oder RSA JceMasterKey

Wichtiges Engagement

Das AWS Encryption SDK unterstützt Key Commitment (manchmal auch als Robustheit bezeichnet), eine Sicherheitseigenschaft, die garantiert, dass jeder Chiffretext nur in einen einzigen

Klartext entschlüsselt werden kann. Zu diesem Zweck garantiert Key Commitment, dass nur der Datenschlüssel, mit dem Ihre Nachricht verschlüsselt wurde, zur Entschlüsselung verwendet wird. [Das Verschlüsseln und Entschlüsseln mit Key Commitment ist eine AWS Encryption SDK bewährte Methode.](#)

Die meisten modernen symmetrischen Chiffren (einschließlich AES) verschlüsseln Klartext unter einem einzigen geheimen Schlüssel, z. B. dem [eindeutigen Datenschlüssel](#), mit dem jede Klartextnachricht verschlüsselt wird AWS Encryption SDK. Beim Entschlüsseln dieser Daten mit demselben Datenschlüssel wird ein Klartext zurückgegeben, der mit dem Original identisch ist. Die Entschlüsselung mit einem anderen Schlüssel schlägt normalerweise fehl. Es ist jedoch möglich, einen Chiffretext unter zwei verschiedenen Schlüsseln zu entschlüsseln. In seltenen Fällen ist es möglich, einen Schlüssel zu finden, der einige Byte Chiffretext in einen anderen, aber dennoch verständlichen Klartext entschlüsseln kann.

Der verschlüsselt AWS Encryption SDK immer jede Klartext-Nachricht unter einem eindeutigen Datenschlüssel. Es kann diesen Datenschlüssel unter mehreren Umschließungsschlüsseln (oder Hauptschlüsseln) verschlüsseln, aber die Umschließungsschlüssel verschlüsseln immer denselben Datenschlüssel. Dennoch kann eine ausgeklügelte, manuell erstellte [verschlüsselte Nachricht](#) tatsächlich unterschiedliche Datenschlüssel enthalten, von denen jeder mit einem anderen Umschließungsschlüssel verschlüsselt ist. Entschlüsselt beispielsweise ein Benutzer die verschlüsselte Nachricht, wird 0x0 (falsch) zurückgegeben, während ein anderer Benutzer, der dieselbe verschlüsselte Nachricht entschlüsselt, 0x1 (wahr) erhält.

Um dieses Szenario zu verhindern, AWS Encryption SDK unterstützt der Key Commitment beim Verschlüsseln und Entschlüsseln. Beim AWS Encryption SDK Verschlüsseln einer Nachricht mit Schlüsselzusage wird der eindeutige Datenschlüssel, der den Chiffretext erzeugt hat, kryptografisch an die Schlüsselverbindungszeichenfolge gebunden, eine nicht geheime Datenschlüssel-ID. Anschließend speichert es die Schlüsselbestätigungszeichenfolge in den Metadaten der verschlüsselten Nachricht. Beim Entschlüsseln einer Nachricht mit Schlüsselzusage wird AWS Encryption SDK überprüft, ob der Datenschlüssel der einzige Schlüssel für diese verschlüsselte Nachricht ist. Wenn die Überprüfung des Datenschlüssels fehlschlägt, schlägt der Entschlüsselungsvorgang fehl.

Die Support für Key Commitment wurde in Version 1.7 eingeführt. x, das Nachrichten mit Key Commitment entschlüsseln kann, aber nicht mit Key Commitment verschlüsselt. Sie können diese Version verwenden, um die Fähigkeit zur Entschlüsselung von Chiffretext mit Schlüsselbindung vollständig auszuschöpfen. Version 2.0. x beinhaltet volle Unterstützung für Key Commitment. Standardmäßig verschlüsselt und entschlüsselt es nur mit Key Commitment. Dies ist eine ideale

Konfiguration für Anwendungen, die keinen Chiffretext entschlüsseln müssen, der mit früheren Versionen von verschlüsselt wurde. AWS Encryption SDK

Obwohl das Verschlüsseln und Entschlüsseln mit Schlüsselbindung eine bewährte Methode ist, überlassen wir Ihnen die Entscheidung, wann es verwendet wird, und Sie können das Tempo, in dem Sie es einführen, anpassen. Ab Version 1.7. x AWS Encryption SDK unterstützt eine [Commitment-Richtlinie](#), die die [standardmäßige Algorithmensuite](#) festlegt und die Anzahl der Algorithmen-Suiten einschränkt, die verwendet werden können. Diese Richtlinie legt fest, ob Ihre Daten mit Schlüsselbindung ver- oder entschlüsselt werden.

Key Commitment führt zu einer [etwas größeren \(+ 30 Byte\) verschlüsselten Nachricht](#) und die Verarbeitung nimmt mehr Zeit in Anspruch. Wenn Ihre Anwendung sehr empfindlich auf Größe oder Leistung reagiert, können Sie sich dafür entscheiden, die Schlüsselzuweisung zu deaktivieren. Aber tun Sie das nur, wenn Sie müssen.

Weitere Informationen zur Migration auf Versionen 1.7. x und 2.0. x, einschließlich ihrer wichtigsten Commitment-Funktionen, siehe [Migrieren Sie Ihre AWS Encryption SDK](#). Technische Informationen zu den wichtigsten Verpflichtungen finden Sie unter [the section called “Algorithmen – Referenz”](#) und [the section called “Nachrichtenformat – Referenz”](#).

Verpflichtungspolitik





Eine Commitment-Richtlinie ist eine Konfigurationseinstellung, die bestimmt, ob Ihre Anwendung mit [Key Commitment](#) ver- oder entschlüsselt wird. [Das Verschlüsseln und Entschlüsseln mit Key Commitment ist eine bewährte Methode.](#) AWS Encryption SDK

Die Verpflichtungspolitik hat drei Werte.

Note

Möglicherweise müssen Sie horizontal oder vertikal scrollen, um die gesamte Tabelle zu sehen.

Verbindungspolitische Werte

Wert	Verschlüsselt mit Schlüsselzusage	Verschlüsselt ohne Schlüsselbindung	Entschlüsselt mit Schlüsselbindung	Entschlüsselt ohne Schlüsselbindung
ForbidEncryptAllowDecrypt				
RequireEncryptAllowDecrypt				
RequireEncryptRequireDecrypt				

Die Richtlinieneinstellung für Verpflichtungen wurde in AWS Encryption SDK Version 1.7 eingeführt. x. Es ist in allen unterstützten [Programmiersprachen](#) gültig.

- `ForbidEncryptAllowDecrypt` entschlüsselt mit oder ohne Key Commitment, verschlüsselt aber nicht mit Key Commitment. Dieser Wert wurde in Version 1.7 eingeführt. x wurde entwickelt, um alle Hosts, auf denen Ihre Anwendung ausgeführt wird, darauf vorzubereiten, mit Key Commitment zu entschlüsseln, bevor sie jemals auf einen mit Key Commitment verschlüsselten Chiffretext stoßen.
- `RequireEncryptAllowDecrypt` verschlüsselt immer mit Key Commitment. Es kann mit oder ohne Schlüsselbindung entschlüsselt werden. Dieser Wert wurde in Version 2.0 eingeführt. x, ermöglicht es Ihnen, mit der Verschlüsselung mit Schlüsselzusage zu beginnen, ältere Chiffretexte aber trotzdem ohne Schlüsselbindung zu entschlüsseln.
- `RequireEncryptRequireDecrypt` verschlüsselt und entschlüsselt nur mit Schlüsselbindung. Dieser Wert ist der Standardwert für Version 2.0. x. Verwenden Sie diesen Wert, wenn Sie sicher sind, dass alle Ihre Chiffretexte mit Schlüsselbindung verschlüsselt sind.

Die Richtlinieneinstellung „Commitment“ bestimmt, welche Algorithmus-Suites Sie verwenden können. Ab Version 1.7. x, der AWS Encryption SDK unterstützt [Algorithmus-Suites](#) für Key

Commitment; mit und ohne Signierung. Wenn Sie eine Algorithmus-Suite angeben, die mit Ihrer Commitment-Richtlinie in Konflikt steht, wird ein Fehler AWS Encryption SDK zurückgegeben.

Hilfe bei der Festlegung Ihrer Verpflichtungsrichtlinie finden Sie unter [Festlegung Ihrer Verpflichtungspolitik](#).

Digitale Signaturen

Um die Integrität einer digitalen Nachricht bei der Übertragung zwischen Systemen sicherzustellen, können Sie eine digitale Signatur auf die Nachricht anwenden. Digitale Signaturen sind immer asymmetrisch. Sie verwenden Ihren privaten Schlüssel, um die Signatur zu erstellen, und hängen sie an die ursprüngliche Nachricht an. Ihr Empfänger verwendet einen öffentlichen Schlüssel, um sicherzustellen, dass die Nachricht seit dem Signieren nicht geändert wurde.

Der AWS Encryption SDK verschlüsselt Ihre Daten mit einem authentifizierten Verschlüsselungsalgorithmus, AES -GCM, und der Entschlüsselungsprozess überprüft die Integrität und Authentizität einer verschlüsselten Nachricht ohne Verwendung einer digitalen Signatur. Da AES - jedoch symmetrische Schlüssel GCM verwendet, könnte jeder, der den zur Entschlüsselung des Chiffretextes verwendeten Datenschlüssel entschlüsseln kann, auch manuell einen neuen verschlüsselten Chiffretext erstellen, was zu potenziellen Sicherheitsbedenken führen kann. Wenn Sie beispielsweise einen AWS KMS Schlüssel als Umschließungsschlüssel verwenden, bedeutet dies, dass ein Benutzer mit KMS Decrypt-Berechtigungen verschlüsselte Chiffretexte erstellen kann, ohne Encrypt aufrufen zu müssen. KMS

Um dieses Problem zu vermeiden, AWS Encryption SDK unterstützt das Hinzufügen einer Signatur mit dem Elliptic Curve Digital Signature Algorithm (ECDSA) am Ende verschlüsselter Nachrichten. Wenn eine Signaturalgorithmus-Suite verwendet wird, AWS Encryption SDK generiert sie für jede verschlüsselte Nachricht ein temporäres Paar aus privatem Schlüssel und öffentlichem Schlüssel. Der AWS Encryption SDK speichert den öffentlichen Schlüssel im Verschlüsselungskontext des Datenschlüssels und verwirft den privaten Schlüssel, sodass niemand eine weitere Signatur erstellen kann, die mit dem öffentlichen Schlüssel verifiziert wird. Da der Algorithmus den öffentlichen Schlüssel als zusätzliche authentifizierte Daten im Nachrichtenkopf an den verschlüsselten Datenschlüssel bindet, kann ein Benutzer, der nur Nachrichten entschlüsseln kann, den öffentlichen Schlüssel nicht ändern.

Die Signaturüberprüfung führt zu erheblichen Leistungseinbußen bei der Entschlüsselung. Wenn die Benutzer, die Daten verschlüsseln, und die Benutzer, die Daten entschlüsseln, gleichermaßen vertrauenswürdig sind, sollten Sie erwägen, eine Algorithmusuite zu verwenden, die das Signieren nicht beinhaltet.

Funktionsweise des AWS Encryption SDK

[In den Workflows in diesem Abschnitt wird erklärt, wie Daten AWS Encryption SDK verschlüsselt und verschlüsselte Nachrichten entschlüsselt werden.](#) In diesen Workflows wird der grundlegende Prozess unter Verwendung der Standardfunktionen beschrieben. Einzelheiten zur Definition und Verwendung benutzerdefinierter Komponenten finden Sie im GitHub Repository für jede unterstützte [Sprachimplementierung](#).

Der AWS Encryption SDK verwendet Umschlagverschlüsselung, um Ihre Daten zu schützen. Jede Nachricht wird unter einem eindeutigen Datenschlüssel verschlüsselt. Anschließend wird der Datenschlüssel mit den von Ihnen angegebenen Wrapping-Schlüsseln verschlüsselt. Um die verschlüsselte Nachricht zu entschlüsseln, AWS Encryption SDK verwendet der die von Ihnen angegebenen Umschließungsschlüssel, um mindestens einen verschlüsselten Datenschlüssel zu entschlüsseln. Dann kann es den Chiffretext entschlüsseln und eine Klartext-Nachricht zurückgeben.

Benötigen Sie Hilfe mit der Terminologie, die wir in der verwenden? AWS Encryption SDK Siehe [the section called "Konzepte"](#).

Wie AWS Encryption SDK verschlüsselt der Daten

Das AWS Encryption SDK stellt Methoden zur Verschlüsselung von Zeichenketten, Byte-Arrays und Byte-Streams bereit. Codebeispiele finden Sie im Thema Beispiele in den einzelnen [Programmiersprachen](#) Abschnitten.

1. Erstellen Sie einen [Schlüsselbund](#) (oder einen [Hauptschlüsselanbieter](#)), der die Umschließungsschlüssel angibt, die Ihre Daten schützen.
2. Übergeben Sie den Schlüsselbund und die Klartextdaten an eine Verschlüsselungsmethode. [Wir empfehlen, dass Sie einen optionalen, nicht geheimen Verschlüsselungskontext angeben.](#)
3. Die Verschlüsselungsmethode fragt den Schlüsselbund nach Verschlüsselungsmaterial. Der Schlüsselbund gibt eindeutige Datenverschlüsselungsschlüssel für die Nachricht zurück: einen Klartext-Datenschlüssel und eine Kopie dieses Datenschlüssels, der mit jedem der angegebenen Umschließungsschlüssel verschlüsselt wurde.
4. Die Verschlüsselungsmethode verwendet den Klartext-Datenschlüssel, um die Daten zu verschlüsseln, und verwirft dann den Klartext-Datenschlüssel. Wenn Sie einen Verschlüsselungskontext angeben (eine AWS Encryption SDK [bewährte](#) Methode), bindet die Verschlüsselungsmethode den Verschlüsselungskontext kryptografisch an die verschlüsselten Daten.

5. Die Verschlüsselungsmethode gibt eine [verschlüsselte Nachricht](#) zurück, die die verschlüsselten Daten, die verschlüsselten Datenschlüssel und andere Metadaten, einschließlich des Verschlüsselungskontextes, falls Sie einen verwendet haben, enthält.

Wie AWS Encryption SDK entschlüsselt der eine verschlüsselte Nachricht

Das AWS Encryption SDK bietet Methoden, mit denen die [verschlüsselte Nachricht entschlüsselt und Klartext](#) zurückgegeben wird. Codebeispiele finden Sie im Thema Beispiele in den einzelnen [Programmiersprachen](#) Abschnitten.

Der [Schlüsselbund](#) (oder der [Hauptschlüsselanbieter](#)), der die verschlüsselte Nachricht entschlüsselt, muss mit dem Schlüsselbund kompatibel sein, der zum Verschlüsseln der Nachricht verwendet wurde. Einer seiner Wrapping-Schlüssel muss in der Lage sein, einen verschlüsselten Datenschlüssel in der verschlüsselten Nachricht zu entschlüsseln. Hinweise zur Kompatibilität mit Schlüsselringen und Hauptschlüsselanbietern finden Sie unter [the section called "Schlüsselbund-Kompatibilität"](#)

1. Erstellen Sie einen Schlüsselbund oder einen Hauptschlüsselanbieter mit Wrap-Schlüsseln, der Ihre Daten entschlüsseln kann. Sie können denselben Schlüsselbund verwenden, den Sie für die Verschlüsselungsmethode bereitgestellt haben, oder einen anderen.
2. Übergeben Sie die [verschlüsselte Nachricht](#) und den Schlüsselbund an eine Entschlüsselungsmethode.
3. Bei der Entschlüsselungsmethode wird der Schlüsselbund oder der Hauptschlüsselanbieter aufgefordert, einen der verschlüsselten Datenschlüssel in der verschlüsselten Nachricht zu entschlüsseln. Es übergibt Informationen aus der verschlüsselten Nachricht, einschließlich der verschlüsselten Datenschlüssel.
4. Der Schlüsselbund verwendet seinen Umhüllungsschlüssel zum Entschlüsseln einer der verschlüsselten Datenschlüssel. Wenn sie erfolgreich ist, enthält die Antwort den Klartext-Datenschlüssel. Wenn keiner der vom Schlüsselbund oder Hauptschlüsselanbieter angegebenen Umschließungsschlüssel einen verschlüsselten Datenschlüssel entschlüsseln kann, schlägt der Entschlüsselungsaufruf fehl.
5. Die Entschlüsselungsmethode verwendet den Klartext-Datenschlüssel, um die Daten zu entschlüsseln, verwirft den Klartext-Datenschlüssel und gibt die Klartextdaten zurück.

Unterstützte Algorithmen-Pakete im AWS Encryption SDK

Ein Algorithmen-Paket ist eine Sammlung von kryptografischen Algorithmen und zugehörigen Werten. Kryptografische Systeme verwenden die Algorithmen-Implementierung, um die Verschlüsselungstext-Nachricht zu generieren.

Das AWS Encryption SDK-Algorithmen-Paket verwendet den Advanced Encryption Standard (AES)-Algorithmus im Galois/Counter Mode (GCM), als AES-GCM bezeichnet, zur Verschlüsselung von Rohdaten. Die AWS Encryption SDK unterstützt 256-, 192- und 128-Bit-Verschlüsselungsschlüssel. Die Länge des Initialisierungsvektors (IV) beträgt immer 12 Bytes. Die Länge des Authentifizierungstags beträgt immer 16 Bytes.

Standardmäßig ist der AWS Encryption SDK verwendet eine Algorithmen-Suite mit AES-GCM mit einem HMAC-basierten extract-and-expand Funktion der Schlüsselableitung ([HKDF](#)), Signatur und einen 256-Bit-Verschlüsselungsschlüssel. Wenn das Symbol [Commit Politik](#) erfordert [Schlüsselliches Engagement](#), der AWS Encryption SDK wählt eine Algorithmen-Suite aus, die auch die Schlüsselverpflichtung unterstützt, andernfalls wählt sie eine Algorithmen-Suite mit Schlüsselableitung und Signierung aus, aber nicht mit Schlüsselverpflichtung.

Empfohlen: AES-GCM mit Schlüsselableitung, Signatur und Schlüsselverpflichtung

Die AWS Encryption SDK empfiehlt eine Algorithmen-Suite, die einen AES-GCM-Verschlüsselungsschlüssel ableitet, indem ein 256-Bit-Datenverschlüsselungsschlüssel für den HMAC bereitgestellt wird extract-and-expand Schlüsselableitungsfunktion (HKDF). Die AWS Encryption SDK fügt eine ECDSA-Signatur (Elliptic Curve Digital Signature Algorithm) hinzu. Zu - Support [Schlüsselliches Engagement](#) leitet diese Algorithmen-Suite auch ein Schlüsselzeichenfolge für Verpflichtungen— Ein ungeheimer Datenschlüssel-Identifikator — der in den Metadaten der verschlüsselten Nachricht gespeichert wird. Diese wichtige Verpflichtungszeichenfolge wird auch durch HKDF mit einem Verfahren abgeleitet, das dem Ableiten des Datenverschlüsselungsschlüssels ähnelt.

AWS Encryption SDK Algorithmen-Paket

Verschlüsselungsalgorithmus	Länge des Datenverschlüsselungsschlüssels (in Bit)	Schlüsselableitungsalgorithmus	Signatur-Algorithmus	Wesentliche Verpflichtung
AES-GCM	256	HKDF mit SHA-384	ECDSA mit P-384 und SHA-384	HKDF mit SHA-512

Das HKDF hilft Ihnen, eine unbeabsichtigte Wiederverwendung eines Datenverschlüsselungsschlüssels zu vermeiden und reduziert das Risiko einer Übernutzung eines Datenschlüssels.

Zum Signieren verwendet dieses Algorithmen-Paket ECDSA mit einem kryptografischen Hash-Funktionsalgorithmus (SHA-384). ECDSA wird standardmäßig verwendet, auch wenn es nicht in der Richtlinie für den zugrundeliegenden Masterschlüssel angegeben ist. [Signieren von Nachrichten](#) überprüft, dass der Absender der Nachricht berechtigt war, Nachrichten zu verschlüsseln, und stellt keine Ablehnung bereit. Sie ist besonders nützlich, wenn die Berechtigungsrichtlinie für einen Masterschlüssel es einer Gruppe von Benutzern erlaubt, Daten zu verschlüsseln, und einer anderen Gruppe von Benutzern, Daten zu entschlüsseln.

Algorithm Suites mit wesentlicher Verpflichtung stellen sicher, dass jeder Chiffretext nur in einen Klartext entschlüsselt wird. Sie tun dies, indem sie die Identität des Datenschlüssels validieren, der als Eingabe für den Verschlüsselungsalgorithmus verwendet wird. Bei der Verschlüsselung leiten diese Algorithmen-Suiten eine Schlüsselverbindungszeichenfolge ab. Vor dem Entschlüsseln überprüfen sie, ob der Datenschlüssel mit der Schlüsselverpflichtungszeichenfolge übereinstimmt. Wenn dies nicht der Fall ist, schlägt der Entschlüsselungsaufruf fehl.

Andere unterstützte Algorithmen-Pakete

Das AWS Encryption SDK unterstützt die folgenden alternativen Algorithmen-Pakete für Abwärtskompatibilität. Im Allgemeinen empfehlen wir diese Algorithmen-Pakete nicht. Wir sind uns jedoch bewusst, dass die Unterzeichnung die Leistung erheblich behindern kann, daher bieten wir eine wichtige Committing-Suite mit Schlüsselableitung für diese Fälle an. Für Anwendungen, die

bedeutendere Leistungskompromisse eingehen müssen, bieten wir weiterhin Suiten an, denen es an Unterzeichnung, Schlüsselverpflichtung und wichtige Ableitung mangelt.

AES-GCM ohne Schlüsselverpflichtung

Algorithm Suites ohne Schlüsselverpflichtung validieren den Datenschlüssel vor dem Entschlüsseln nicht. Infolgedessen entschlüsseln diese Algorithm-Suiten möglicherweise einen einzelnen Chiffretext in verschiedene Klartext-Nachrichten. Da Algorithm-Suiten mit wesentlicher Verpflichtung jedoch ein [etwas größere \(+30 Byte\) verschlüsselte Nachricht](#) und dauert länger, um zu verarbeiten, sie sind möglicherweise nicht die beste Wahl für jede Anwendung.

Die AWS Encryption SDK unterstützt eine Algorithm-Suite mit Schlüsselableitung, Schlüsselverpflichtung, Unterzeichnung und eine mit Schlüsselableitung und Schlüsselverpflichtung, aber nicht Unterzeichnung. Wir raten von der Verwendung einer Algorithmen-Paket ohne Schlüsselverpflichtung ab. Wenn nötig, empfehlen wir eine Algorithm-Suite mit Schlüsselableitung und Schlüsselverpflichtung, aber nicht signieren. Wenn Ihr Anwendungsleistungsprofil jedoch die Verwendung einer Algorithm-Suite unterstützt, ist die Verwendung einer Algorithm-Suite mit wesentlicher Verpflichtung, Schlüsselableitung und Signierung eine bewährte Methode.

AES-GCM ohne Signatur

Dieses Algorithmen-Paket ohne Signatur fehlt die ECDSA-Signatur, die Authentizität und Nachweisbarkeitskonzeption bietet. Verwenden Sie dieses Paket nur, wenn die Benutzer, die Daten verschlüsseln und die, die Daten entschlüsseln, gleichermaßen vertrauenswürdig sind.

Wenn Sie eine Algorithm-Suite ohne Signieren verwenden, empfehlen wir Ihnen, eine mit Schlüsselableitung und Schlüsselverpflichtung zu wählen.

AES-GCM ohne Schlüsselableitung

Algorithmen-Paket ohne Schlüsselableitung verwenden den Datenverschlüsselungsschlüssel als AES-GCM-Verschlüsselungsschlüssel, anstatt eine Schlüsselableitungsfunktion zu verwenden, um einen eindeutigen Schlüssel abzuleiten. Wir raten davon ab, dieses Paket zur Generierung von Chiffretext zu verwenden, aber AWS Encryption SDK unterstützt es aus Kompatibilitätsgründen.

Weitere Informationen darüber, wie diese Pakete in der Bibliothek dargestellt und verwendet werden, finden Sie unter [the section called "Algorithmen – Referenz"](#).

Verwenden der AWS Encryption SDK mit AWS KMS

Um den verwenden zu können AWS Encryption SDK, müssen Sie [Schlüsselringe](#) oder [Master-Key-Anbieter mit Wrapping-Schlüsseln](#) konfigurieren. Wenn Sie keine Schlüsselinfrastruktur haben, empfehlen wir die Verwendung von [AWS Key Management Service \(AWS KMS\)](#). Viele der Codebeispiele in der AWS Encryption SDK erfordern eine [AWS KMS key](#).

Für die Interaktion mit AWS KMS AWS Encryption SDK ist das AWS SDK für Ihre bevorzugte Programmiersprache erforderlich. Die AWS Encryption SDK Clientbibliothek arbeitet mit den AWS SDKs zusammen, um die darin gespeicherten Masterschlüssel zu unterstützen AWS KMS.

So bereiten Sie die Verwendung des AWS Encryption SDK mit AWS KMS vor

1. Erstellen einer AWS-Konto VPC Weitere Informationen dazu finden Sie unter [Wie erstelle und aktiviere ich ein neues -Konto für Amazon Web Services?](#) im AWS Knowledge Center.
2. Erstellen Sie eine symmetrische Verschlüsselung AWS KMS key. Hilfe finden Sie im AWS Key Management Service Entwicklerhandbuch unter [Creating Keys](#).

Tip

Um den AWS KMS key programmgesteuert verwenden zu können, benötigen Sie die Schlüssel-ID oder den Amazon-Ressourcennamen (ARN) des AWS KMS key. Hilfe bei der Suche nach der ID oder dem ARN eines AWS KMS key [finden Sie im AWS Key Management Service Entwicklerhandbuch unter Finden der Schlüssel-ID und des ARN](#).

3. Generieren Sie eine Zugriffsschlüssel-ID und einen Sicherheitszugriffsschlüssel. Sie können entweder die Zugriffsschlüssel-ID und den geheimen Zugriffsschlüssel für einen IAM-Benutzer verwenden, oder Sie können den verwenden, AWS Security Token Service um eine neue Sitzung mit temporären Sicherheitsanmeldeinformationen zu erstellen, die eine Zugriffsschlüssel-ID, einen geheimen Zugriffsschlüssel und ein Sitzungstoken enthalten. Im Sinne bewährter Sicherheitsmethoden wird empfohlen, temporäre Anmeldeinformationen anstelle bewährter Anmeldeinformationen für Ihre IAM-Benutzer- oder AWS (Root-) -Benutzerkonten zu nutzen.

Informationen zum Erstellen eines IAM-Benutzers mit einem Zugriffsschlüssel finden Sie unter [IAM-Benutzer erstellen](#) im IAM-Benutzerhandbuch.

Informationen zum Generieren temporärer Sicherheitsanmeldeinformationen finden Sie im IAM-Benutzerhandbuch unter [Anfordern temporärer Sicherheitsanmeldeinformationen](#).

4. Geben Sie Ihre AWS Anmeldeinformationen anhand der Anweisungen in [AWS SDK for Java](#), [AWS SDK for JavaScript](#), [AWS SDK for Python \(Boto\)](#) oder [AWS SDK for C++](#) (für C) sowie der Zugriffsschlüssel-ID und dem geheimen Zugriffsschlüssel ein, die Sie in Schritt 3 generiert haben. Wenn Sie temporäre Anmeldeinformationen generiert haben, müssen Sie auch das Sitzungstoken angeben.

Dieses Verfahren ermöglicht es AWS SDKs, Anfragen an AWS für Sie zu signieren. Code-Beispiele im AWS Encryption SDK, die mit AWS KMS interagieren, gehen davon aus, dass Sie diesen Schritt abgeschlossen haben.

5. Laden Sie das AWS Encryption SDK herunter und installieren Sie es. Weitere Informationen über die Installation finden Sie in den Anweisungen für die [Programmiersprache](#), die Sie verwenden möchten.

Bewährte Methoden für AWS Encryption SDK

Die AWS Encryption SDK wurde entwickelt, um Ihnen den Schutz Ihrer Daten unter Verwendung von Branchenstandards und bewährten Methoden zu erleichtern. Während viele Best Practices für Sie in Standardwerten ausgewählt werden, sind einige Vorgehensweisen optional, werden jedoch empfohlen, wann immer es praktikabel ist.

Verwenden Sie die aktuelle Version

Wenn Sie beginnen, das AWS Encryption SDK, verwenden Sie die aktuelle Version, die in Ihrer bevorzugten [Programmiersprache](#) aus. Wenn Sie das AWS Encryption SDK, führen Sie so bald wie möglich ein Upgrade auf die aktuelle Version durch. Dadurch wird sichergestellt, dass Sie die empfohlene Konfiguration verwenden und die neuen Sicherheitseigenschaften zum Schutz Ihrer Daten nutzen. Einzelheiten zu unterstützten Versionen, einschließlich Anleitungen für Migration und Bereitstellung, finden Sie unter [Support und Wartung](#) und [Versionen der AWS Encryption SDK](#) aus.

Wenn eine neue Version Elemente in Ihrem Code veraltet, ersetzen Sie sie so schnell wie möglich. Verwertungswarnungen und Codekommentare empfehlen in der Regel eine gute Alternative.

Um wichtige Upgrades einfacher und weniger fehleranfällig zu machen, bieten wir gelegentlich eine temporäre Version oder eine Übergangsversion an. Verwenden Sie diese Releases und die dazugehörige Dokumentation, um sicherzustellen, dass Sie Ihre Anwendung aktualisieren können, ohne Ihren Produktionsablauf zu unterbrechen.

Verwenden von Standardwerten

Die AWS Encryption SDK entwirft Best Practices in ihre Standardwerte. Verwenden Sie sie, wann immer möglich. Für Fälle, in denen der Standard nicht praktikabel ist, bieten wir Alternativen an, z. B. Algorithmus-Suites ohne Signierung. Wir bieten auch fortgeschrittenen Benutzern Möglichkeiten zur Anpassung, z. B. benutzerdefinierte Schlüsselringe, Master Key-Anbieter und kryptografische Materialmanager (CMMs). Verwenden Sie diese fortschrittlichen Alternativen vorsichtig und lassen Sie Ihre Auswahl nach Möglichkeit von einem Sicherheitsingenieur überprüfen.

Verwenden eines Verschlüsselungskontexts

Um die Sicherheit Ihrer kryptographischen Operationen zu verbessern, fügen Sie ein [Verschlüsselungskontext](#) mit einem aussagekräftigen Wert in allen Anfragen zur

Verschlüsselung von Daten. Die Verwendung eines Verschlüsselungskontexts ist optional, aber wir empfehlen dies als bewährte Methode für die Kryptografie. Ein Verschlüsselungskontext stellt zusätzliche authentifizierte Daten (AAD) für authentifizierte Verschlüsselungsdaten im AWS Encryption SDK aus. Obwohl dies nicht geheim ist, kann der Verschlüsselungskontext Ihnen helfen [die Integrität und Authentizität schützen](#) Ihrer verschlüsselten Daten.

In der AWS Encryption SDK verwenden, geben Sie nur beim Verschlüsseln einen Verschlüsselungskontext an. Beim Entschlüsseln wird der AWS Encryption SDK verwendet den Verschlüsselungskontext im Header der verschlüsselten Nachricht, dass das AWS Encryption SDK gibt zurück. Bevor Ihre Anwendung Klartext-Daten zurückgibt, stellen Sie sicher, dass der Verschlüsselungskontext, den Sie zum Verschlüsseln der Nachricht verwendet haben, im Verschlüsselungskontext enthalten ist, der zum Entschlüsseln der Nachricht verwendet wurde. Weitere Informationen finden Sie in den Beispielen in Ihrer Programmiersprache.

Wenn Sie die Befehlszeilen-Schnittstelle verwenden, wird das AWS Encryption SDK überprüft den Verschlüsselungskontext für Sie.

Schützen Sie Ihre Verpackungsschlüssel

Die AWS Encryption SDK generiert einen eindeutigen Datenschlüssel zum Verschlüsseln jeder Klartext-Nachricht. Dann verschlüsselt es den Datenschlüssel mit den von Ihnen bereitgestellten Umbruchschlüsseln. Wenn Ihre Wrapping Keys verloren gehen oder gelöscht werden, können Ihre verschlüsselten Daten nicht wiederhergestellt werden. Wenn Ihre Schlüssel nicht gesichert sind, sind Ihre Daten möglicherweise anfällig.

Verwenden Sie Wrapping Keys, die durch eine sichere Schlüsselinfrastruktur geschützt sind, wie [AWS Key Management Service](#) (AWS KMS) enthalten. Wenn Sie AES- oder RSA-Rohschlüsseln verwenden, verwenden Sie eine zufällige und dauerhafte Speicherquelle, die Ihre Anforderungen erfüllt. Generieren und Speichern von Hüllungsschlüsseln in einem HSM oder einem Dienst, der HSM bereitstellt, wie AWS CloudHSM, ist eine bewährte Methode.

Verwenden Sie die Autorisierungsmechanismen Ihrer Schlüsselinfrastruktur, um den Zugriff auf Ihre Wrapping Keys auf die Benutzer zu beschränken, die ihn benötigen. Implementieren Sie Best Practice-Prinzipien wie Least Privilege. Bei Verwendung von AWS KMS keys, verwenden Sie Schlüsselrichtlinien und IAM-Richtlinien, die [Bewährte Methoden](#) aus.

Geben Sie Ihre Verpackungsschlüssel an

Es ist immer eine Best Practice [geben Sie Ihre Umbruchschlüssel](#) explizit beim Entschlüsseln sowie beim Verschlüsseln. Wenn du das tust, wird der AWS Encryption SDK verwendet nur die von Ihnen angegebenen Schlüssel. Diese Vorgehensweise stellt sicher, dass Sie nur die

beabsichtigten Verschlüsselungsschlüssel verwenden. Für AWS KMS Das Umschließen von Schlüsseln verbessert auch die Leistung, indem verhindert wird, dass Sie versehentlich Schlüssel in einem anderen verwenden AWS-Konto oder Region oder versuchen, mit Schlüsseln zu entschlüsseln, für die Sie keine Berechtigung haben.

Bei der Verschlüsselung werden die Schlüsselbunde und Master Key-Anbieter, die der AWS Encryption SDK Verbrauchsmaterialien erfordern die Angabe von Umbruchschlüsseln. Sie verwenden alle und nur die von Ihnen angegebenen Umbruchschlüssel. Sie müssen auch Umbruchschlüssel angeben, wenn Sie mit AES-Rohschlüsselringen, RSA-Rohschlüsseln und JCE verschlüsseln und entschlüsseln Master Keys aus.

Beim Entschlüsseln mit AWS KMS Schlüsselbunde und Master Key-Anbieter müssen Sie keine Umbruchschlüssel angeben. Die AWS Encryption SDK kann die Schlüsselkennung aus den Metadaten des verschlüsselten Datenschlüssels abrufen. Aber die Angabe von Umschließen von Schlüsseln ist eine bewährte bewährte Methode, die wir empfehlen

Unterstützung dieser Best Practice bei der Arbeit mit AWS KMS Wenn Sie Schlüssel umschließen, empfehlen wir die folgenden Schritte:

- Verwenden von AWS KMS Schlüsselbunde, die Umbruchschlüssel angeben. Beim Verschlüsseln und Entschlüsseln verwenden diese Schlüsselbunde nur die von Ihnen angegebenen Umbruchschlüssel.
- Bei Verwendung von AWS KMS Master Keys und Master Key-Provider verwenden die Strict-Modus-Konstrukturen, die in [Version 1.7.x](#) der AWS Encryption SDK aus. Sie erstellen Anbieter, die nur mit den von Ihnen angegebenen Umbruchschlüsseln ver- und entschlüsseln. Konstrukturen für Master Key-Anbieter, die immer mit einem beliebigen Wrapping-Schlüssel entschlüsseln, sind in Version 1.7 veraltet und wurde in Version 2.0 gelöscht.

Bei der Angabe AWS KMS Das Umschließen von Schlüsseln zum Entschlüsseln ist unpraktisch. Sie können Discovery-Anbieter verwenden. Die AWS Encryption SDK in C und JavaScript Unterstützung [AWS KMS-Discovery-Schlüsselbund](#) aus. Master Key-Anbieter mit einem Erkennungsmodus sind für Java und Python in den Versionen 1.7 verfügbar und höher. Diese Discovery-Anbieter, die nur zum Entschlüsseln mit verwendet werden AWS KMS Schlüssel umbrechen, leiten Sie explizit die AWS Encryption SDK um einen beliebigen Umbruchschlüssel zu verwenden, der einen Datenschlüssel verschlüsselt hat.

Wenn Sie einen Discovery-Anbieter verwenden müssen, verwenden Sie dessen-Discovery-Filter Funktionen, um die von ihnen verwendeten Umbruchschlüssel einzuschränken. Zum Beispiel, das [AWS KMS regionalen -Discovery-Schlüsselbund](#) verwendet nur die

Umbruchschlüssel in einem bestimmten AWS-Region aus. Sie können auch konfigurieren AWS KMS-Schlüsselbunden und AWS KMS [-Masterschlüsselanbieter](#) nur das zu verwenden [Schlüssel umschließen](#) insbesondere AWS-Konten aus. Verwenden Sie wie immer Schlüsselrichtlinien und IAM-Richtlinien, um den Zugriff auf Ihre AWS KMS Schlüssel umbrechen.

Verwenden Sie digitale Signaturen

Es ist eine bewährte Methode, eine Algorithmus-Suite mit Signieren zu verwenden. [Digitale Signaturen](#) überprüfen Sie, ob der Absender der Nachricht autorisiert war, die Nachricht zu senden, und schützen Sie die Integrität der Nachricht. Alle Versionen des AWS Encryption SDK verwendet standardmäßig Algorithmus-Suites mit Signierung.

Wenn Ihre Sicherheitsanforderungen keine digitalen Signaturen beinhalten, können Sie eine Algorithmus-Suite ohne digitale Signaturen auswählen. Wir empfehlen jedoch, digitale Signaturen zu verwenden, insbesondere wenn eine Benutzergruppe Daten verschlüsselt und eine andere Gruppe von Benutzern diese Daten entschlüsselt.

Verwenden Sie wichtige Verpflichtung

Es ist eine bewährte Methode, die wichtige Sicherheitsfunktion für Verpflichtungen zu verwenden. Durch Überprüfung der Überprüfung der Identität des Unikats [Datenschlüssel](#) die deine Daten verschlüsselt haben, [Schlüsselhafte Verpflichtung](#) verhindert, dass Sie Chiffretext entschlüsseln, der zu mehr als einer Klartext-Nachricht führen könnte.

Die AWS Encryption SDK bietet volle Unterstützung für das Verschlüsseln und Entschlüsseln mit einer Schlüsselverpflichtung ab [Version 2.0.x](#) aus. Standardmäßig werden alle Ihre Nachrichten mit Schlüsselverpflichtung verschlüsselt und entschlüsselt. [Version 1.7.x](#) der AWS Encryption SDK kann Chiffretexte mit Schlüsselverpflichtung entschlüsseln. Es wurde entwickelt, um Benutzern früherer Versionen bei der Bereitstellung von Version 2.0 zu helfen. erfolgreich.

Die Support für wichtige Verpflichtungen umfasst [neue Algorithmen-Pakete](#) und ein [neues Nachrichtenformat](#) das erzeugt einen Chiffretext, der nur 30 Byte größer ist als ein Chiffretext ohne Schlüsselverpflichtung. Das Design minimiert die Auswirkungen auf die Leistung, sodass die meisten Benutzer die Vorteile eines wichtigen Engagements nutzen können. Wenn Ihre Anwendung sehr empfindlich auf Größe und Leistung reagiert, können Sie sich für die [Commitment](#) Einstellung, um die Schlüsselübergabe zu deaktivieren oder die AWS Encryption SDK um Nachrichten unverbindlich zu entschlüsseln, aber tun Sie dies nur, wenn Sie müssen.

Begrenzung der Anzahl der verschlüsselten Datenschlüssel

Es ist eine bewährte Methode, [begrenzt die Anzahl der verschlüsselten Datenschlüssel](#) in Nachrichten, die Sie entschlüsseln, insbesondere Nachrichten aus nicht vertrauenswürdigen

Quellen. Das Entschlüsseln einer Nachricht mit zahlreichen verschlüsselten Datenschlüsseln, die Sie nicht entschlüsseln können, kann zu längeren Verzögerungen führen, Kosten verursachen, Ihre Anwendung und andere, die Ihr Konto gemeinsam nutzen, drosseln und möglicherweise Ihre Schlüsselinfrastruktur erschöpfen. Ohne Einschränkungen kann eine verschlüsselte Nachricht bis zu $65.535 (2^{16} - 1)$ verschlüsselte Datenschlüssel haben. Details hierzu finden Sie unter [Beschränkung verschlüsselter Datenschlüssel](#).

Weitere Informationen zu AWS Encryption SDK Sicherheitsfunktionen, die diesen bewährten Methoden zugrunde liegen, finden Sie unter [Verbesserte clientseitige Verschlüsselung: Expliz KeyIds und Schlüsselverpflichtung](#) im AWS Blogaus.

Konfiguration der AWS Encryption SDK

Das AWS Encryption SDK ist so konzipiert, dass es einfach zu bedienen ist. Obwohl es AWS Encryption SDK mehrere Konfigurationsoptionen gibt, wurden die Standardwerte sorgfältig ausgewählt, damit sie für die meisten Anwendungen praktisch und sicher sind. Möglicherweise müssen Sie jedoch Ihre Konfiguration anpassen, um die Leistung zu verbessern, oder eine benutzerdefinierte Funktion in Ihr Design aufnehmen.

Lesen Sie sich bei der Konfiguration Ihrer Implementierung die AWS Encryption SDK [bewährten Methoden](#) durch und implementieren Sie so viele wie möglich.

Themen

- [Auswahl einer Programmiersprache](#)
- [Auswahl von Schlüsseln zum Umbrechen](#)
- [Verwenden Sie mehrere Regionen AWS KMS keys](#)
- [Auswahl einer Algorithmus-Suite](#)
- [Beschränkung verschlüsselter Datenschlüssel](#)
- [Einen Discovery-Filter erstellen](#)
- [Festlegung einer Verpflichtungspolitik](#)
- [Arbeiten mit Streaming-Daten](#)
- [Zwischenspeichern von Datenschlüsseln](#)

Auswahl einer Programmiersprache

Das AWS Encryption SDK ist in mehreren [Programmiersprachen](#) verfügbar. Die Sprachimplementierungen sind so konzipiert, dass sie vollständig interoperabel sind und dieselben Funktionen bieten, obwohl sie möglicherweise auf unterschiedliche Weise implementiert werden. In der Regel verwenden Sie die Bibliothek, die mit Ihrer Anwendung kompatibel ist. Sie können jedoch eine Programmiersprache für eine bestimmte Implementierung auswählen. Wenn Sie beispielsweise lieber mit [Schlüsselanhängern](#) arbeiten, können Sie den AWS-Verschlüsselungs-SDK for C oder den AWS-Verschlüsselungs-SDK for JavaScript wählen.

Auswahl von Schlüsseln zum Umbrechen

Der AWS Encryption SDK generiert einen eindeutigen symmetrischen Datenschlüssel, um jede Nachricht zu verschlüsseln. Sofern Sie das [Zwischenspeichern von Datenschlüsseln nicht verwenden, müssen Sie die Datenschlüssel](#) nicht konfigurieren, verwalten oder verwenden. Das AWS Encryption SDK erledigt das für Sie.

Sie müssen jedoch einen oder mehrere Wrapping-Schlüssel auswählen, um jeden Datenschlüssel zu verschlüsseln. Der AWS Encryption SDK unterstützt AES symmetrische Schlüssel und RSA asymmetrische Schlüssel in verschiedenen Größen. Es unterstützt auch [AWS Key Management Service](#) (AWS KMS) symmetrische Verschlüsselung. AWS KMS keys Sie sind für die Sicherheit und Haltbarkeit Ihrer Wrapping-Schlüssel verantwortlich. Wir empfehlen Ihnen daher, einen Verschlüsselungsschlüssel in einem Hardware-Sicherheitsmodul oder einem wichtigen Infrastrukturdienst zu verwenden, z. B. AWS KMS

Um Ihre Wrapping-Schlüssel für die Verschlüsselung und Entschlüsselung anzugeben, verwenden Sie einen Schlüsselbund (C und JavaScript) oder einen Hauptschlüsselanbieter (Java, Python, AWS VerschlüsselungCLI). Sie können einen Umschließungsschlüssel oder mehrere Umschließungsschlüssel desselben oder verschiedener Typen angeben. Wenn Sie mehrere Umschließungsschlüssel verwenden, um einen Datenschlüssel zu umschließen, verschlüsselt jeder Umschließungsschlüssel eine Kopie desselben Datenschlüssels. Die verschlüsselten Datenschlüssel (einer pro Umschließungsschlüssel) werden zusammen mit den verschlüsselten Daten in der verschlüsselten Nachricht gespeichert, die AWS Encryption SDK zurückgegeben wird. Um die Daten zu entschlüsseln, AWS Encryption SDK müssen sie zuerst einen Ihrer Verpackungsschlüssel verwenden, um einen verschlüsselten Datenschlüssel zu entschlüsseln.

Um AWS KMS key in einem Schlüsselbund oder einem Hauptschlüsselanbieter anzugeben, verwenden Sie eine unterstützte AWS KMS Schlüssel-ID. Einzelheiten zu den Schlüsselbezeichnern für einen AWS KMS Schlüssel finden Sie unter [Schlüsselkennungen im AWS Key Management Service Entwicklerhandbuch](#).

- Bei der Verschlüsselung mit der AWS-Verschlüsselungs-SDK for Java, AWS-Verschlüsselungs-SDK for JavaScript AWS-Verschlüsselungs-SDK for Python, oder der AWS Verschlüsselung CLI können Sie jede gültige Schlüssel-ID (Schlüssel-ID, SchlüsselARN, Aliasname oder AliasARN) für einen Schlüssel verwenden. KMS Bei der Verschlüsselung mit dem AWS-Verschlüsselungs-SDK for C können Sie nur eine Schlüssel-ID oder einen Schlüssel verwenden. ARN

Wenn Sie bei der Verschlüsselung einen Aliasnamen oder Alias ARN für einen KMS Schlüssel angeben, AWS Encryption SDK speichert der den Schlüssel, der ARN aktuell mit diesem Alias

verknüpft ist. Der Alias wird nicht gespeichert. Änderungen am Alias wirken sich nicht auf den KMS Schlüssel aus, der zum Entschlüsseln Ihrer Datenschlüssel verwendet wird.

- Bei der Entschlüsselung im strikten Modus (in dem Sie bestimmte Wrapping-Schlüssel angeben) müssen Sie einen Schlüssel ARN zur Identifizierung verwenden. AWS KMS keys Diese Anforderung gilt für alle Sprachenimplementierungen des AWS Encryption SDK.

Wenn Sie mit einem AWS KMS Schlüsselbund verschlüsseln, AWS Encryption SDK speichert der den Schlüssel ARN von AWS KMS key in den Metadaten des verschlüsselten Datenschlüssels. Bei der Entschlüsselung im strikten Modus AWS Encryption SDK überprüft der, ob derselbe Schlüssel im ARN Schlüsselbund (oder Hauptschlüsselanbieter) enthalten ist, bevor versucht wird, den verschlüsselten Datenschlüssel mit dem Wrapping-Schlüssel zu entschlüsseln. Wenn Sie eine andere Schlüssel-ID verwenden, AWS Encryption SDK wird sie nicht erkannt oder verwendet AWS KMS key, auch wenn sich die Kennungen auf denselben Schlüssel beziehen.

Um einen [AESRohschlüssel](#) oder ein [RSARohschlüsselpaar](#) als Umschließungsschlüssel in einem Schlüsselbund anzugeben, müssen Sie einen Namespace und einen Namen angeben. In einem Hauptschlüsselanbieter `Provider ID` entspricht der dem Namespace und der `Key ID` entspricht dem Namen. Beim Entschlüsseln müssen Sie für jeden Rohverpackungsschlüssel genau denselben Namespace und denselben Namen verwenden wie beim Verschlüsseln. Wenn Sie einen anderen Namespace oder Namen verwenden, AWS Encryption SDK wird der Wrapping-Schlüssel nicht erkannt oder verwendet, selbst wenn das Schlüsselmaterial identisch ist.

Verwenden Sie mehrere Regionen AWS KMS keys

Sie können AWS Key Management Service (AWS KMS) Schlüssel mit mehreren Regionen verwenden, um Schlüssel in der einzuschließen. AWS Encryption SDK Wenn Sie mit einem Schlüssel für mehrere Regionen in einem verschlüsseln AWS-Region, können Sie mit einem zugehörigen Schlüssel für mehrere Regionen in einem anderen verschlüsseln. AWS-Region Die Support für Schlüssel mit mehreren Regionen wurde in Version 2.3 eingeführt. x der Version AWS Encryption SDK und Version 3.0. x der AWS VerschlüsselungCLI.

AWS KMS Schlüssel für mehrere Regionen bestehen aus AWS KMS keys verschiedenen Schlüsseln AWS-Regionen , die dasselbe Schlüsselmaterial und dieselbe Schlüssel-ID haben. Sie können diese verwandten Schlüssel so verwenden, als ob es sich um denselben Schlüssel in verschiedenen Regionen handeln würde. Schlüssel mit mehreren Regionen unterstützen gängige Notfallwiederherstellungs- und Sicherungsszenarien, bei denen die Verschlüsselung in einer Region und die Entschlüsselung in einer anderen Region erforderlich ist, ohne dass

ein regionsübergreifender Anruf erforderlich ist. AWS KMS Informationen zu Schlüsseln für mehrere Regionen finden Sie unter [Verwenden von Schlüsseln für mehrere Regionen](#) im Entwicklerhandbuch. AWS Key Management Service

Zur Unterstützung von Schlüsseln für mehrere Regionen AWS Encryption SDK umfasst dies Schlüsselringe, die AWS KMS mehrere Regionen berücksichtigen, und Hauptschlüsselanbieter. Das neue Symbol für mehrere Regionen in jeder Programmiersprache unterstützt sowohl Schlüssel für einzelne Regionen als auch Schlüssel für mehrere Regionen.

- Bei Schlüsseln mit nur einer Region verhält sich das Symbol für mehrere Regionen genauso wie der Schlüsselbund für einzelne Regionen und der Hauptschlüsselanbieter. AWS KMS Es versucht, Chiffretext nur mit dem Schlüssel für eine einzelne Region zu entschlüsseln, mit dem die Daten verschlüsselt wurden.
- [Bei Schlüsseln mit mehreren Regionen versucht das Symbol für mehrere Regionen, Chiffretext mit demselben Schlüssel für mehrere Regionen zu entschlüsseln, mit dem die Daten verschlüsselt wurden, oder mit dem zugehörigen Replikatschlüssel für mehrere Regionen in der von Ihnen angegebenen Region.](#)

In den Schlüsselbunden für mehrere Regionen und bei Anbietern von Hauptschlüsseln, die mehr als einen Schlüssel verwenden, können Sie mehrere Schlüssel für einzelne Regionen und Schlüssel für mehrere Regionen angeben. KMS Sie können jedoch nur einen Schlüssel aus jedem Satz verwandter Replikatschlüssel für mehrere Regionen angeben. Wenn Sie mehr als einen Schlüsselbezeichner mit derselben Schlüssel-ID angeben, schlägt der Konstruktoraufruf fehl.

Sie können auch einen Schlüssel für mehrere Regionen zusammen mit den standardmäßigen AWS KMS Schlüsselanhängern und Hauptschlüsselanbietern für einzelne Regionen verwenden. Sie müssen jedoch denselben Schlüssel für mehrere Regionen in derselben Region zum Verschlüsseln und Entschlüsseln verwenden. Die Schlüsselringe für einzelne Regionen und die Anbieter von Masterschlüsseln versuchen, Chiffretext nur mit den Schlüsseln zu entschlüsseln, mit denen die Daten verschlüsselt wurden.

Die folgenden Beispiele zeigen, wie Daten mithilfe von Schlüsseln für mehrere Regionen und den neuen Schlüsselanhängern und Masterschlüsselanbietern, die mehrere Regionen unterstützen, ver- und entschlüsselt werden. In diesen Beispielen werden Daten in der Region verschlüsselt und die Daten in der us-east-1 Region mithilfe verwandter multiregionaler Replikatschlüssel in jeder us-west-2 Region entschlüsselt. Bevor Sie diese Beispiele ausführen, ersetzen Sie den Beispielschlüssel für mehrere Regionen durch einen ARN gültigen Wert aus Ihrem AWS-Konto

C

Um mit einem Schlüssel für mehrere Regionen zu verschlüsseln, verwenden Sie die `Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder()` Methode, um den Schlüsselbund zu instanziiieren. Geben Sie einen Schlüssel für mehrere Regionen an.

Dieses einfache Beispiel enthält keinen [Verschlüsselungskontext](#). Ein Beispiel, das einen Verschlüsselungskontext in C verwendet, finden Sie unter [Verschlüsseln und Entschlüsseln von Zeichenfolgen](#).

Ein vollständiges Beispiel finden Sie unter [kms_multi_region_keys.cpp](#) im AWS-Verschlüsselungs-SDK for C Repository unter GitHub.

```
/* Encrypt with a multi-Region KMS key in us-east-1 */

/* Load error strings for debugging */
aws_cryptosdk_load_error_strings();

/* Initialize a multi-Region keyring */
const char *mrk_us_east_1 = "arn:aws:kms:us-east-1:111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab";

struct aws_cryptosdk_keyring *mrk_keyring =
    Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder().Build(mrk_us_east_1);

/* Create a session; release the keyring */
struct aws_cryptosdk_session *session =
    aws_cryptosdk_session_new_from_keyring_2(aws_default_allocator(),
    AWS_CRYPTOSDK_ENCRYPT, mrk_keyring);

aws_cryptosdk_keyring_release(mrk_keyring);

/* Encrypt the data
 * aws_cryptosdk_session_process_full is designed for non-streaming data
 */
aws_cryptosdk_session_process_full(
    session, ciphertext, ciphertext_buf_sz, &ciphertext_len, plaintext,
    plaintext_len));

/* Clean up the session */
aws_cryptosdk_session_destroy(session);
```

C# / .NET

Um mit einem Schlüssel für mehrere Regionen in der Region USA Ost (Nord-Virginia) (us-east-1) zu verschlüsseln, instanziiieren Sie ein `CreateAwsKmsMrkKeyringInput` Objekt mit einer Schlüssel-ID für den Schlüssel für mehrere Regionen und einem Client für die angegebene Region. AWS KMS Verwenden Sie dann die Methode, um den Schlüsselbund zu erstellen.

```
CreateAwsKmsMrkKeyring()
```

Die `CreateAwsKmsMrkKeyring()` Methode erstellt einen Schlüsselbund mit genau einem Schlüssel für mehrere Regionen. Verwenden Sie die Methode, um mit mehreren Umschließungsschlüsseln, einschließlich eines Schlüssels für mehrere Regionen, zu verschlüsseln. `CreateAwsKmsMrkMultiKeyring()`

Ein vollständiges Beispiel finden Sie unter [AwsKmsMrkKeyringExample.cs im Forum](#). AWS Encryption SDK NETRepository aktiviert. GitHub

```
//Encrypt with a multi-Region KMS key in us-east-1 Region

// Instantiate the AWS Encryption SDK and material providers
var encryptionSdk = AwsEncryptionSdkFactory.CreateDefaultAwsEncryptionSdk();
var materialProviders =

    AwsCryptographicMaterialProvidersFactory.CreateDefaultAwsCryptographicMaterialProviders();

// Multi-Region keys have a distinctive key ID that begins with 'mrk'
// Specify a multi-Region key in us-east-1
string mrkUSEast1 = "arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab";

// Create the keyring
// You can specify the Region or get the Region from the key ARN
var createMrkEncryptKeyringInput = new CreateAwsKmsMrkKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(RegionEndpoint.USEast1),
    KmsKeyId = mrkUSEast1
};
var mrkEncryptKeyring =
    materialProviders.CreateAwsKmsMrkKeyring(createMrkEncryptKeyringInput);

// Define the encryption context
var encryptionContext = new Dictionary<string, string>()
```

```

{
    {"purpose", "test"}
};

// Encrypt your plaintext data.
var encryptInput = new EncryptInput
{
    Plaintext = plaintext,
    Keyring = mrkEncryptKeyring,
    EncryptionContext = encryptionContext
};
var encryptOutput = encryptionSdk.Encrypt(encryptInput);

```

AWS Encryption CLI

In diesem Beispiel wird die `hello.txt` Datei unter einem Schlüssel für mehrere Regionen in der Region `us-east-1` verschlüsselt. Da in dem Beispiel ein Schlüssel ARN mit einem Region-Element angegeben wird, wird in diesem Beispiel das Region-Attribut des Parameters nicht verwendet. `--wrapping-keys`

Wenn die Schlüssel-ID des Wrapping-Schlüssels keine Region angibt, können Sie das Region-Attribut von verwenden, `--wrapping-keys` um die Region anzugeben, z. `--wrapping-keys key=$keyID region=us-east-1` B.

```

# Encrypt with a multi-Region KMS key in us-east-1 Region

# To run this example, replace the fictitious key ARN with a valid value.
$ mrkUSEast1=arn:aws:kms:us-east-1:111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab

$ aws-encryption-cli --encrypt \
    --input hello.txt \
    --wrapping-keys key=$mrkUSEast1 \
    --metadata-output ~/metadata \
    --encryption-context purpose=test \
    --output .

```

Java

Um mit einem Schlüssel für mehrere Regionen zu verschlüsseln, instanziiieren Sie einen `AwsKmsMrkAwareMasterKeyProvider` und geben Sie einen Schlüssel für mehrere Regionen an.

Ein vollständiges Beispiel finden Sie im Repository unter [BasicMultiRegionKeyEncryptionExample.java](#). AWS-Verschlüsselungs-SDK for Java GitHub

```
//Encrypt with a multi-Region KMS key in us-east-1 Region

// Instantiate the client
final AwsCrypto crypto = AwsCrypto.builder()
    .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
    .build();

// Multi-Region keys have a distinctive key ID that begins with 'mrk'
// Specify a multi-Region key in us-east-1
final String mrkUSEast1 = "arn:aws:kms:us-east-1:111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab";

// Instantiate an AWS KMS master key provider in strict mode for multi-Region keys
// Configure it to encrypt with the multi-Region key in us-east-1
final AwsKmsMrkAwareMasterKeyProvider kmsMrkProvider =
    AwsKmsMrkAwareMasterKeyProvider
        .builder()
        .buildStrict(mrkUSEast1);

// Create an encryption context
final Map<String, String> encryptionContext = Collections.singletonMap("Purpose",
    "Test");

// Encrypt your plaintext data
final CryptoResult<byte[], AwsKmsMrkAwareMasterKey> encryptResult =
    crypto.encryptData(
        kmsMrkProvider,
        encryptionContext,
        sourcePlaintext);
byte[] ciphertext = encryptResult.getResult();
```

JavaScript Browser

Um mit einem Schlüssel für mehrere Regionen zu verschlüsseln, verwenden Sie die `buildAwsKmsMrkAwareStrictMultiKeyringBrowser()` Methode, um den Schlüsselbund zu erstellen, und geben Sie einen Schlüssel für mehrere Regionen an.

Ein vollständiges Beispiel finden Sie unter [kms_multi_region_simple.ts im Repository](#) unter. AWS-Verschlüsselungs-SDK for JavaScript GitHub

```
/* Encrypt with a multi-Region KMS key in us-east-1 Region */

import {
  buildAwsKmsMrkAwareStrictMultiKeyringBrowser,
  buildClient,
  CommitmentPolicy,
  KMS,
} from '@aws-crypto/client-browser'

/* Instantiate an AWS Encryption SDK client */
const { encrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

declare const credentials: {
  accessKeyId: string
  secretAccessKey: string
  sessionToken: string
}

/* Instantiate an AWS KMS client
 * The AWS-Verschlüsselungs-SDK for JavaScript gets the Region from the key ARN
 */
const clientProvider = (region: string) => new KMS({ region, credentials })

/* Specify a multi-Region key in us-east-1 */
const multiRegionUsEastKey =
  'arn:aws:kms:us-east-1:111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab'

/* Instantiate the keyring */
const encryptKeyring = buildAwsKmsMrkAwareStrictMultiKeyringBrowser({
  generatorKeyId: multiRegionUsEastKey,
  clientProvider,
})

/* Set the encryption context */
const context = {
  purpose: 'test',
}
```



```
/* Test data to encrypt */
const cleartext = new Uint8Array([1, 2, 3, 4, 5])

/* Encrypt the data */
const { result } = await encrypt(encryptKeyring, cleartext, {
  encryptionContext: context,
})
```

JavaScript Node.js

Um mit einem Schlüssel für mehrere Regionen zu verschlüsseln, verwenden Sie die Methode, um den Schlüsselbund zu erstellen, und geben Sie einen Schlüssel für mehrere Regionen an. `buildAwsKmsMrkAwareStrictMultiKeyringNode()`

Ein vollständiges Beispiel finden Sie unter [kms_multi_region_simple.ts im Repository](#) unter [AWS-Verschlüsselungs-SDK for JavaScript GitHub](#)

```
//Encrypt with a multi-Region KMS key in us-east-1 Region

import { buildClient } from '@aws-crypto/client-node'

/* Instantiate the AWS Encryption SDK client
const { encrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

/* Test string to encrypt */
const cleartext = 'asdf'

/* Multi-Region keys have a distinctive key ID that begins with 'mrk'
 * Specify a multi-Region key in us-east-1
 */
const multiRegionUsEastKey =
  'arn:aws:kms:us-east-1:111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab'

/* Create an AWS KMS keyring */
const mrkEncryptKeyring = buildAwsKmsMrkAwareStrictMultiKeyringNode({
  generatorKeyId: multiRegionUsEastKey,
})

/* Specify an encryption context */
const context = {
  purpose: 'test',
```

```

}

/* Create an encryption keyring */
const { result } = await encrypt(mrkEncryptKeyring, cleartext, {
  encryptionContext: context,
})

```

Python

Um mit einem Schlüssel für mehrere Regionen zu verschlüsseln, verwenden Sie die Methode und geben Sie einen Schlüssel für AWS KMS mehrere Regionen an. `MRKAwareStrictAwsKmsMasterKeyProvider()`

Ein vollständiges Beispiel finden Sie unter [mrk_aware_kms_provider.py](#) im AWS-Verschlüsselungs-SDK for Python Repository unter. GitHub

```

* Encrypt with a multi-Region KMS key in us-east-1 Region

# Instantiate the client
client =
  aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_R

# Specify a multi-Region key in us-east-1
mrk_us_east_1 = "arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab"

# Use the multi-Region method to create the master key provider
# in strict mode
strict_mrk_key_provider = MRKAwareStrictAwsKmsMasterKeyProvider(
  key_ids=[mrk_us_east_1]
)

# Set the encryption context
encryption_context = {
  "purpose": "test"
}

# Encrypt your plaintext data
ciphertext, encrypt_header = client.encrypt(
  source=source_plaintext,
  encryption_context=encryption_context,
  key_provider=strict_mrk_key_provider
)

```

Verschieben Sie als Nächstes Ihren Chiffretext in die `us-west-2` Region. Sie müssen den Chiffretext nicht erneut verschlüsseln.

Um den Chiffretext im strikten Modus in der `us-west-2` Region zu entschlüsseln, instanzieren Sie das Symbol für mehrere Regionen mit dem Schlüssel des zugehörigen Multiregions-Schlüssels in der Region. ARN `us-west-2` Wenn Sie den Schlüssel ARN eines zugehörigen Schlüssels mit mehreren Regionen in einer anderen Region angeben (einschließlich des Ortes, an dem er verschlüsselt wurde) `us-east-1`, führt das Symbol für mehrere Regionen einen regionsübergreifenden Aufruf für diesen Schlüssel aus. `AWS KMS key`

Bei der Entschlüsselung im strikten Modus ist für das Symbol für mehrere Regionen ein Schlüssel erforderlich. ARN Es akzeptiert nur einen Schlüssel ARN aus jedem Satz verwandter Schlüssel für mehrere Regionen.

Bevor Sie diese Beispiele ausführen, ersetzen Sie den Beispielschlüssel für mehrere Regionen ARN durch einen gültigen Wert aus Ihrem `AWS-Konto`

C

Um im strikten Modus mit einem Schlüssel für mehrere Regionen zu entschlüsseln, verwenden Sie die `Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder()` Methode, um den Schlüsselbund zu instanzieren. Geben Sie den zugehörigen Schlüssel für mehrere Regionen in der lokalen Region (`us-west-2`) an.

Ein vollständiges Beispiel finden Sie unter [kms_multi_region_keys.cpp](#) im AWS-Verschlüsselungs-SDK for C Repository unter `GitHub`

```
/* Decrypt with a related multi-Region KMS key in us-west-2 Region */

/* Load error strings for debugging */
aws_cryptosdk_load_error_strings();

/* Initialize a multi-Region keyring */
const char *mrk_us_west_2 = "arn:aws:kms:us-west-2:111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab";

struct aws_cryptosdk_keyring *mrk_keyring =
    Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder().Build(mrk_us_west_2);

/* Create a session; release the keyring */
struct aws_cryptosdk_session *session =
```

```

    aws_cryptosdk_session_new_from_keyring_2(aws_default_allocator(),
    AWS_CRYPTOSDK_ENCRYPT, mrk_keyring);

aws_cryptosdk_session_set_commitment_policy(session,
    COMMITMENT_POLICY_REQUIRE_ENCRYPT_REQUIRE_DECRYPT);

aws_cryptosdk_keyring_release(mrk_keyring);

/* Decrypt the ciphertext
 * aws_cryptosdk_session_process_full is designed for non-streaming data
 */
aws_cryptosdk_session_process_full(
    session, plaintext, plaintext_buf_sz, &plaintext_len, ciphertext,
    ciphertext_len));

/* Clean up the session */
aws_cryptosdk_session_destroy(session);

```

C# / .NET

Um im strikten Modus mit einem einzigen Schlüssel für mehrere Regionen zu entschlüsseln, verwenden Sie dieselben Konstruktoren und Methoden, mit denen Sie die Eingabe zusammengestellt und den Schlüsselbund für die Verschlüsselung erstellt haben. Instanzieren Sie ein `CreateAwsKmsMrkKeyringInput` Objekt mit dem Schlüssel ARN eines zugehörigen Multiregions-Schlüssels und einem AWS KMS Client für die Region USA West (Oregon) (us-west-2). Verwenden Sie dann die `CreateAwsKmsMrkKeyring()` Methode, um einen Schlüsselbund für mehrere Regionen mit einem Schlüssel für mehrere Regionen zu erstellen. KMS

Ein vollständiges Beispiel finden Sie unter [AwsKmsMrkKeyringExample.cs](#) im Forum. AWS Encryption SDK NETRepository aktiviert. GitHub

```

// Decrypt with a related multi-Region KMS key in us-west-2 Region

// Instantiate the AWS Encryption SDK and material providers
var encryptionSdk = AwsEncryptionSdkFactory.CreateDefaultAwsEncryptionSdk();
var materialProviders =

    AwsCryptographicMaterialProvidersFactory.CreateDefaultAwsCryptographicMaterialProviders();

// Specify the key ARN of the multi-Region key in us-west-2

```

```

string mrkUSWest2 = "arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab";

// Instantiate the keyring input
// You can specify the Region or get the Region from the key ARN
var createMrkDecryptKeyringInput = new CreateAwsKmsMrkKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(RegionEndpoint.USWest2),
    KmsKeyId = mrkUSWest2
};

// Create the multi-Region keyring
var mrkDecryptKeyring =
    materialProviders.CreateAwsKmsMrkKeyring(createMrkDecryptKeyringInput);

// Decrypt the ciphertext
var decryptInput = new DecryptInput
{
    Ciphertext = ciphertext,
    Keyring = mrkDecryptKeyring
};
var decryptOutput = encryptionSdk.Decrypt(decryptInput);

```

AWS Encryption CLI

Um mit dem zugehörigen Schlüssel für mehrere Regionen in der Region us-west-2 zu entschlüsseln, verwenden Sie das Schlüsselattribut des `--wrapping-keys` Parameters, um seinen Schlüssel anzugeben. ARN

```

# Decrypt with a related multi-Region KMS key in us-west-2 Region

# To run this example, replace the fictitious key ARN with a valid value.
$ mrkUSWest2=arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab

$ aws-encryption-cli --decrypt \
    --input hello.txt.encrypted \
    --wrapping-keys key=$mrkUSWest2 \
    --commitment-policy require-encrypt-require-decrypt \
    --encryption-context purpose=test \
    --metadata-output ~/metadata \
    --max-encrypted-data-keys 1 \
    --buffer \

```

```
--output .
```

Java

Um im strikten Modus zu entschlüsseln, instanziiieren Sie einen `AwsKmsMrkAwareMasterKeyProvider` und geben Sie den zugehörigen Schlüssel für mehrere Regionen in der lokalen Region (`us-west-2`) an.

[Ein vollständiges Beispiel finden Sie unter `.java` im Repository unter `BasicMultiRegionKeyEncryptionExample`.](#) AWS-Verschlüsselungs-SDK for Java GitHub

```
// Decrypt with a related multi-Region KMS key in us-west-2 Region

// Instantiate the client
final AwsCrypto crypto = AwsCrypto.builder()
    .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
    .build();

// Related multi-Region keys have the same key ID. Their key ARNs differs only in
// the Region field.
String mrkUSWest2 = "arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab";

// Use the multi-Region method to create the master key provider
// in strict mode
AwsKmsMrkAwareMasterKeyProvider kmsMrkProvider =
    AwsKmsMrkAwareMasterKeyProvider.builder()
        .buildStrict(mrkUSWest2);

// Decrypt your ciphertext
CryptoResult<byte[], AwsKmsMrkAwareMasterKey> decryptResult = crypto.decryptData(
    kmsMrkProvider,
    ciphertext);
byte[] decrypted = decryptResult.getResult();
```

JavaScript Browser

Um im strikten Modus zu entschlüsseln, verwenden Sie die `buildAwsKmsMrkAwareStrictMultiKeyringBrowser()` Methode, um den Schlüsselbund zu erstellen, und geben Sie den zugehörigen Schlüssel für mehrere Regionen in der lokalen Region (`us-west-2`) an.

Ein vollständiges Beispiel finden Sie unter [kms_multi_region_simple.ts](#) im Repository unter [AWS-Verschlüsselungs-SDK for JavaScript GitHub](#)

```
/* Decrypt with a related multi-Region KMS key in us-west-2 Region */

import {
  buildAwsKmsMrkAwareStrictMultiKeyringBrowser,
  buildClient,
  CommitmentPolicy,
  KMS,
} from '@aws-crypto/client-browser'

/* Instantiate an AWS Encryption SDK client */
const { decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

declare const credentials: {
  accessKeyId: string
  secretAccessKey: string
  sessionToken: string
}

/* Instantiate an AWS KMS client
 * The AWS-Verschlüsselungs-SDK for JavaScript gets the Region from the key ARN
 */
const clientProvider = (region: string) => new KMS({ region, credentials })

/* Specify a multi-Region key in us-west-2 */
const multiRegionUsWestKey =
  'arn:aws:kms:us-west-2:111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab'

/* Instantiate the keyring */
const mrkDecryptKeyring = buildAwsKmsMrkAwareStrictMultiKeyringBrowser({
  generatorKeyId: multiRegionUsWestKey,
  clientProvider,
})

/* Decrypt the data */
const { plaintext, messageHeader } = await decrypt(mrkDecryptKeyring, result)
```

JavaScript Node.js

Um im strikten Modus zu entschlüsseln, verwenden Sie die `buildAwsKmsMrkAwareStrictMultiKeyringNode()` Methode, um den Schlüsselbund zu erstellen, und geben Sie den zugehörigen Schlüssel für mehrere Regionen in der lokalen Region (us-west-2) an.

Ein vollständiges Beispiel finden Sie unter [kms_multi_region_simple.ts](#) im Repository unter `AWS-Verschlüsselungs-SDK for JavaScript` GitHub

```
/* Decrypt with a related multi-Region KMS key in us-west-2 Region */

import { buildClient } from '@aws-crypto/client-node'

/* Instantiate the client
const { decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

/* Multi-Region keys have a distinctive key ID that begins with 'mrk'
 * Specify a multi-Region key in us-west-2
 */
const multiRegionUsWestKey =
  'arn:aws:kms:us-west-2:111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab'

/* Create an AWS KMS keyring */
const mrkDecryptKeyring = buildAwsKmsMrkAwareStrictMultiKeyringNode({
  generatorKeyId: multiRegionUsWestKey,
})

/* Decrypt your ciphertext */
const { plaintext, messageHeader } = await decrypt(decryptKeyring, result)
```

Python

Um im strikten Modus zu entschlüsseln, verwenden Sie die Methode, um den Hauptschlüsselanbieter zu erstellen. `MRKAwareStrictAwsKmsMasterKeyProvider()` Geben Sie den zugehörigen Schlüssel für mehrere Regionen in der lokalen Region (us-west-2) an.

Ein vollständiges Beispiel finden Sie unter [mrk_aware_kms_provider.py](#) im `AWS-Verschlüsselungs-SDK for Python` Repository unter `GitHub`


```
# Decrypt with a related multi-Region KMS key in us-west-2 Region

# Instantiate the client
client =
  aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_R

# Related multi-Region keys have the same key ID. Their key ARNs differs only in the
  Region field
mrk_us_west_2 = "arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab"

# Use the multi-Region method to create the master key provider
# in strict mode
strict_mrk_key_provider = MRKAwareStrictAwsKmsMasterKeyProvider(
    key_ids=[mrk_us_west_2]
)

# Decrypt your ciphertext
plaintext, _ = client.decrypt(
    source=ciphertext,
    key_provider=strict_mrk_key_provider
)
```

Sie können auch im Discovery-Modus mit Schlüsseln für AWS KMS mehrere Regionen entschlüsseln. Beim Entschlüsseln im Discovery-Modus geben Sie keine an. AWS KMS keys (Informationen zu Schlüsselanhängern für die AWS KMS Erkennung einzelner Regionen finden Sie unter.) [Verwenden Sie einen Discovery-Schlüsselbund AWS KMS](#)

Wenn Sie mit einem Schlüssel für mehrere Regionen verschlüsselt haben, versucht das Symbol für mehrere Regionen im Erkennungsmodus, die Entschlüsselung mithilfe eines zugehörigen Regionsschlüssels in der lokalen Region durchzuführen. Wenn keine vorhanden ist, schlägt der Anruf fehl. Im Erkennungsmodus versucht der nicht, den Schlüssel für mehrere Regionen, der für die Verschlüsselung verwendet AWS Encryption SDK wird, regionsübergreifend aufzurufen.

Note

Wenn Sie im Erkennungsmodus ein Symbol für mehrere Regionen verwenden, um Daten zu verschlüsseln, schlägt der Verschlüsselungsvorgang fehl.

Das folgende Beispiel zeigt, wie im Erkennungsmodus mit dem Symbol für mehrere Regionen entschlüsselt wird. Da Sie kein angeben AWS KMS key, AWS Encryption SDK muss die Region aus einer anderen Quelle abgerufen werden. Wenn möglich, geben Sie die lokale Region explizit an. Andernfalls AWS Encryption SDK ruft der die lokale Region aus der Region ab, die AWS SDK in Ihrer Programmiersprache konfiguriert ist.

Bevor Sie diese Beispiele ausführen, ersetzen Sie die Beispielkonto-ID und den Schlüssel für mehrere Regionen ARN durch gültige Werte aus Ihrem AWS-Konto.

C

Um im Discovery-Modus mit einem Schlüssel für mehrere Regionen zu entschlüsseln, verwenden Sie die `Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder()` Methode, um den Schlüsselbund zu erstellen, und die Methode, um den `Aws::Cryptosdk::KmsKeyring::DiscoveryFilter::Builder()` Discovery-Filter zu erstellen. Um die lokale Region anzugeben, definieren Sie a `ClientConfiguration` und geben Sie sie im Client an. AWS KMS

Ein vollständiges Beispiel finden Sie unter [kms_multi_region_keys.cpp](#) im AWS-Verschlüsselungs-SDK for C Repository unter GitHub.

```

/* Decrypt in discovery mode with a multi-Region KMS key */

/* Load error strings for debugging */
aws_cryptosdk_load_error_strings();

/* Construct a discovery filter for the account and partition. The
 * filter is optional, but it's a best practice that we recommend.
 */
const char *account_id = "111122223333";
const char *partition = "aws";
const std::shared_ptr<Aws::Cryptosdk::KmsKeyring::DiscoveryFilter> discovery_filter
=

    Aws::Cryptosdk::KmsKeyring::DiscoveryFilter::Builder(partition).AddAccount(account_id).Build();

/* Create an AWS KMS client in the desired region. */
const char *region = "us-west-2";

Aws::Client::ClientConfiguration client_config;
client_config.region = region;
const std::shared_ptr<Aws::KMS::KMSClient> kms_client =

```

```

    Aws::MakeShared<Aws::KMS::KMSClient>("AWS_SAMPLE_CODE", client_config);

struct aws_cryptosdk_keyring *mrk_keyring =
    Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder()
        .WithKmsClient(kms_client)
        .BuildDiscovery(region, discovery_filter);

/* Create a session; release the keyring */
struct aws_cryptosdk_session *session =
    aws_cryptosdk_session_new_from_keyring_2(aws_default_allocator(),
    AWS_CRYPTOSDK_DECRYPT, mrk_keyring);

aws_cryptosdk_keyring_release(mrk_keyring);
commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
/* Decrypt the ciphertext
 * aws_cryptosdk_session_process_full is designed for non-streaming data
 */
aws_cryptosdk_session_process_full(
    session, plaintext, plaintext_buf_sz, &plaintext_len, ciphertext,
    ciphertext_len));

/* Clean up the session */
aws_cryptosdk_session_destroy(session);

```

C# / .NET

Um einen Discovery-Schlüsselbund für mehrere Regionen im Formular zu erstellen. AWS Encryption SDK NET, instanzieren Sie ein `CreateAwsKmsMrkDiscoveryKeyringInput` Objekt, das einen AWS KMS Client für ein bestimmtes Objekt verwendet AWS-Region, und einen optionalen Erkennungsfiler, der KMS Schlüssel auf eine bestimmte Partition und ein bestimmtes Konto beschränkt. AWS Rufen Sie dann die `CreateAwsKmsMrkDiscoveryKeyring()` Methode mit dem Eingabeobjekt auf. Ein vollständiges Beispiel finden Sie unter [AwsKmsMrkDiscoveryKeyringExample.cs](#) im Forum AWS Encryption SDK . NETRepository aktiviert. GitHub

Um einen Schlüsselbund für mehrere Regionen zu erstellen, verwenden Sie die Methode, um einen Schlüsselbund für mehrere Regionen zu erstellen AWS-Region, oder verwenden Sie die `CreateAwsKmsMrkDiscoveryMultiKeyring()` Methode, um mehrere Schlüsselbunde für die Erkennung mehrerer Regionen `CreateAwsKmsMrkDiscoveryKeyring()` zu erstellen, und kombinieren Sie sie dann mit der Methode zu einem Schlüsselbund mit mehreren Schlüsselbunden. `CreateMultiKeyring()`

Ein [AwsKmsMrkDiscoveryMultiKeyringExampleBeispiel](#) finden Sie unter `.cs`.

```
// Decrypt in discovery mode with a multi-Region KMS key

// Instantiate the AWS Encryption SDK and material providers
var encryptionSdk = AwsEncryptionSdkFactory.CreateDefaultAwsEncryptionSdk();
var materialProviders =

    AwsCryptographicMaterialProvidersFactory.CreateDefaultAwsCryptographicMaterialProviders();

List<string> account = new List<string> { "111122223333" };

// Instantiate the discovery filter
DiscoveryFilter mrkDiscoveryFilter = new DiscoveryFilter()
{
    AccountIds = account,
    Partition = "aws"
}

// Create the keyring
var createMrkDiscoveryKeyringInput = new CreateAwsKmsMrkDiscoveryKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(RegionEndpoint.USWest2),
    DiscoveryFilter = mrkDiscoveryFilter
};
var mrkDiscoveryKeyring =
    materialProviders.CreateAwsKmsMrkDiscoveryKeyring(createMrkDiscoveryKeyringInput);

// Decrypt the ciphertext
var decryptInput = new DecryptInput
{
    Ciphertext = ciphertext,
    Keyring = mrkDiscoveryKeyring
};
var decryptOutput = encryptionSdk.Decrypt(decryptInput);
```

AWS Encryption CLI

Verwenden Sie das `Discovery`-Attribut des Parameters, um im `Discovery`-Modus zu entschlüsseln. `--wrapping-keys` Die Attribute `discovery-account` und `discovery-partition` erstellen einen `Discovery-Filter`, der optional, aber empfohlen wird.

Um die Region anzugeben, enthält dieser Befehl das Regionsattribut des Parameters. --wrapping-keys

```
# Decrypt in discovery mode with a multi-Region KMS key

$ aws-encryption-cli --decrypt \
    --input hello.txt.encrypted \
    --wrapping-keys discovery=true \
        discovery-account=111122223333 \
        discovery-partition=aws \
        region=us-west-2 \
    --encryption-context purpose=test \
    --metadata-output ~/metadata \
    --max-encrypted-data-keys 1 \
    --buffer \
    --output .
```

Java

Verwenden Sie den `builder().withDiscoveryMrkRegion` Parameter, um die lokale Region anzugeben. Andernfalls AWS Encryption SDK ruft der die lokale Region aus der Region ab, die in der konfiguriert ist [AWS SDK for Java](#).

Ein vollständiges Beispiel finden Sie unter [DiscoveryMultiRegionDecryptionExample.java](#) im AWS-Verschlüsselungs-SDK for Java Repository unter GitHub.

```
// Decrypt in discovery mode with a multi-Region KMS key

// Instantiate the client
final AwsCrypto crypto = AwsCrypto.builder()
    .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
    .build();

DiscoveryFilter discoveryFilter = new DiscoveryFilter("aws", 111122223333);

AwsKmsMrkAwareMasterKeyProvider mrkDiscoveryProvider =
    AwsKmsMrkAwareMasterKeyProvider
        .builder()
        .withDiscoveryMrkRegion(Region.US_WEST_2)
        .buildDiscovery(discoveryFilter);

// Decrypt your ciphertext
final CryptoResult<byte[], AwsKmsMrkAwareMasterKey> decryptResult = crypto
```

```
.decryptData(mrkDiscoveryProvider, ciphertext);
```

JavaScript Browser

Verwenden Sie die Methode, um im Discovery-Modus mit einem symmetrischen Schlüssel für mehrere Regionen zu entschlüsseln.

```
AwsKmsMrkAwareSymmetricDiscoveryKeyringBrowser()
```

Ein vollständiges Beispiel finden Sie unter [kms_multi_region_discovery.ts](#) im Repository unter [AWS-Verschlüsselungs-SDK for JavaScript GitHub](#)

```
/* Decrypt in discovery mode with a multi-Region KMS key */

import {
  AwsKmsMrkAwareSymmetricDiscoveryKeyringBrowser,
  buildClient,
  CommitmentPolicy,
  KMS,
} from '@aws-crypto/client-browser'

/* Instantiate an AWS Encryption SDK client */
const { decrypt } = buildClient()

declare const credentials: {
  accessKeyId: string
  secretAccessKey: string
  sessionToken: string
}

/* Instantiate the KMS client with an explicit Region */
const client = new KMS({ region: 'us-west-2', credentials })

/* Create a discovery filter */
const discoveryFilter = { partition: 'aws', accountIDs: ['111122223333'] }

/* Create an AWS KMS discovery keyring */
const mrkDiscoveryKeyring = new AwsKmsMrkAwareSymmetricDiscoveryKeyringBrowser({
  client,
  discoveryFilter,
})
```

```
/* Decrypt the data */
const { plaintext, messageHeader } = await decrypt(mrkDiscoveryKeyring, ciphertext)
```

JavaScript Node.js

Verwenden Sie die Methode, um im Discovery-Modus mit einem symmetrischen Schlüssel für mehrere Regionen zu entschlüsseln.

`AwsKmsMrkAwareSymmetricDiscoveryKeyringNode()`

Ein vollständiges Beispiel finden Sie unter [kms_multi_region_discovery.ts](#) im Repository unter [AWS-Verschlüsselungs-SDK for JavaScript GitHub](#)

```
/* Decrypt in discovery mode with a multi-Region KMS key */

import {
  AwsKmsMrkAwareSymmetricDiscoveryKeyringNode,
  buildClient,
  CommitmentPolicy,
  KMS,
} from '@aws-crypto/client-node'

/* Instantiate the Encryption SDK client
const { decrypt } = buildClient()

/* Instantiate the KMS client with an explicit Region */
const client = new KMS({ region: 'us-west-2' })

/* Create a discovery filter */
const discoveryFilter = { partition: 'aws', accountIDs: ['111122223333'] }

/* Create an AWS KMS discovery keyring */
const mrkDiscoveryKeyring = new AwsKmsMrkAwareSymmetricDiscoveryKeyringNode({
  client,
  discoveryFilter,
})

/* Decrypt your ciphertext */
const { plaintext, messageHeader } = await decrypt(mrkDiscoveryKeyring, result)
```

Python

Verwenden Sie die Methode, um im Discovery-Modus mit einem Schlüssel für mehrere Regionen zu entschlüsseln. `MRKAwareDiscoveryAwsKmsMasterKeyProvider()`

Ein vollständiges Beispiel finden Sie unter [mrk_aware_kms_provider.py](#) im AWS-Verschlüsselungs-SDK for Python Repository unter. GitHub

```
# Decrypt in discovery mode with a multi-Region KMS key

# Instantiate the client
client = aws_encryption_sdk.EncryptionSDKClient()

# Create the discovery filter and specify the region
decrypt_kwargs = dict(
    discovery_filter=DiscoveryFilter(account_ids="111122223333",
    partition="aws"),
    discovery_region="us-west-2",
)

# Use the multi-Region method to create the master key provider
# in discovery mode
mrk_discovery_key_provider =
    MRKAwareDiscoveryAwsKmsMasterKeyProvider(**decrypt_kwargs)

# Decrypt your ciphertext
plaintext, _ = client.decrypt(
    source=ciphertext,
    key_provider=mrk_discovery_key_provider
)
```

Auswahl einer Algorithmus-Suite

Die AWS Encryption SDK unterstützt mehrere [symmetrische und asymmetrische Verschlüsselungsalgorithmen zur Verschlüsselung](#) Ihrer Datenschlüssel unter den von Ihnen angegebenen Wrapping-Schlüsseln. [Wenn diese Datenschlüssel jedoch zur Verschlüsselung Ihrer Daten verwendet werden, wird AWS Encryption SDK standardmäßig eine empfohlene Algorithmusuite verwendet, die den GCM Algorithmus AES - mit Schlüsselableitung, digitalen Signaturen und Schlüsselbindung verwendet.](#) Obwohl die Standard-Algorithmus-Suite wahrscheinlich für die meisten Anwendungen geeignet ist, können Sie auch eine alternative Algorithmusuite

wählen. Einige Vertrauensmodelle würden beispielsweise durch eine Algorithmus-Suite ohne [digitale Signaturen](#) erfüllt. Hinweise zu den von der AWS Encryption SDK unterstützten Algorithmus-Suites finden Sie unter [Unterstützte Algorithmen-Pakete im AWS Encryption SDK](#).

Die folgenden Beispiele zeigen Ihnen, wie Sie beim Verschlüsseln eine alternative Algorithmus-Suite auswählen. In diesen Beispielen wird eine empfohlene AES GCM Algorithmus-Suite mit Schlüsselableitung und Schlüsselzusage ausgewählt, jedoch ohne digitale Signaturen. Wenn Sie mit einer Algorithmus-Suite verschlüsseln, die keine digitalen Signaturen enthält, verwenden Sie beim Entschlüsseln den Entschlüsselungsmodus nur ohne Vorzeichen. Dieser Modus schlägt fehl, wenn er auf einen signierten Chiffretext trifft, und ist vor allem bei der Streaming-Entschlüsselung nützlich.

C

Um eine alternative Algorithmus-Suite in der anzugeben AWS-Verschlüsselungs-SDK for C, müssen Sie explizit eine erstellen. CMM Verwenden Sie dann die `aws_cryptosdk_default_cmm_set_alg_id` mit der CMM und der ausgewählten Algorithmus-Suite.

```
/* Specify an algorithm suite without signing */

/* Load error strings for debugging */
aws_cryptosdk_load_error_strings();

/* Construct an AWS KMS keyring */
struct aws_cryptosdk_keyring *kms_keyring =
    Aws::Cryptosdk::KmsKeyring::Builder().Build(key_arn);

/* To set an alternate algorithm suite, create an cryptographic
   materials manager (CMM) explicitly
   */
struct aws_cryptosdk_cmm *cmm =
    aws_cryptosdk_default_cmm_new(aws_default_allocator(), kms_keyring);
aws_cryptosdk_keyring_release(kms_keyring);

/* Specify the algorithm suite for the CMM */
aws_cryptosdk_default_cmm_set_alg_id(cmm, ALG_AES256_GCM_HKDF_SHA512_COMMIT_KEY);

/* Construct the session with the CMM,
   then release the CMM reference
   */
struct aws_cryptosdk_session *session = aws_cryptosdk_session_new_from_cmm_2(alloc,
    AWS_CRYPTOSDK_ENCRYPT, cmm);
```

```

aws_cryptosdk_cmm_release(cmm);

/* Encrypt the data
   Use aws_cryptosdk_session_process_full with non-streaming data
*/
if (AWS_OP_SUCCESS != aws_cryptosdk_session_process_full(
    session,
    ciphertext,
    ciphertext_buf_sz,
    &ciphertext_len,
    plaintext,
    plaintext_len)) {
    aws_cryptosdk_session_destroy(session);
    return AWS_OP_ERR;
}

```

Verwenden `AWS_CRYPTOSDK_DECRYPT_UNSIGNED` Sie beim Entschlüsseln von Daten, die ohne digitale Signaturen verschlüsselt wurden. Dies führt dazu, dass die Entschlüsselung fehlschlägt, wenn signierter Chiffretext gefunden wird.

```

/* Decrypt unsigned streaming data */

/* Load error strings for debugging */
aws_cryptosdk_load_error_strings();

/* Construct an AWS KMS keyring */
struct aws_cryptosdk_keyring *kms_keyring =
    Aws::Cryptosdk::KmsKeyring::Builder().Build(key_arn);

/* Create a session for decrypting with the AWS KMS keyring
   Then release the keyring reference
*/
struct aws_cryptosdk_session *session =

    aws_cryptosdk_session_new_from_keyring_2(alloc, AWS_CRYPTOSDK_DECRYPT_UNSIGNED,
    kms_keyring);
aws_cryptosdk_keyring_release(kms_keyring);

if (!session) {
    return AWS_OP_ERR;
}

/* Limit encrypted data keys */

```

```

aws_cryptosdk_session_set_max_encrypted_data_keys(session, 1);

/* Decrypt
   Use aws_cryptosdk_session_process_full with non-streaming data
*/
if (AWS_OP_SUCCESS != aws_cryptosdk_session_process_full(
    session,
    plaintext,
    plaintext_buf_sz,
    &plaintext_len,
    ciphertext,
    ciphertext_len)) {
    aws_cryptosdk_session_destroy(session);
    return AWS_OP_ERR;
}

```

C# / .NET

Um eine alternative Algorithmus-Suite im Formular anzugeben. AWS Encryption SDK NET, geben Sie die `AlgorithmSuiteId` Eigenschaft eines [EncryptInput](#) Objekts an. Das AWS Encryption SDK für .NET enthält [Konstanten](#), anhand derer Sie Ihre bevorzugte Algorithmensuite identifizieren können.

Das AWS Encryption SDK für .NET verfügt nicht über eine Methode zur Erkennung von signiertem Chiffretext bei der Streaming-Entschlüsselung, da diese Bibliothek keine Streaming-Daten unterstützt.

```

// Specify an algorithm suite without signing

// Instantiate the AWS Encryption SDK and material providers
var encryptionSdk = AwsEncryptionSdkFactory.CreateDefaultAwsEncryptionSdk();
var materialProviders =

    AwsCryptographicMaterialProvidersFactory.CreateDefaultAwsCryptographicMaterialProviders();

// Create the keyring
var keyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = keyArn
};
var keyring = materialProviders.CreateAwsKmsKeyring(keyringInput);

```

```
// Encrypt your plaintext data
var encryptInput = new EncryptInput
{
    Plaintext = plaintext,
    Keyring = keyring,
    AlgorithmSuiteId = AlgorithmSuiteId.ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY
};
var encryptOutput = encryptionSdk.Encrypt(encryptInput);
```

AWS Encryption CLI

Bei der Verschlüsselung der `hello.txt` Datei verwendet dieses Beispiel den `--algorithm` Parameter, um eine Algorithmussuite ohne digitale Signaturen anzugeben.

```
# Specify an algorithm suite without signing

# To run this example, replace the fictitious key ARN with a valid value.
$ keyArn=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

$ aws-encryption-cli --encrypt \
    --input hello.txt \
    --wrapping-keys key=$keyArn \
    --algorithm AES_256_GCM_HKDF_SHA512_COMMIT_KEY \
    --metadata-output ~/metadata \
    --encryption-context purpose=test \
    --commitment-policy require-encrypt-require-decrypt \
    --output hello.txt.encrypted \
    --decode
```

Bei der Entschlüsselung verwendet dieses Beispiel den `--decrypt-unsigned` Parameter. Dieser Parameter wird empfohlen, um sicherzustellen, dass Sie unsignierten Chiffretext entschlüsseln, insbesondere bei der, bei der immer ein- und CLI ausgegebene Daten gestreamt werden.

```
# Decrypt unsigned streaming data

# To run this example, replace the fictitious key ARN with a valid value.
$ keyArn=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

$ aws-encryption-cli --decrypt-unsigned \
    --input hello.txt.encrypted \
    --wrapping-keys key=$keyArn \
```

```
--max-encrypted-data-keys 1 \  
--commitment-policy require-encrypt-require-decrypt \  
--encryption-context purpose=test \  
--metadata-output ~/metadata \  
--output .
```

Java

Verwenden Sie die Methode, um eine alternative Algorithmus-Suite anzugeben.

`AwsCrypto.builder().withEncryptionAlgorithm()` Dieses Beispiel spezifiziert eine alternative Algorithmussuite ohne digitale Signaturen.

```
// Specify an algorithm suite without signing  
  
// Instantiate the client  
AwsCrypto crypto = AwsCrypto.builder()  
    .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)  
    .withEncryptionAlgorithm(CryptoAlgorithm.ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY)  
    .build();  
  
String awsKmsKey = "arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";  
  
// Create a master key provider in strict mode  
KmsMasterKeyProvider masterKeyProvider = KmsMasterKeyProvider.builder()  
    .buildStrict(awsKmsKey);  
  
// Create an encryption context to identify this ciphertext  
Map<String, String> encryptionContext = Collections.singletonMap("Example",  
    "FileStreaming");  
  
// Encrypt your plaintext data  
CryptoResult<byte[], KmsMasterKey> encryptResult = crypto.encryptData(  
    masterKeyProvider,  
    sourcePlaintext,  
    encryptionContext);  
byte[] ciphertext = encryptResult.getResult();
```

Verwenden Sie beim Streamen von Daten zur Entschlüsselung diese

`createUnsignedMessageDecryptingStream()` Methode, um sicherzustellen, dass der gesamte Chiffretext, den Sie entschlüsseln, nicht signiert ist.

```
// Decrypt unsigned streaming data

// Instantiate the client
AwsCrypto crypto = AwsCrypto.builder()
    .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
    .withMaxEncryptedDataKeys(1)
    .build();

// Create a master key provider in strict mode
String awsKmsKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
KmsMasterKeyProvider masterKeyProvider = KmsMasterKeyProvider.builder()
    .buildStrict(awsKmsKey);

// Decrypt the encrypted message
FileInputStream in = new FileInputStream(srcFile + ".encrypted");
CryptoInputStream<KmsMasterKey> decryptingStream =
    crypto.createUnsignedMessageDecryptingStream(masterKeyProvider, in);

// Return the plaintext data
// Write the plaintext data to disk
FileOutputStream out = new FileOutputStream(srcFile + ".decrypted");
IOUtils.copy(decryptingStream, out);
decryptingStream.close();
```

JavaScript Browser

Um eine alternative Algorithmussuite anzugeben, verwenden Sie den `suiteId` Parameter mit einem Enum-Wert. `AlgorithmSuiteIdentifier`

```
// Specify an algorithm suite without signing

// Instantiate the client
const { encrypt } = buildClient( CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT )

// Specify a KMS key
const generatorKeyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

// Create a keyring with the KMS key
const keyring = new KmsKeyringBrowser({ generatorKeyId })

// Encrypt your plaintext data
```

```
const { result } = await encrypt(keyring, cleartext, { suiteId:  
AlgorithmSuiteIdentifier.ALG_AES256_GCM_IV12_TAG16_HKDF_SHA512_COMMIT_KEY,  
encryptionContext: context, })
```

Verwenden Sie beim Entschlüsseln die `decrypt` Standardmethode. AWS-Verschlüsselungs-SDK for JavaScript hat im Browser keinen `decrypt-unsigned` Modus, weil der Browser kein Streaming unterstützt.

```
// Decrypt unsigned streaming data  
  
// Instantiate the client  
const { decrypt } = buildClient( CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT )  
  
// Create a keyring with the same KMS key used to encrypt  
const generatorKeyId = "arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";  
const keyring = new KmsKeyringBrowser({ generatorKeyId })  
  
// Decrypt the encrypted message  
const { plaintext, messageHeader } = await decrypt(keyring, ciphertextMessage)
```

JavaScript Node.js

Um eine alternative Algorithmus-Suite anzugeben, verwenden Sie den `suiteId` Parameter mit einem `AlgorithmSuiteIdentifier` Enum-Wert.

```
// Specify an algorithm suite without signing  
  
// Instantiate the client  
const { encrypt } = buildClient( CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT )  
  
// Specify a KMS key  
const generatorKeyId = "arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";  
  
// Create a keyring with the KMS key  
const keyring = new KmsKeyringNode({ generatorKeyId })  
  
// Encrypt your plaintext data  
const { result } = await encrypt(keyring, cleartext, { suiteId:  
AlgorithmSuiteIdentifier.ALG_AES256_GCM_IV12_TAG16_HKDF_SHA512_COMMIT_KEY,  
encryptionContext: context, })
```

Verwenden `decryptUnsignedMessage` Sie Stream, wenn Sie Daten entschlüsseln, die ohne digitale Signaturen verschlüsselt wurden. Diese Methode schlägt fehl, wenn sie auf signierten Chiffretext trifft.

```
// Decrypt unsigned streaming data

// Instantiate the client
const { decryptUnsignedMessageStream } =
  buildClient( CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT )

// Create a keyring with the same KMS key used to encrypt
const generatorKeyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
const keyring = new KmsKeyringNode({ generatorKeyId })

// Decrypt the encrypted message
const outputStream =
  createReadStream(filename) .pipe(decryptUnsignedMessageStream(keyring))
```

Python

Um einen alternativen Verschlüsselungsalgorithmus anzugeben, verwenden Sie den `algorithm` Parameter mit einem `Algorithm Enum`-Wert.

```
# Specify an algorithm suite without signing

# Instantiate a client
client =
  aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT,
                                         max_encrypted_data_keys=1)

# Create a master key provider in strict mode
aws_kms_key = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
aws_kms_strict_master_key_provider = StrictAwsKmsMasterKeyProvider(
  key_ids=[aws_kms_key]
)

# Encrypt the plaintext using an alternate algorithm suite
ciphertext, encrypted_message_header = client.encrypt(
  algorithm=Algorithm.AES_256_GCM_HKDF_SHA512_COMMIT_KEY, source=source_plaintext,
  key_provider=kms_key_provider
```



```
)
```

Verwenden Sie beim Entschlüsseln von Nachrichten, die ohne digitale Signaturen verschlüsselt wurden, den `decrypt-unsigned` Streaming-Modus, insbesondere beim Entschlüsseln während des Streamings.

```
# Decrypt unsigned streaming data

# Instantiate the client
client =
    aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_R
                                         max_encrypted_data_keys=1)

# Create a master key provider in strict mode
aws_kms_key = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
aws_kms_strict_master_key_provider = StrictAwsKmsMasterKeyProvider(
    key_ids=[aws_kms_key]
)

# Decrypt with decrypt-unsigned
with open(ciphertext_filename, "rb") as ciphertext, open(cycled_plaintext_filename,
"wb") as plaintext:
    with client.stream(mode="decrypt-unsigned",
                       source=ciphertext,
                       key_provider=master_key_provider) as decryptor:
        for chunk in decryptor:
            plaintext.write(chunk)

# Verify that the encryption context
assert all(
    pair in decryptor.header.encryption_context.items() for pair in
    encryptor.header.encryption_context.items()
)
return ciphertext_filename, cycled_plaintext_filename
```

Beschränkung verschlüsselter Datenschlüssel

Sie können die Anzahl der verschlüsselten Datenschlüssel in einer verschlüsselten Nachricht einschränken. Diese bewährte Methode kann Ihnen helfen, einen falsch konfigurierten Schlüsselbund beim Verschlüsseln oder einen böartigen Chiffretext beim Entschlüsseln zu erkennen. Es verhindert

auch unnötige, teure und potenziell erschöpfende Zugriffe auf Ihre Schlüsselinfrastruktur. Die Einschränkung verschlüsselter Datenschlüssel ist am wertvollsten, wenn Sie Nachrichten aus einer nicht vertrauenswürdigen Quelle entschlüsseln.

Obwohl die meisten verschlüsselten Nachrichten einen verschlüsselten Datenschlüssel für jeden bei der Verschlüsselung verwendeten Umschließungsschlüssel haben, kann eine verschlüsselte Nachricht bis zu 65.535 verschlüsselte Datenschlüssel enthalten. Ein böswilliger Akteur könnte eine verschlüsselte Nachricht mit Tausenden von verschlüsselten Datenschlüsseln erstellen, von denen keiner entschlüsselt werden kann. In der Folge AWS Encryption SDK würde versucht werden, jeden verschlüsselten Datenschlüssel zu entschlüsseln, bis alle verschlüsselten Datenschlüssel in der Nachricht aufgebraucht sind.

Verwenden Sie den `MaxEncryptedDataKeys` Parameter, um die Anzahl der verschlüsselten Datenschlüssel einzuschränken. Dieser Parameter ist ab Version 1.9 für alle unterstützten Programmiersprachen verfügbar. x und 2.2. x der AWS Encryption SDK. Es ist optional und gültig beim Verschlüsseln und Entschlüsseln. In den folgenden Beispielen werden Daten entschlüsselt, die mit drei verschiedenen Wrapping-Schlüsseln verschlüsselt wurden. Der `MaxEncryptedDataKeys` Wert ist auf 3 festgelegt.

C

```
/* Load error strings for debugging */
aws_cryptosdk_load_error_strings();

/* Construct an AWS KMS keyring */
struct aws_cryptosdk_keyring *kms_keyring =
    Aws::Cryptosdk::KmsKeyring::Builder().Build(key_arn1, { key_arn2, key_arn3 });

/* Create a session */
struct aws_cryptosdk_session *session =
    aws_cryptosdk_session_new_from_keyring_2(alloc, AWS_CRYPTOSDK_DECRYPT,
    kms_keyring);
aws_cryptosdk_keyring_release(kms_keyring);

/* Limit encrypted data keys */
aws_cryptosdk_session_set_max_encrypted_data_keys(session, 3);

/* Decrypt */
size_t ciphertext_consumed_output;
aws_cryptosdk_session_process(session,
    plaintext_output,
```

```

    plaintext_buf_sz_output,
    &plaintext_len_output,
    ciphertext_input,
    ciphertext_len_input,
    &ciphertext_consumed_output);
assert(aws_cryptosdk_session_is_done(session));
assert(ciphertext_consumed == ciphertext_len);

```

C# / .NET

Um verschlüsselte Datenschlüssel im Formular AWS Encryption SDK einzuschränken. NET, instanzieren Sie einen Client für das AWS Encryption SDK for .NET und setzen Sie seinen optionalen `MaxEncryptedDataKeys` Parameter auf den gewünschten Wert. Rufen Sie dann die `Decrypt()` Methode auf der konfigurierten AWS Encryption SDK Instanz auf.

```

// Decrypt with limited data keys

// Instantiate the material providers
var materialProviders =

    AwsCryptographicMaterialProvidersFactory.CreateDefaultAwsCryptographicMaterialProviders();

// Configure the commitment policy on the AWS Encryption SDK instance
var config = new AwsEncryptionSdkConfig
{
    MaxEncryptedDataKeys = 3
};
var encryptionSdk = AwsEncryptionSdkFactory.CreateAwsEncryptionSdk(config);

// Create the keyring
string keyArn = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
var createKeyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = keyArn
};
var decryptKeyring = materialProviders.CreateAwsKmsKeyring(createKeyringInput);

// Decrypt the ciphertext
var decryptInput = new DecryptInput
{
    Ciphertext = ciphertext,

```

```

    Keyring = decryptKeyring
};
var decryptOutput = encryptionSdk.Decrypt(decryptInput);

```

AWS Encryption CLI

```

# Decrypt with limited encrypted data keys

$ aws-encryption-cli --decrypt \
  --input hello.txt.encrypted \
  --wrapping-keys key=$key_arn1 key=$key_arn2 key=$key_arn3 \
  --buffer \
  --max-encrypted-data-keys 3 \
  --encryption-context purpose=test \
  --metadata-output ~/metadata \
  --output .

```

Java

```

// Construct a client with limited encrypted data keys
final AwsCrypto crypto = AwsCrypto.builder()
  .withMaxEncryptedDataKeys(3)
  .build();

// Create an AWS KMS master key provider
final KmsMasterKeyProvider keyProvider = KmsMasterKeyProvider.builder()
  .buildStrict(keyArn1, keyArn2, keyArn3);

// Decrypt
final CryptoResult<byte[], KmsMasterKey> decryptResult =
  crypto.decryptData(keyProvider, ciphertext)

```

JavaScript Browser

```

// Construct a client with limited encrypted data keys
const { encrypt, decrypt } = buildClient({ maxEncryptedDataKeys: 3 })

declare const credentials: {
  accessKeyId: string
  secretAccessKey: string
  sessionToken: string
}

```

```
const clientProvider = getClient(KMS, {
  credentials: { accessKeyId, secretAccessKey, sessionToken }
})

// Create an AWS KMS keyring
const keyring = new KmsKeyringBrowser({
  clientProvider,
  keyIds: [keyArn1, keyArn2, keyArn3],
})

// Decrypt
const { plaintext, messageHeader } = await decrypt(keyring, ciphertext)
```

JavaScript Node.js

```
// Construct a client with limited encrypted data keys
const { encrypt, decrypt } = buildClient({ maxEncryptedDataKeys: 3 })

// Create an AWS KMS keyring
const keyring = new KmsKeyringBrowser({
  keyIds: [keyArn1, keyArn2, keyArn3],
})

// Decrypt
const { plaintext, messageHeader } = await decrypt(keyring, ciphertext)
```

Python

```
# Instantiate a client with limited encrypted data keys
client = aws_encryption_sdk.EncryptionSDKClient(max_encrypted_data_keys=3)

# Create an AWS KMS master key provider
master_key_provider = aws_encryption_sdk.StrictAwsKmsMasterKeyProvider(
    key_ids=[key_arn1, key_arn2, key_arn3])

# Decrypt
plaintext, header = client.decrypt(source=ciphertext,
    key_provider=master_key_provider)
```

Einen Discovery-Filter erstellen

Beim Entschlüsseln von Daten, die mit KMS Schlüsseln verschlüsselt wurden, empfiehlt es sich, im strikten Modus zu entschlüsseln, d. h., die verwendeten Wrapping-Schlüssel auf die von Ihnen angegebenen zu beschränken. Bei Bedarf können Sie jedoch auch im Discovery-Modus entschlüsseln, in dem Sie keine Umschließungsschlüssel angeben. In diesem Modus AWS KMS können Sie den verschlüsselten Datenschlüssel mit dem Schlüssel entschlüsseln, mit dem er verschlüsselt wurde, unabhängig davon, wem dieser KMS Schlüssel gehört oder wer Zugriff darauf KMS hat.

[Wenn Sie im Discovery-Modus entschlüsseln müssen, empfehlen wir, immer einen Discovery-Filter zu verwenden, der die Schlüssel, die verwendet werden können, auf die KMS Schlüssel beschränkt, die sich in einer bestimmten Partition befinden AWS-Konto](#) . Der Discovery-Filter ist optional, hat sich aber bewährt.

Verwenden Sie die folgende Tabelle, um den Partitionswert für Ihren Discovery-Filter zu ermitteln.

Region	Partition
AWS-Regionen	aws
Regionen in China	aws-cn
AWS GovCloud (US) Regions	aws-us-gov

Die Beispiele in diesem Abschnitt zeigen, wie Sie einen Discovery-Filter erstellen. Bevor Sie den Code verwenden, ersetzen Sie die Beispielwerte durch gültige Werte für die Partition AWS-Konto und.

C

Vollständige Beispiele finden Sie in der [Datei kms_discovery.cpp](#) in der AWS-Verschlüsselungs-SDK for C.

```
/* Create a discovery filter for an AWS account and partition */  
  
const char *account_id = "111122223333";  
const char *partition = "aws";
```

```
const std::shared_ptr<Aws::Cryptosdk::KmsKeyring::DiscoveryFilter> discovery_filter
=
    Aws::Cryptosdk::KmsKeyring::DiscoveryFilter::Builder(partition).AddAccount(account_id).Build
```

C# / .NET

Ein vollständiges Beispiel finden Sie unter [DiscoveryFilterExample.cs](#) im Forum AWS Encryption SDK .NET.

```
// Create a discovery filter for an AWS account and partition

List<string> account = new List<string> { "111122223333" };

DiscoveryFilter exampleDiscoveryFilter = new DiscoveryFilter()
{
    AccountIds = account,
    Partition = "aws"
}
```

AWS Encryption CLI

```
# Decrypt in discovery mode with a discovery filter

$ aws-encryption-cli --decrypt \
    --input hello.txt.encrypted \
    --wrapping-keys discovery=true \
        discovery-account=111122223333 \
        discovery-partition=aws \
    --encryption-context purpose=test \
    --metadata-output ~/metadata \
    --max-encrypted-data-keys 1 \
    --buffer \
    --output .
```

Java

Ein vollständiges Beispiel finden Sie unter [DiscoveryDecryptionExample.java](#) in der AWS-Verschlüsselungs-SDK for Java.

```
// Create a discovery filter for an AWS account and partition
```

```
DiscoveryFilter discoveryFilter = new DiscoveryFilter("aws", 111122223333);
```

JavaScript (Node and Browser)

[Vollständige Beispiele finden Sie unter kms_filtered_discovery.ts \(Node.js\) und kms_multi_region_discovery.ts \(Browser\) im AWS-Verschlüsselungs-SDK for JavaScript](#)

```
/* Create a discovery filter for an AWS account and partition */
const discoveryFilter = {
  accountIDs: ['111122223333'],
  partition: 'aws',
}
```

Python

Ein vollständiges Beispiel finden [Sie](#) unter AWS-Verschlüsselungs-SDK for Python `discovery_kms_provider.py` in der.

```
# Create the discovery filter and specify the region
decrypt_kwargs = dict(
    discovery_filter=DiscoveryFilter(account_ids="111122223333",
    partition="aws"),
    discovery_region="us-west-2",
)
```

Festlegung einer Verpflichtungspolitik

Eine [Commitment-Richtlinie ist eine Konfigurationseinstellung, die bestimmt, ob Ihre Anwendung mit Key Commitment ver- oder entschlüsselt wird. Das Verschlüsseln und Entschlüsseln mit Key Commitment ist eine bewährte Methode.](#)[AWS Encryption SDK](#)

Die Festlegung und Anpassung Ihrer Verpflichtungsrichtlinie ist ein wichtiger Schritt bei der [Migration von Versionen 1.7. x und frühere Versionen von AWS Encryption SDK auf Version 2.0. x und später](#). Dieser Fortschritt wird im [Thema Migration](#) ausführlich erklärt.

Der Standardwert der Commitment-Richtlinie in den neuesten Versionen von AWS Encryption SDK (ab Version 2.0. x), `RequireEncryptRequireDecrypt`, ist für die meisten Situationen ideal. Wenn Sie jedoch Chiffretext entschlüsseln müssen, der ohne Schlüsselbindung verschlüsselt wurde, müssen Sie Ihre Verpflichtungsrichtlinie möglicherweise auf ändern.

`RequireEncryptAllowDecrypt` Beispiele dafür, wie Sie in jeder Programmiersprache eine Commitment-Richtlinie einrichten, finden Sie unter [Festlegung Ihrer Verpflichtungspolitik](#)

Arbeiten mit Streaming-Daten

Beachten Sie beim Streamen von Daten zur Entschlüsselung, dass der entschlüsselte Klartext AWS Encryption SDK zurückgibt, nachdem die Integritätsprüfungen abgeschlossen sind, aber bevor die digitale Signatur verifiziert wurde. Um sicherzustellen, dass Sie erst dann Klartext zurückgeben oder verwenden, wenn die Signatur verifiziert ist, empfehlen wir, den gestreamten Klartext zu puffern, bis der gesamte Entschlüsselungsprozess abgeschlossen ist.

[Dieses Problem tritt nur auf, wenn Sie Chiffretext zur Entschlüsselung streamen und nur, wenn Sie eine Algorithmsuite verwenden, z. B. die Standard-Algorithmsuite, die digitale Signaturen enthält.](#)

Um die Pufferung zu vereinfachen, enthalten einige AWS Encryption SDK Sprachimplementierungen, z. B. AWS-Verschlüsselungs-SDK for JavaScript in Node.js, eine Pufferfunktion als Teil der Entschlüsselungsmethode. Mit der AWS VerschlüsselungCLI, die immer Eingabe und Ausgabe streamt, wurde in Version 1.9 ein `--buffer` Parameter eingeführt. x und 2.2. x. In anderen Sprachimplementierungen können Sie vorhandene Pufferfunktionen verwenden. (Das für AWS Encryption SDK .NETunterstützt kein Streaming.)

Wenn Sie eine Algorithmus-Suite ohne digitale Signaturen verwenden, stellen Sie sicher, dass Sie die `decrypt-unsigned` Funktion in jeder Sprachimplementierung verwenden. Diese Funktion entschlüsselt Chiffretext, schlägt jedoch fehl, wenn signierter Chiffretext gefunden wird. Details hierzu finden Sie unter [Auswahl einer Algorithmus-Suite](#).

Zwischenspeichern von Datenschlüsseln

Im Allgemeinen wird von der Wiederverwendung von Datenschlüsseln abgeraten, AWS Encryption SDK bietet jedoch eine Option zum [Zwischenspeichern von Datenschlüsseln](#), die eine begrenzte Wiederverwendung von Datenschlüsseln ermöglicht. Das Zwischenspeichern von Datenschlüsseln kann die Leistung einiger Anwendungen verbessern und die Anzahl der Zugriffe auf Ihre wichtige Infrastruktur reduzieren. Bevor Sie das Zwischenspeichern von Datenschlüsseln in der Produktion verwenden, passen Sie die [Sicherheitsschwellen](#) an und testen Sie, ob die Vorteile die Nachteile der Wiederverwendung von Datenschlüsseln überwiegen.

Verwenden von Schlüsselbunden

Das AWS-Verschlüsselungs-SDK for C, das AWS-Verschlüsselungs-SDK for JavaScript, das AWS-Verschlüsselungs-SDK for Java, das und das AWS Encryption SDK für .NET verwenden Sie Schlüsselringe, um die [Umschlagverschlüsselung](#) durchzuführen. Schlüsselbunde generieren, verschlüsseln und entschlüsseln Datenschlüssel. Schlüsselbunde bestimmen die Quelle der eindeutigen Datenschlüssel, die jede Nachricht schützen, und der [Umschließungsschlüssel, die diesen Datenschlüssel](#) verschlüsseln. Sie geben bei der Verschlüsselung einen Schlüsselbund und bei der Entschlüsselung denselben oder einen anderen Schlüsselbund an. Sie können die von der bereitgestellten Schlüsselbunde verwenden oder Ihre eigenen kompatiblen benutzerdefinierten Schlüsselanhänger schreiben. SDK

Sie können jeden Schlüsselbund einzeln verwenden oder Schlüsselbunde in einen [Multi-Schlüsselbund](#) kombinieren. Obwohl die meisten Schlüsselbunde Datenschlüssel generieren, verschlüsseln und entschlüsseln können, können Sie einen Schlüsselbund erstellen, der nur eine bestimmte Operation ausführt, wie z. B. einen Schlüsselbund, der nur Datenschlüssel generiert. Dieser Schlüsselbund kann dann in Kombination mit anderen verwendet werden.

Wir empfehlen Ihnen, einen Schlüsselbund zu verwenden, der Ihre Schlüssel schützt und kryptografische Operationen innerhalb einer sicheren Grenze ausführt, wie z. B. den AWS KMS Schlüsselbund, der diesen Never Leave () AWS KMS keys unverschlüsselt verwendet. [AWS Key Management Service](#) AWS KMS Sie können auch einen Schlüsselbund schreiben, bei dem Schlüssel zum Umschließen von Schlüsseln verwendet werden, die in Ihren Hardware-Sicherheitsmodulen (HSMs) gespeichert oder durch andere Master-Key-Dienste geschützt sind. Weitere Informationen finden Sie im Thema [Schlüsselbundchnittstelle](#) in der AWS Encryption SDK -Spezifikation.

Schlüsselringe spielen bei der, und der Verschlüsselung die Rolle von [Hauptschlüsseln und Hauptschlüsselanbietern](#). AWS-Verschlüsselungs-SDK for Java, AWS-Verschlüsselungs-SDK for Python, AWS CLI Wenn Sie unterschiedliche Sprachimplementierungen von verwenden, um Ihre Daten AWS Encryption SDK zu verschlüsseln und zu entschlüsseln, stellen Sie sicher, dass Sie kompatible Schlüsselringe und Hauptschlüsselanbieter verwenden. Details hierzu finden Sie unter [Schlüsselbund-Kompatibilität](#).

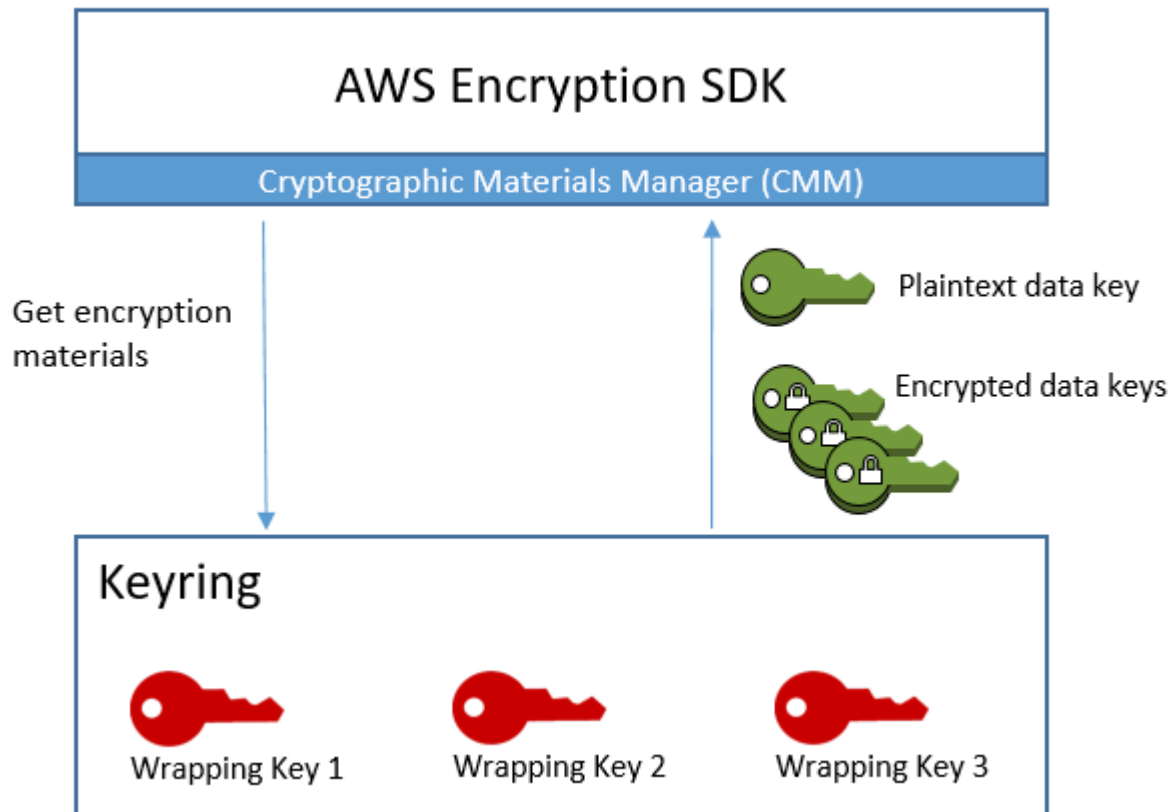
In diesem Thema wird erklärt, wie Sie die Schlüsselbundfunktion von verwenden AWS Encryption SDK und wie Sie einen Schlüsselbund auswählen. Beispiele für das Erstellen und Verwenden von Schlüsselbändern finden Sie in den Themen [C](#) und [JavaScript](#)

Themen

- [Funktionsweise von Schlüsselbunden](#)
- [Schlüsselbund-Kompatibilität](#)
- [Einen Schlüsselbund wählen](#)

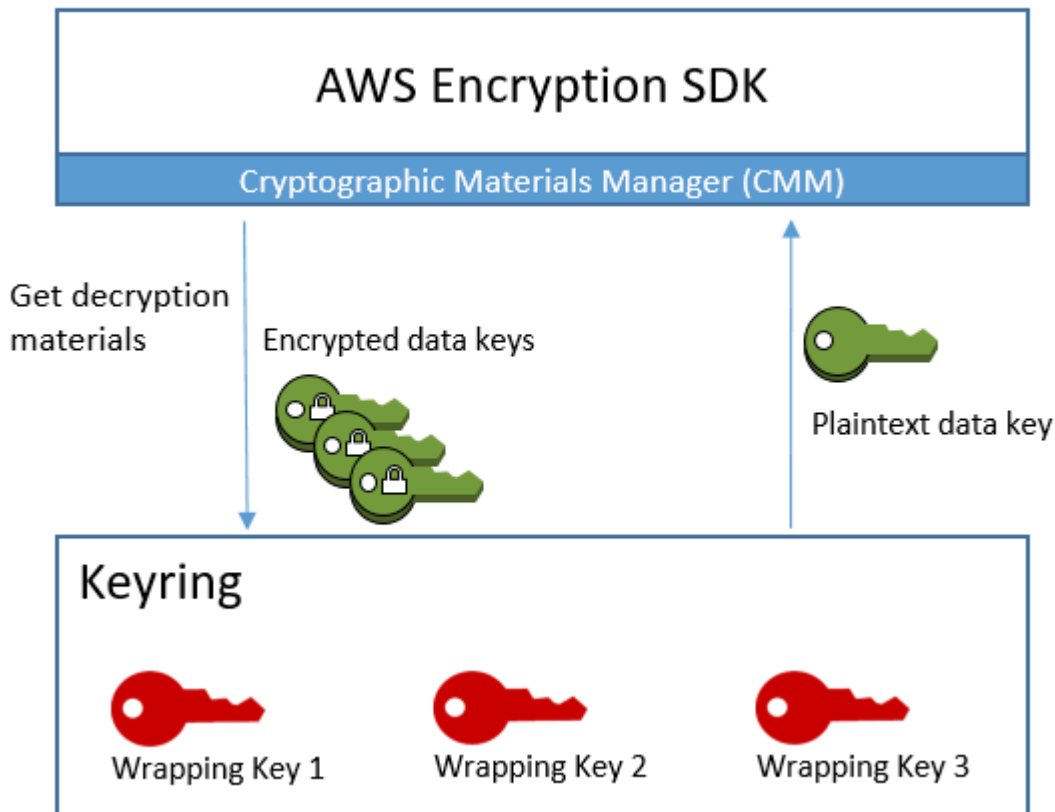
Funktionsweise von Schlüsselbunden

Wenn Sie Daten verschlüsseln, AWS Encryption SDK fragt den Schlüsselbund nach Verschlüsselungsmaterial. Der Schlüsselbund gibt einen Klartext-Datenschlüssel und eine Kopie des Datenschlüssels zurück, der durch die einzelnen Schlüssel im Schlüsselbund verschlüsselt wird. Der AWS Encryption SDK verwendet den Klartext-Schlüssel, um die Daten zu verschlüsseln, und zerstört dann den Klartext-Datenschlüssel. Anschließend wird eine [verschlüsselte Nachricht AWS Encryption SDK zurückgegeben](#), die die [verschlüsselten](#) Datenschlüssel und die verschlüsselten Daten enthält.



Wenn Sie Daten entschlüsseln, können Sie denselben Schlüsselbund verwenden, den Sie zum Verschlüsseln der Daten verwendet haben, oder einen anderen. Um die Daten zu entschlüsseln, muss ein Entschlüsselungsschlüsselbund mindestens einen Umschließungsschlüssel im Schlüsselbund enthalten (oder Zugriff darauf haben).

Der AWS Encryption SDK übergibt die verschlüsselten Datenschlüssel aus der verschlüsselten Nachricht an den Schlüsselbund und fordert den Schlüsselbund auf, einen davon zu entschlüsseln. Der Schlüsselbund verwendet seine Umhüllungsschlüssel zum Entschlüsseln eines der verschlüsselten Datenschlüssel und gibt einen Klartext-Datenschlüssel zurück. Das AWS Encryption SDK entschlüsselt die Daten mithilfe des Klartext-Datenschlüssels. Wenn keiner der Umhüllungsschlüssel im Schlüsselbund einen der verschlüsselten Datenschlüssel entschlüsseln kann, schlägt der Entschlüsselungsvorgang fehl.



Sie können einen einzelnen Schlüsselbund verwenden oder Schlüsselbunde desselben Typs oder eines anderen Typs in einem [Multi-Schlüsselbund](#) kombinieren. Wenn Sie Daten verschlüsseln, gibt der Multi-Schlüsselbund eine Kopie des Datenschlüssels zurück, der von allen Umhüllungsschlüsseln in allen Schlüsselbunden verschlüsselt wurde, aus denen der Multi-Schlüsselbund besteht. Sie können die Daten mithilfe eines Schlüsselbundes entschlüsseln, wobei jeder der Schlüssel im Mehrfachschlüsselbund eingeschlossen ist.

Schlüsselbund-Kompatibilität

Obwohl die verschiedenen Sprachimplementierungen von einige architektonische Unterschiede AWS Encryption SDK aufweisen, sind sie vollständig kompatibel und unterliegen sprachlichen Einschränkungen. Sie können Ihre Daten mit einer Sprachimplementierung verschlüsseln und mit jeder anderen Sprachimplementierung entschlüsseln. Sie müssen jedoch dieselben oder entsprechende Wrapping-Schlüssel verwenden, um Ihre Datenschlüssel zu verschlüsseln und zu entschlüsseln. Informationen zu Spracheinschränkungen finden Sie im Thema zu den einzelnen Sprachimplementierungen, z. B. [the section called "Kompatibilität"](#) im AWS-Verschlüsselungs-SDK for JavaScript Thema.

Unterschiedliche Anforderungen für Verschlüsselungsschlüsselringe

In anderen AWS Encryption SDK Sprachimplementierungen als dem AWS-Verschlüsselungs-SDK for C müssen alle Schlüssel in einem Verschlüsselungsschlüsselbund (oder einem Masterschlüsselbund) oder einem Masterschlüsselanbieter in der Lage sein, den Datenschlüssel zu verschlüsseln. Wenn ein Umschließungsschlüssel nicht verschlüsselt werden kann, schlägt die Verschlüsselungsmethode fehl. Daher muss der Anrufer über die [erforderlichen Berechtigungen](#) für alle Schlüssel im Schlüsselbund verfügen. Wenn Sie einen Discovery-Schlüsselbund verwenden, um Daten allein oder in einem Mehrfachschlüsselbund zu verschlüsseln, schlägt der Verschlüsselungsvorgang fehl.


Die Ausnahme ist der AWS-Verschlüsselungs-SDK for C, bei dem der Verschlüsselungsvorgang einen standardmäßigen Erkennungsschlüsselbund ignoriert, aber fehlschlägt, wenn Sie einen Erkennungsschlüsselbund mit mehreren Regionen angeben, allein oder in einem Schlüsselbund mit mehreren Schlüsselbunden.

Kompatible Schlüsselbunde und Masterschlüssel-Anbieter

Die folgende Tabelle zeigt, welche Hauptschlüssel und Hauptschlüsselanbieter mit den mitgelieferten Schlüsselbunden kompatibel sind. AWS Encryption SDK Geringfügige Inkompatibilitäten aufgrund von Spracheinschränkungen werden im Thema über die Sprachimplementierung beschrieben.

Schlüsselbund:	Hauptschlüsselanbieter:
AWS KMS Schlüsselring	KMSMasterKey(Java) KMSMasterKeyProvider(Java)

Schlüsselbund:	Hauptschlüsselanbieter:
	<p>KMSMasterKey(Python)</p> <p>KMSMasterKeyProvider(Python)</p> <div data-bbox="516 367 1507 682" style="border: 1px solid #add8e6; border-radius: 10px; padding: 10px; margin-top: 10px;"> <p> Note</p> <p>Die AWS-Verschlüsselungs-SDK for Python und enthalten AWS-Verschlüsselungs-SDK for Java keinen Hauptschlüssel oder Hauptschlüsselanbieter, was dem AWS KMS regionalen Discovery-Schlüsselbund entspricht.</p> </div>
AWS KMS Hierarchischer Schlüsselbund	Nur mit Version 4 verfügbar. x der AWS Encryption SDK Form. NET und Version 3. x der AWS-Verschlüsselungs-SDK for Java.
AWS KMS ECDH Schlüsselring	Nur mit Version 3 verfügbar. x der AWS-Verschlüsselungs-SDK for Java.
Roher AES Schlüsselring	<p>Wenn sie mit symmetrischen Verschlüsselungsschlüsseln verwendet werden:</p> <p>JceMasterKey(Java)</p> <p>RawMasterKey(Python)</p>

Schlüsselbund:	Hauptschlüsselanbieter:
Roher RSA Schlüsselbund	<p>Wenn sie mit asymmetrischen Verschlüsselungsschlüsseln verwendet werden:</p> <p>JceMasterKey(Java)</p> <p>RawMasterKey(Python)</p> <div data-bbox="516 478 1510 940" style="border: 1px solid #add8e6; border-radius: 10px; padding: 10px; margin-top: 10px;"> <p> Note</p> <p>Der RSA Raw-Schlüsselbund unterstützt keine asymmetrischen KMS Schlüssel. Wenn Sie asymmetrische RSA KMS Schlüssel verwenden möchten, Version 4. x der Form AWS Encryption SDK . NETunterstützt AWS KMS Schlüsselringe, die symmetrische Verschlüsselung (SYMMETRIC_DEFAULT) oder asymmetrische Verschlüsselung verwenden. RSA AWS KMS keys</p> </div>
Roher ECDH Schlüsselbund	<p>Nur mit Version 4 verfügbar. x der AWS Encryption SDK Form. NETund Version 3. x der AWS-Verschlüsselungs-SDK for Java.</p>

Einen Schlüsselbund wählen

Ihr Schlüsselbund bestimmt die Umschließungsschlüssel, die Ihre Datenschlüssel und letztlich Ihre Daten schützen. Verwenden Sie die sichersten Verpackungsschlüssel, die für Ihre Aufgabe praktisch sind. Verwenden Sie nach Möglichkeit Schlüssel, die durch ein Hardware-Sicherheitsmodul oder eine Schlüsselverwaltungsinfrastruktur geschützt sind, z. B. KMS Schlüssel in [AWS Key Management Service](#)(AWS KMS) oder Verschlüsselungsschlüssel [AWS CloudHSM](#).

Das AWS Encryption SDK bietet mehrere Schlüsselringe und Schlüsselbundkonfigurationen in mehreren Programmiersprachen, und Sie können Ihre eigenen benutzerdefinierten Schlüsselanhänger erstellen. Sie können auch einen [Mehrfachschlüsselbund](#) erstellen, der einen oder mehrere Schlüsselanhänger desselben oder eines anderen Typs enthält.

Themen

- [AWS KMS Schlüsselanhänger](#)

- [AWS KMS Hierarchische Schlüsselanhänger](#)
- [AWS KMS ECDH-Schlüsselbünde](#)
- [Unformatierte Schlüsselanhänger AES](#)
- [RSARaw-Schlüsselringe](#)
- [Rohe Schlüsselanhänger ECDH](#)
- [Multi-Schlüsselbünde](#)

AWS KMS Schlüsselanhänger

Ein AWS KMS Schlüsselbund verwendet symmetrische Verschlüsselung, um Datenschlüssel [AWS KMS keys](#) zu generieren, zu verschlüsseln und zu entschlüsseln. AWS Key Management Service (AWS KMS) schützt Ihre KMS Schlüssel und führt kryptografische Operationen innerhalb der Grenzen durch. FIPS Wir empfehlen, wann immer möglich einen AWS KMS Schlüsselbund oder einen Schlüsselbund mit ähnlichen Sicherheitseigenschaften zu verwenden.

[Ab Version 2.3 können Sie einen Schlüssel für AWS KMS mehrere Regionen in einem AWS KMS Schlüsselbund oder einem Hauptschlüsselanbieter verwenden.](#) x der Version AWS Encryption SDK und Version 3.0. x der AWS VerschlüsselungCLI. Einzelheiten und Beispiele zur Verwendung des neuen Symbols für mehrere Regionen finden Sie unter [Verwenden Sie mehrere Regionen AWS KMS keys](#) Informationen zu Schlüsseln für mehrere Regionen finden Sie unter [Verwenden von Schlüsseln für mehrere Regionen](#) im Entwicklerhandbuch.AWS Key Management Service

Note

Version 4. x der AWS Encryption SDK Form. NETund Version 3. x davon AWS-Verschlüsselungs-SDK for Java sind die einzigen Programmiersprachenimplementierungen, die AWS KMS Schlüsselringe unterstützen, die asymmetrisch verwenden. RSA AWS KMS keys

Wenn Sie versuchen, in einer anderen Sprachimplementierung einen asymmetrischen KMS Schlüssel in einen Verschlüsselungsschlüsselbund aufzunehmen, schlägt der Verschlüsselungsaufwurf fehl. Wenn Sie ihn in einen Schlüsselbund für die Entschlüsselung aufnehmen, wird er ignoriert.

Alle Erwähnungen von KMS Schlüsselanhängern in der AWS Encryption SDK beziehen sich auf Schlüsselanhänger. AWS KMS

AWS KMS Bei Schlüsselanhängern gibt es zwei Arten von Wickelschlüsseln:

- **Generatorschlüssel:** Generiert einen Klartext-Datenschlüssel und verschlüsselt ihn. Ein Schlüsselbund, der Daten verschlüsselt, muss einen Generatorschlüssel haben.
- **Zusätzliche Schlüssel:** Verschlüsselt den Klartext-Datenschlüssel, den der Generatorschlüssel generiert hat. AWS KMS Schlüsselbunde können null oder mehr zusätzliche Schlüssel haben.

Beim Verschlüsseln muss der AWS KMS Schlüsselbund, den Sie verwenden, über einen Generatorschlüssel verfügen. Bei der Entschlüsselung ist der Generatorschlüssel optional, und die Unterscheidung zwischen Generatorschlüsseln und zusätzlichen Schlüsseln wird ignoriert.

Wenn ein AWS KMS Verschlüsselungsschlüsselbund nur einen AWS KMS Schlüssel hat, wird dieser Schlüssel verwendet, um den Datenschlüssel zu generieren und zu verschlüsseln.

Wie alle Schlüsselanhänger können AWS KMS Schlüsselringe unabhängig voneinander oder in einem [Mehrfachschlüsselbund mit anderen Schlüsselanhängern desselben](#) oder eines anderen Typs verwendet werden.

Themen

- [AWS KMS Erforderliche Berechtigungen für Schlüsselanhänger](#)
- [Identifizierung AWS KMS keys in einem AWS KMS Schlüsselbund](#)
- [Einen AWS KMS Schlüsselbund für die Verschlüsselung erstellen](#)
- [Einen AWS KMS Schlüsselbund für die Entschlüsselung erstellen](#)
- [Verwenden Sie einen Discovery-Schlüsselbund AWS KMS](#)
- [Verwenden Sie einen AWS KMS regionalen Discovery-Schlüsselbund](#)

AWS KMS Erforderliche Berechtigungen für Schlüsselanhänger

Das benötigt AWS Encryption SDK kein AWS-Konto und es hängt auch nicht von einem ab. AWS - Service Um einen AWS KMS Schlüsselbund verwenden zu können, benötigen Sie jedoch eine AWS-Konto und die folgenden Mindestberechtigungen für AWS KMS keys den Schlüsselbund.

- Um mit einem AWS KMS Schlüsselbund zu verschlüsseln, benötigen Sie die [kms:GenerateDataKey](#) -Berechtigung für den Generatorschlüssel. Sie benötigen die [kms:Encrypt-Berechtigung für alle zusätzlichen](#) Schlüssel im Schlüsselbund. AWS KMS

- Um mit einem AWS KMS Schlüsselbund zu entschlüsseln, benötigen Sie die [kms:Decrypt-Berechtigung](#) für mindestens einen Schlüssel im Schlüsselbund. AWS KMS
- Um mit einem aus Schlüsselbunden bestehenden Mehrfachschlüsselbund zu verschlüsseln, benötigen Sie die kms-Berechtigung für den AWS KMS Generatorschlüssel im Generator-Schlüsselbund. [GenerateDataKey](#) Sie benötigen die [kms:Encrypt-Berechtigung](#) für alle anderen Schlüssel in allen anderen Schlüsselbunden. AWS KMS

Ausführliche Informationen zu den Berechtigungen für AWS KMS keys finden Sie unter [Authentifizierung und Zugriffskontrolle](#) im Entwicklerhandbuch.AWS Key Management Service

Identifizierung AWS KMS keys in einem AWS KMS Schlüsselbund

Ein AWS KMS Schlüsselbund kann einen oder mehrere enthalten. AWS KMS keys Um einen AWS KMS key in einem AWS KMS Schlüsselbund anzugeben, verwenden Sie eine unterstützte AWS KMS Schlüssel-ID. Die Schlüsselbezeichner, die Sie zur Identifizierung eines AWS KMS key in einem Schlüsselbund verwenden können, variieren je nach Vorgang und Sprachimplementierung. Einzelheiten zu den Schlüsselkennungen für einen AWS KMS key finden Sie unter [Schlüsselkennungen](#) im Entwicklerhandbuch.AWS Key Management Service

Es hat sich bewährt, die spezifischste Schlüssel-ID zu verwenden, die für Ihre Aufgabe praktikabel ist.

- In einem Verschlüsselungsschlüsselbund für können Sie einen [Schlüssel ARN](#) oder [Alias](#) verwenden AWS-Verschlüsselungs-SDK for C, um KMS Schlüssel ARN zu identifizieren. [In allen anderen Sprachimplementierungen können Sie eine Schlüssel-ID, einen Schlüssel ARN, einen Aliasnamen oder einen Alias verwenden, ARN um Daten zu verschlüsseln.](#)
- In einem Schlüsselbund zur Entschlüsselung müssen Sie einen Schlüssel zur Identifizierung verwenden. ARN AWS KMS keys Diese Anforderung gilt für alle Sprachenimplementierungen des AWS Encryption SDK. Details hierzu finden Sie unter [Auswahl von Schlüsseln zum Umbrechen.](#)
- In einem Schlüsselbund, der zur Verschlüsselung und Entschlüsselung verwendet wird, müssen Sie einen Schlüssel zur Identifizierung verwenden. ARN AWS KMS keys Diese Anforderung gilt für alle Sprachenimplementierungen des AWS Encryption SDK.

Wenn Sie einen Aliasnamen oder Alias ARN für einen KMS Schlüssel in einem Verschlüsselungsschlüsselbund angeben, speichert der Verschlüsselungsvorgang den ARN aktuell mit dem Alias verknüpften Schlüssel in den Metadaten des verschlüsselten Datenschlüssels. Der Alias wird nicht gespeichert. Änderungen am Alias wirken sich nicht auf den KMS Schlüssel aus, der zum Entschlüsseln Ihrer verschlüsselten Datenschlüssel verwendet wird.

Einen AWS KMS Schlüsselbund für die Verschlüsselung erstellen

Sie können jeden AWS KMS Schlüsselbund mit einem AWS KMS key oder mehreren Schlüsselbändern AWS KMS keys im selben oder einem anderen AWS-Konten und konfigurieren. AWS-Regionen Das AWS KMS keys müssen symmetrische Verschlüsselungsschlüssel (SYMMETRIC_DEFAULT) sein. Sie können auch einen Schlüssel für [mehrere KMS](#) Regionen mit symmetrischer Verschlüsselung verwenden. [Wie bei allen Schlüsselbänden können Sie einen oder mehrere AWS KMS Schlüsselringe in einem Schlüsselbund verwenden.](#)

Wenn Sie einen AWS KMS Schlüsselbund zum Verschlüsseln von Daten erstellen, müssen Sie einen Generatorschlüssel angeben. Dieser wird verwendet, um einen Klartext-Datenschlüssel zu generieren und AWS KMS key diesen zu verschlüsseln. Der Datenschlüssel hat mathematisch gesehen nichts mit dem Schlüssel zu tun. KMS Wenn Sie möchten, können Sie dann weitere angeben, AWS KMS keys die denselben Klartext-Datenschlüssel verschlüsseln.

Um die durch diesen Schlüsselbund geschützte verschlüsselte Nachricht zu entschlüsseln, muss der verwendete Schlüsselbund mindestens einen der im Schlüsselbund AWS KMS keys definierten Elemente enthalten, oder nein. AWS KMS keys([Ein AWS KMS Schlüsselbund ohne AWS KMS keys wird als Discovery-Schlüsselbund bezeichnet.](#))[AWS KMS](#)

In anderen AWS Encryption SDK Sprachimplementierungen als dem müssen alle Schlüssel AWS-Verschlüsselungs-SDK for C, die in einen Verschlüsselungsschlüsselbund oder einen Mehrfachschlüsselbund eingeschlossen werden, in der Lage sein, den Datenschlüssel zu verschlüsseln. Wenn ein Wrapping-Schlüssel nicht verschlüsselt werden kann, schlägt die Verschlüsselungsmethode fehl. Daher muss der Anrufer über die [erforderlichen Berechtigungen](#) für alle Schlüssel im Schlüsselbund verfügen. Wenn Sie einen Discovery-Schlüsselbund verwenden, um Daten allein oder in einem Mehrfachschlüsselbund zu verschlüsseln, schlägt der Verschlüsselungsvorgang fehl. Die Ausnahme ist der AWS-Verschlüsselungs-SDK for C, bei dem der Verschlüsselungsvorgang einen standardmäßigen Erkennungsschlüsselbund ignoriert, aber fehlschlägt, wenn Sie einen Erkennungsschlüsselbund mit mehreren Regionen angeben, allein oder in einem Schlüsselbund mit mehreren Schlüsselbänden.

In den folgenden Beispielen wird ein AWS KMS Schlüsselbund mit einem Generatorschlüssel und einem zusätzlichen Schlüssel erstellt. In diesen Beispielen werden [Schlüssel](#) verwendet ARNs, um die KMS Schlüssel zu identifizieren. Dies ist eine bewährte Methode für AWS KMS Schlüsselringe, die zur Verschlüsselung verwendet werden, und eine Voraussetzung für AWS KMS Schlüsselringe, die zur Entschlüsselung verwendet werden. Details hierzu finden Sie unter [Identifizierung AWS KMS keys in einem AWS KMS Schlüsselbund.](#)

C

[Um einen Schlüsselbund AWS KMS key in einer Verschlüsselung zu identifizieren AWS-Verschlüsselungs-SDK for C, geben Sie einen Schlüssel oder Alias an. ARN ARN](#) In einem Schlüsselbund zur Entschlüsselung müssen Sie einen Schlüssel verwenden. ARN Details hierzu finden Sie unter [Identifizierung AWS KMS keys in einem AWS KMS Schlüsselbund](#).

Ein vollständiges Beispiel finden Sie unter [string.cpp](#).

```
const char * generator_key = "arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"  
  
const char * additional_key = "arn:aws:kms:us-  
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321"  
  
struct aws_cryptosdk_keyring *kms_encrypt_keyring =  
    Aws::Cryptosdk::KmsKeyring::Builder().Build(generator_key, {additional_key});
```

C# / .NET

Um einen AWS KMS Schlüsselbund mit einem oder mehreren AWS KMS Schlüsseln im Formular zu erstellen. AWS Encryption SDK NET, erstellen Sie einen Schlüsselbund mit mehreren Schlüsseln. Das AWS Encryption SDK für .NET enthält einen Schlüsselring mit mehreren Schlüsseln nur für AWS KMS Schlüssel.

Wenn Sie im Feld AWS KMS key für einen Verschlüsselungsschlüsselbund einen Schlüsselbund angeben. AWS Encryption SDK NET, können Sie eine beliebige gültige Schlüssel-ID verwenden: eine [Schlüssel-ID](#), einen [Schlüssel ARN](#), einen [Aliasnamen](#) oder einen [Alias ARN](#). Hilfe zur Identifizierung von AWS KMS keys in einem AWS KMS Schlüsselbund finden Sie unter [Identifizierung AWS KMS keys in einem AWS KMS Schlüsselbund](#).

Im folgenden Beispiel wird Version 4 verwendet. x des AWS Encryption SDK Forums. NET Um einen AWS KMS Schlüsselbund mit einem Generatorschlüssel und zusätzlichen Tasten zu erstellen. Ein vollständiges Beispiel finden Sie unter [AwsKmsMultiKeyringExample.cs](#).

```
// Instantiate the AWS Encryption SDK and material provider  
var mpl = new MaterialProviders(new MaterialProvidersConfig());  
var esdk = new ESDK(new AwsEncryptionSdkConfig());  
  
string generatorKey = "arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
```

```
List<string> additionalKey = new List<string> { "alias/exampleAlias" };

// Instantiate the keyring input object
var kmsEncryptKeyringInput = new CreateAwsKmsMultiKeyringInput
{
    Generator = generatorKey,
    KmsKeyIds = additionalKey
};

var kmsEncryptKeyring =
    materialProviders.CreateAwsKmsMultiKeyring(kmsEncryptKeyringInput);
```

JavaScript Browser

[Wenn Sie in der einen Schlüsselbund AWS KMS key für die Verschlüsselung angeben AWS-Verschlüsselungs-SDK for JavaScript, können Sie eine beliebige gültige Schlüssel-ID verwenden: eine Schlüssel-ID, einen Schlüssel ARN, einen Aliasnamen oder einen Alias. ARN Hilfe bei der Identifizierung von AWS KMS keys in einem AWS KMS Schlüsselbund finden Sie unter. Identifizierung AWS KMS keys in einem AWS KMS Schlüsselbund](#)

Ein vollständiges Beispiel finden Sie unter [kms_simple.ts](#) im Repository unter. AWS-Verschlüsselungs-SDK for JavaScript GitHub

```
const clientProvider = getClient(KMS, { credentials })
const generatorKeyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
const additionalKey = 'alias/exampleAlias'

const keyring = new KmsKeyringBrowser({
    clientProvider,
    generatorKeyId,
    keyIds: [additionalKey]
})
```

JavaScript Node.js

[Wenn Sie in der einen Schlüsselbund AWS KMS key für die Verschlüsselung angeben AWS-Verschlüsselungs-SDK for JavaScript, können Sie eine beliebige gültige Schlüssel-ID verwenden: eine Schlüssel-ID, einen Schlüssel ARN, einen Aliasnamen oder einen Alias. ARN Hilfe bei der Identifizierung von AWS KMS keys in einem AWS KMS Schlüsselbund finden Sie unter. Identifizierung AWS KMS keys in einem AWS KMS Schlüsselbund](#)

Ein vollständiges Beispiel finden Sie unter [kms_simple.ts](#) im Repository unter. AWS-Verschlüsselungs-SDK for JavaScript GitHub

```
const generatorKeyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'  
  
const additionalKey = 'alias/exampleAlias'  
  
const keyring = new KmsKeyringNode({  
  generatorKeyId,  
  keyIds: [additionalKey]  
})
```

Java

Um einen AWS KMS Schlüsselbund mit einem oder mehreren Schlüsseln zu erstellen, erstellen Sie einen Schlüsselbund mit mehreren AWS KMS Schlüsseln. AWS-Verschlüsselungs-SDK for Java Der AWS-Verschlüsselungs-SDK for Java beinhaltet einen Mehrfachschlüsselbund nur für Schlüssel. AWS KMS

[Wenn Sie in der einen Schlüsselbund AWS KMS key für die Verschlüsselung angeben AWS-Verschlüsselungs-SDK for Java, können Sie eine beliebige gültige Schlüssel-ID verwenden: eine Schlüssel-ID, einen Schlüssel ARN, einen Aliasnamen oder einen Alias. ARN Hilfe bei der Identifizierung von AWS KMS keys in einem AWS KMS Schlüsselbund finden Sie unter. Identifizierung AWS KMS keys in einem AWS KMS Schlüsselbund](#)

Ein vollständiges Beispiel finden Sie unter [BasicEncryptionKeyringExample.java](#) im AWS-Verschlüsselungs-SDK for Java Repository unter. GitHub

```
// Instantiate the AWS Encryption SDK and material providers  
final AwsCrypto crypto = AwsCrypto.builder().build();  
final MaterialProviders materialProviders = MaterialProviders.builder()  
    .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())  
    .build();  
  
String generatorKey = "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";  
List<String> additionalKey = Collections.singletonList("alias/exampleAlias");  
  
// Create the AWS KMS keyring  
final CreateAwsKmsMultiKeyringInput keyringInput =  
    CreateAwsKmsMultiKeyringInput.builder()
```

```
.generator(generatorKey)
.kmsKeyIds(additionalKey)
.build();
final IKeyring kmsKeyring = matProv.CreateAwsKmsMultiKeyring(keyringInput);
```

Einen AWS KMS Schlüsselbund für die Entschlüsselung erstellen

Sie geben auch einen AWS KMS Schlüsselbund an, wenn Sie die [verschlüsselte Nachricht entschlüsseln, die zurückgegeben wird](#). AWS Encryption SDK Wenn der Schlüsselbund für die Entschlüsselung einen Wert angibt AWS KMS keys, verwendet er nur diese Umschließungsschlüssel, um die verschlüsselten Datenschlüssel in der verschlüsselten Nachricht zu entschlüsseln. AWS Encryption SDK (Sie können auch einen [AWS KMS Discovery-Schlüsselbund](#) verwenden, der keinen angibt.) AWS KMS keys

Beim Entschlüsseln AWS Encryption SDK durchsucht der den AWS KMS Schlüsselbund nach einem Schlüssel AWS KMS key , der einen der verschlüsselten Datenschlüssel entschlüsseln kann. Insbesondere AWS Encryption SDK verwendet der das folgende Muster für jeden verschlüsselten Datenschlüssel in einer verschlüsselten Nachricht.

- Der AWS Encryption SDK ruft den Schlüssel ARN des Datenschlüssels AWS KMS key , der den Datenschlüssel verschlüsselt hat, aus den Metadaten der verschlüsselten Nachricht ab.
- Der AWS Encryption SDK durchsucht den Schlüsselbund für die Entschlüsselung nach einem AWS KMS key mit einem passenden Schlüssel. ARN
- Wenn ARN im Schlüsselbund ein Schlüssel AWS KMS key mit einem passenden Schlüssel gefunden wird, werden Sie AWS Encryption SDK aufgefordert, den Schlüssel AWS KMS zum Entschlüsseln des verschlüsselten KMS Datenschlüssels zu verwenden.
- Andernfalls springt er zum nächsten verschlüsselten Datenschlüssel, falls vorhanden.

Der versucht AWS Encryption SDK nie, einen verschlüsselten Datenschlüssel zu entschlüsseln, es sei denn, der Schlüssel ARN des Schlüssels AWS KMS key , der diesen Datenschlüssel verschlüsselt hat, ist im Schlüsselbund für die Entschlüsselung enthalten. Wenn der Schlüsselbund für die Entschlüsselung keinen der Schlüssel enthält, ARNs der einen der Datenschlüssel verschlüsselt hat AWS KMS keys , schlägt der Entschlüsselungsaufwurf AWS Encryption SDK fehl, ohne jemals aufgerufen zu haben. AWS KMS

[Ab Version 1.7.](#) x, wenn ein verschlüsselter Datenschlüssel entschlüsselt wird, übergibt der AWS Encryption SDK immer den Schlüssel ARN von AWS KMS key an den KeyId Parameter

der AWS KMS [Decrypt-Operation](#). Es hat sich AWS KMS key bewährt, beim Entschlüsseln den zu identifizieren, um sicherzustellen, dass Sie den verschlüsselten Datenschlüssel mit dem Umschließungsschlüssel entschlüsseln, den Sie verwenden möchten. AWS KMS

Ein Entschlüsselungsaufruf mit einem AWS KMS Schlüsselbund ist erfolgreich, wenn mindestens ein Schlüsselbund AWS KMS key in der Lage ist, einen der verschlüsselten Datenschlüssel in der verschlüsselten Nachricht zu entschlüsseln. Außerdem muss der Aufrufer über die `kms:Decrypt`-Berechtigung für diesen AWS KMS key verfügen. Dieses Verhalten ermöglicht es Ihnen, Daten unter mehreren Konten AWS KMS keys in verschiedenen AWS-Regionen AND-Konten zu verschlüsseln, aber einen eingeschränkteren Entschlüsselungsschlüsselbund bereitzustellen, der auf ein bestimmtes Konto, eine Region, einen Benutzer, eine Gruppe oder eine Rolle zugeschnitten ist.

Wenn Sie AWS KMS key in einem Entschlüsselungsschlüsselbund einen angeben, müssen Sie seinen Schlüssel verwenden. ARN Andernfalls AWS KMS key wird der nicht erkannt. Hilfe bei der Suche nach dem Schlüssel ARN [finden Sie unter Finden der Schlüssel-ID und ARN](#) im AWS Key Management Service Entwicklerhandbuch.

Note

Wenn Sie einen verschlüsselten Schlüsselbund für die Entschlüsselung wiederverwenden, stellen Sie sicher, dass die AWS KMS keys im Schlüsselbund enthaltenen Schlüssel identifiziert werden. ARNs

Der folgende AWS KMS Schlüsselbund enthält beispielsweise nur den zusätzlichen Schlüssel, der im Verschlüsselungsschlüsselbund verwendet wurde. Anstatt jedoch mit seinem Alias auf den Zusatzschlüssel zu verweisen, verwendet das Beispiel den Schlüssel des zusätzlichen Schlüssels `alias/exampleAlias`, ARN wie er für Decrypt-Aufrufe erforderlich ist.

Sie können diesen Schlüsselbund verwenden, um eine Nachricht zu entschlüsseln, die sowohl unter dem Generatorschlüssel als auch dem zusätzlichen Schlüssel verschlüsselt wurde, vorausgesetzt, dass Sie über die Berechtigung verfügen, den zusätzlichen Schlüssel zum Entschlüsseln von Daten zu verwenden.

C

```
const char * additional_key = "arn:aws:kms:us-  
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321"  
  
struct aws_cryptosdk_keyring *kms_decrypt_keyring =
```



```
Aws::Cryptosdk::KmsKeyring::Builder().Build(additional_key);
```

C# / .NET

Da dieser Schlüsselbund für die Entschlüsselung nur einen AWS KMS Schlüssel enthält, verwendet das Beispiel die `CreateAwsKmsKeyring()` Methode mit einer Instanz ihres Objekts. `CreateAwsKmsKeyringInput` Um einen AWS KMS Schlüsselbund mit einem Schlüssel zu erstellen, können Sie einen AWS KMS Schlüsselbund mit einem Schlüssel oder einem Schlüsselbund mit mehreren Schlüsseln verwenden. Details hierzu finden Sie unter [Verschlüsseln von Daten im AWS Encryption SDK für .NET](#). Im folgenden Beispiel wird Version 4 verwendet. x des AWS Encryption SDK Forums. NET Um einen AWS KMS Schlüsselbund für die Entschlüsselung zu erstellen.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk = new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

string additionalKey = "arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321";

// Instantiate a KMS keyring for one AWS KMS key.
var kmsDecryptKeyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = additionalKey
};

var kmsDecryptKeyring =
    materialProviders.CreateAwsKmsKeyring(kmsDecryptKeyringInput);
```

JavaScript Browser

```
const clientProvider = getClient(KMS, { credentials })
const additionalKey = 'arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321'

const keyring = new KmsKeyringBrowser({ clientProvider, keyIds: [additionalKey] })
```

JavaScript Node.js

```
const additionalKey = 'arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321'
```

```
const keyring = new KmsKeyringNode({ keyIds: [additionalKey] })
```

Java

Da dieser Schlüsselbund zur Entschlüsselung nur einen AWS KMS Schlüssel enthält, verwendet das Beispiel die `CreateAwsKmsKeyring()` Methode mit einer Instanz ihres Objekts. `CreateAwsKmsKeyringInput` Um einen AWS KMS Schlüsselbund mit einem Schlüssel zu erstellen, können Sie einen AWS KMS Schlüsselbund mit einem Schlüssel oder einem Schlüsselbund mit mehreren Schlüsseln verwenden.

```
// Instantiate the AWS Encryption SDK and material providers
final AwsCrypto crypto = AwsCrypto.builder().build();
final MaterialProviders materialProviders = MaterialProviders.builder()
    .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
    .build();

String additionalKey = "arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321";

// Create a AwsKmsKeyring
CreateAwsKmsKeyringInput kmsDecryptKeyringInput = CreateAwsKmsKeyringInput.builder()
    .generator(additionalKey)
    .kmsClient(KmsClient.create())
    .build();
IKeyring kmsKeyring = materialProviders.CreateAwsKmsKeyring(kmsDecryptKeyringInput);
```

Sie können auch einen AWS KMS Schlüsselbund verwenden, der einen Generatorschlüssel für die Entschlüsselung angibt, z. B. den folgenden. Beim Entschlüsseln AWS Encryption SDK ignoriert der den Unterschied zwischen Generatorschlüsseln und zusätzlichen Schlüsseln. Es kann jeden der angegebenen Schlüssel verwenden, AWS KMS keys um einen verschlüsselten Datenschlüssel zu entschlüsseln. Der Aufruf von AWS KMS ist nur erfolgreich, wenn der Aufrufer die Erlaubnis hat, ihn zum Entschlüsseln von Daten AWS KMS key zu verwenden.

C

```
struct aws_cryptosdk_keyring *kms_decrypt_keyring =
    Aws::Cryptosdk::KmsKeyring::Builder().Build(generator_key, {additional_key,
    other_key});
```

C# / .NET

Im folgenden Beispiel wird Version 4 verwendet. x des AWS Encryption SDK Forums. NET.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk = new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

string generatorKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

// Instantiate a KMS keyring for one AWS KMS key.
var kmsDecryptKeyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = generatorKey
};

var kmsDecryptKeyring =
    materialProviders.CreateAwsKmsKeyring(kmsDecryptKeyringInput);
```

JavaScript Browser

```
const clientProvider = getClient(KMS, { credentials })

const keyring = new KmsKeyringBrowser({
    clientProvider,
    generatorKeyId,
    keyIds: [additionalKey, otherKey]
})
```

JavaScript Node.js

```
const keyring = new KmsKeyringNode({
    generatorKeyId,
    keyIds: [additionalKey, otherKey]
})
```

Java

```
// Instantiate the AWS Encryption SDK and material providers
final AwsCrypto crypto = AwsCrypto.builder().build();
final MaterialProviders materialProviders = MaterialProviders.builder()
```

```
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();

String generatorKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

// Create a AwsKmsKeyring
CreateAwsKmsKeyringInput kmsDecryptKeyringInput = CreateAwsKmsKeyringInput.builder()
    .generator(generatorKey)
    .kmsClient(KmsClient.create())
    .build();
IKeyring kmsKeyring = materialProviders.CreateAwsKmsKeyring(kmsDecryptKeyringInput);
```

Im Gegensatz zu einem Schlüsselbund, der alle angegebenen Daten verwendet AWS KMS keys, können Sie eine verschlüsselte Nachricht mit einem Schlüsselbund entschlüsseln, der auch solche enthält AWS KMS keys, die nichts mit der verschlüsselten Nachricht zu tun haben und AWS KMS keys zu deren Verwendung der Anrufer nicht berechtigt ist. Wenn ein Entschlüsselungsaufruf an AWS KMS fehlschlägt, z. B. wenn der Aufrufer nicht über die erforderliche Berechtigung verfügt, springt das AWS Encryption SDK einfach zum nächsten verschlüsselten Datenschlüssel.

Verwenden Sie einen Discovery-Schlüsselbund AWS KMS

Beim Entschlüsseln empfiehlt es sich, die [Umschließungsschlüssel](#) anzugeben, die sie verwenden AWS Encryption SDK können. Um dieser bewährten Methode zu folgen, sollten Sie einen Schlüsselbund für die AWS KMS Entschlüsselung verwenden, der die Anzahl der AWS KMS Umschließungsschlüssel auf die von Ihnen angegebenen beschränkt. Sie können jedoch auch einen AWS KMS Discovery-Schlüsselbund erstellen, d. h. einen AWS KMS Schlüsselbund, der keine Schlüssel zum Umschließen festlegt.

Der AWS Encryption SDK bietet einen AWS KMS Standard-Discovery-Schlüsselbund und einen Discovery-Schlüsselbund für Schlüssel mit mehreren Regionen. AWS KMS Hinweise zur Verwendung von Regionsschlüsseln mit dem finden Sie unter [AWS Encryption SDK Verwenden Sie mehrere Regionen AWS KMS keys](#)

Da er keine Umschließungsschlüssel angibt, kann ein Discovery-Schlüsselbund keine Daten verschlüsseln. Wenn Sie einen Discovery-Schlüsselbund verwenden, um Daten allein oder in einem Mehrfachschlüsselbund zu verschlüsseln, schlägt der Verschlüsselungsvorgang fehl. Die Ausnahme ist der AWS-Verschlüsselungs-SDK for C, bei dem der Verschlüsselungsvorgang einen standardmäßigen Erkennungsschlüsselbund ignoriert, aber fehlschlägt, wenn Sie einen

Erkennungsschlüsselbund mit mehreren Regionen angeben, allein oder in einem Schlüsselbund mit mehreren Schlüsselbunden.

Bei der Entschlüsselung ermöglicht ein Discovery-Schlüsselbund, die Entschlüsselung eines beliebigen verschlüsselten Datenschlüssels mithilfe des Schlüssels AWS Encryption SDK AWS KMS anzufordern, der ihn verschlüsselt hat, unabhängig davon, wem dieser gehört oder wer Zugriff AWS KMS key darauf hat. AWS KMS key Der Anruf ist nur erfolgreich, wenn der Anrufer die Erlaubnis für hat. kms:Decrypt AWS KMS key

⚠ Important

Wenn Sie einen AWS KMS Discovery-Schlüsselbund in einen [Mehrschlüsselbund für die Entschlüsselung aufnehmen, setzt der Discovery-Schlüsselbund](#) alle Schlüsseleinschränkungen außer Kraft, die durch andere Schlüsselbunde im KMS Mehrfachschlüsselbund festgelegt wurden. Der Mehrfachschlüsselbund verhält sich wie sein am wenigsten restriktiver Schlüsselbund. Ein AWS KMS Discovery-Schlüsselbund hat keine Auswirkung auf die Verschlüsselung, wenn er alleine oder in einem Mehrfachschlüsselbund verwendet wird.

Der AWS Encryption SDK bietet der Einfachheit halber einen AWS KMS Discovery-Schlüsselbund. Wir empfehlen jedoch aus folgenden Gründen, dass Sie nach Möglichkeit einen beschränkteren Schlüsselbund verwenden.

- **Authentizität** — Ein AWS KMS Discovery-Schlüsselbund kann jeden Schlüsselbund verwenden AWS KMS key , der zum Verschlüsseln eines Datenschlüssels in der verschlüsselten Nachricht verwendet wurde, nur damit der Anrufer berechtigt ist, diesen Schlüssel zum Entschlüsseln zu verwenden. AWS KMS key Dies ist möglicherweise nicht der AWS KMS key , den der Aufrufer verwenden möchte. Beispielsweise könnte einer der verschlüsselten Datenschlüssel unter einem weniger sicheren Verfahren verschlüsselt worden sein AWS KMS key , das jeder verwenden kann.
- **Latenz und Leistung** — Ein AWS KMS Discovery-Schlüsselbund ist möglicherweise merklich langsamer als andere Schlüsselbunde, da er AWS Encryption SDK versucht, alle verschlüsselten Datenschlüssel zu entschlüsseln, einschließlich der Schlüssel, die AWS KMS keys in anderen Regionen verschlüsselt wurden, AWS-Konten und der Anrufer nicht berechtigt ist, diese für AWS KMS keys die Entschlüsselung zu verwenden.

Wenn Sie einen Discovery-Schlüsselbund verwenden, empfehlen wir die Verwendung eines Discovery-Filters, um die Schlüssel, die verwendet werden können, auf die KMS Schlüssel in bestimmten Partitionen zu beschränken. [AWS-Konten](#) Discovery-Filter werden in den Versionen 1.7 unterstützt. x und später von AWS Encryption SDK. Hilfe bei der Suche nach Ihrer Konto-ID und Partition finden [Sie unter Ihre AWS-Konto Identifikatoren](#) und Ihr [ARNFormat](#) im Allgemeine AWS-Referenz.

Der folgende Code instanziiert einen AWS KMS Discovery-Schlüsselbund mit einem Erkennungsfilter, der die Schlüssel, die er verwenden AWS Encryption SDK kann, auf die KMS Schlüssel in der aws Partition und im Beispielkonto 111122223333 beschränkt.

Bevor Sie diesen Code verwenden, ersetzen Sie die Beispiel AWS-Konto - und Partitionswerte durch gültige Werte für Ihre Partition und. AWS-Konto Wenn sich Ihre KMS Schlüssel in Regionen Chinas befinden, verwenden Sie den aws-cn Partitionswert. Wenn sich Ihre KMS Schlüssel darin befinden AWS GovCloud (US) Regions, verwenden Sie den aws-us-gov Partitionswert. Verwenden Sie für alle anderen AWS-Regionen den aws Partitionswert.

C

Ein vollständiges Beispiel finden Sie unter [kms_discovery.cpp](#).

```
std::shared_ptr<KmsKeyring::> discovery_filter(
    KmsKeyring::DiscoveryFilter::Builder("aws")
        .AddAccount("111122223333")
        .Build());

struct aws_cryptosdk_keyring *kms_discovery_keyring =
    Aws::Cryptosdk::KmsKeyring::Builder()
        .BuildDiscovery(discovery_filter);
```

C# / .NET

Das folgende Beispiel verwendet Version 4. x des AWS Encryption SDK Forums. NET.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk = new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

List<string> account = new List<string> { "111122223333" };

// In a discovery keyring, you specify an AWS KMS client and a discovery filter,
```

```
// but not a AWS KMS key
var kmsDiscoveryKeyringInput = new CreateAwsKmsDiscoveryKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    DiscoveryFilter = new DiscoveryFilter()
    {
        AccountIds = account,
        Partition = "aws"
    }
};

var kmsDiscoveryKeyring =
    materialProviders.CreateAwsKmsDiscoveryKeyring(kmsDiscoveryKeyringInput);
```

JavaScript Browser

JavaScriptIn müssen Sie die Discovery-Eigenschaft explizit angeben.

```
const clientProvider = getClient(KMS, { credentials })

const discovery = true
const keyring = new KmsKeyringBrowser(clientProvider, {
    discovery,
    discoveryFilter: { accountIDs: [111122223333], partition: 'aws' }
})
```

JavaScript Node.js

JavaScriptIn müssen Sie die Discovery-Eigenschaft explizit angeben.

```
const discovery = true

const keyring = new KmsKeyringNode({
    discovery,
    discoveryFilter: { accountIDs: ['111122223333'], partition: 'aws' }
})
```

Java

```
// Create discovery filter
DiscoveryFilter discoveryFilter = DiscoveryFilter.builder()
    .partition("aws")
    .accountIds(111122223333)
```

```
        .build();
// Create the discovery keyring
CreateAwsKmsMrkDiscoveryMultiKeyringInput createAwsKmsMrkDiscoveryMultiKeyringInput
    = CreateAwsKmsMrkDiscoveryMultiKeyringInput.builder()
        .discoveryFilter(discoveryFilter)
        .build();
IKeyring decryptKeyring =
    matProv.CreateAwsKmsMrkDiscoveryMultiKeyring(createAwsKmsMrkDiscoveryMultiKeyringInput);
```

Verwenden Sie einen AWS KMS regionalen Discovery-Schlüsselbund

Ein AWS KMS Regional Discovery-Schlüsselbund ist ein Schlüsselbund, bei dem ARNs die Schlüssel nicht angegeben sind. KMS Stattdessen ermöglicht es die Entschlüsselung AWS Encryption SDK , indem nur die KMS Schlüssel verwendet werden. AWS-Regionen

Bei der Entschlüsselung mit einem AWS KMS regionalen Discovery-Schlüsselbund werden alle verschlüsselten Datenschlüssel AWS Encryption SDK entschlüsselt, die unter einem AWS KMS key der angegebenen Schlüssel verschlüsselt wurden. AWS-Region Um erfolgreich zu sein, muss der Aufrufer über `kms:Decrypt` Berechtigungen für mindestens einen der angegebenen Schlüssel verfügen AWS-Region , AWS KMS keys der einen Datenschlüssel verschlüsselt hat.

Wie andere Discovery-Schlüsselringe hat auch der regionale Discovery-Schlüsselbund keine Auswirkung auf die Verschlüsselung. Er funktioniert nur beim Entschlüsseln verschlüsselter Nachrichten. Wenn Sie einen Regional Discovery-Schlüsselbund in einem Schlüsselbund mit mehreren Schlüsseln verwenden, der zum Verschlüsseln und Entschlüsseln verwendet wird, ist dieser nur beim Entschlüsseln wirksam. Wenn Sie einen Schlüsselbund für die Erkennung mehrerer Regionen verwenden, um Daten allein oder in einem Schlüsselbund zu verschlüsseln, schlägt der Verschlüsselungsvorgang fehl.

Important

Wenn Sie einen Schlüsselbund für die AWS KMS regionale Erkennung in einen Schlüsselbund für die Entschlüsselung mit mehreren Schlüsseln aufnehmen, setzt der Schlüsselbund für die regionale Erkennung [alle Schlüsseleinschränkungen außer Kraft, die durch andere Schlüsselbunde im Mehrfachschlüsselbund festgelegt wurden](#). KMS Der Mehrfachschlüsselbund verhält sich wie sein am wenigsten restriktiver Schlüsselbund. Ein AWS KMS Discovery-Schlüsselbund hat keine Auswirkung auf die Verschlüsselung, wenn er alleine oder in einem Mehrfachschlüsselbund verwendet wird.

Der regionale Discovery-Schlüsselbund bei AWS-Verschlüsselungs-SDK for C Versuchen, nur mit KMS Schlüsseln in der angegebenen Region zu entschlüsseln. Wenn Sie einen Discovery-Schlüsselbund im und für verwenden. AWS-Verschlüsselungs-SDK for JavaScript AWS Encryption SDK NET, konfigurieren Sie die Region auf dem AWS KMS Client. Bei diesen AWS Encryption SDK Implementierungen AWS KMS werden KMS Schlüssel nicht nach Region gefiltert, aber eine Entschlüsselungsanforderung für KMS Schlüssel außerhalb der angegebenen Region schlägt fehl.

Wenn Sie einen Discovery-Schlüsselbund verwenden, empfehlen wir die Verwendung eines Discovery-Filters, um die bei der Entschlüsselung verwendeten KMS Schlüssel auf die Schlüssel in den angegebenen Partitionen zu beschränken. AWS-Konten Discovery-Filter werden in den Versionen 1.7 unterstützt. x und später von AWS Encryption SDK.

Der folgende Code erstellt beispielsweise einen AWS KMS regionalen Discovery-Schlüsselbund mit einem Discovery-Filter. Dieser Schlüsselbund beschränkt die Anzahl der AWS Encryption SDK KMS Schlüssel im Konto 111122223333 in der Region USA West (Oregon) (us-west-2).

C

Weitere Informationen zum Anzeigen dieses Schlüsselbunds und der `create_kms_client`-Methode in einem funktionierenden Beispiel finden Sie unter [kms_discovery.cpp](#).

```
std::shared_ptr<KmsKeyring::DiscoveryFilter> discovery_filter(  
    KmsKeyring::DiscoveryFilter::Builder("aws")  
        .AddAccount("111122223333")  
        .Build());  
  
struct aws_cryptosdk_keyring *kms_regional_keyring =  
    Aws::Cryptosdk::KmsKeyring::Builder()  
  
        .WithKmsClient(create_kms_client(Aws::Region::US_WEST_2)).BuildDiscovery(discovery_filter))
```

C# / .NET

AWS Encryption SDK Das für. NET hat keinen speziellen Regional Discovery-Schlüsselanhänger. Sie können jedoch verschiedene Techniken verwenden, um die bei der Entschlüsselung verwendeten KMS Schlüssel auf eine bestimmte Region zu beschränken.

Die effizienteste Methode, die Regionen in einem Discovery-Schlüsselbund einzuschränken, ist die Verwendung eines Discovery-Schlüsselbunds, der mehrere Regionen berücksichtigt, auch wenn Sie die Daten nur mit Schlüsseln aus einer Region verschlüsselt haben. Wenn Schlüssel für

einzelne Regionen gefunden werden, verwendet der Schlüsselbund für mehrere Regionen keine Funktionen für mehrere Regionen.

Der von der `CreateAwsKmsMrkDiscoveryKeyring()` Methode zurückgegebene Schlüsselbund filtert Schlüssel vor dem Aufruf nach Region. KMS AWS KMS Sie sendet AWS KMS nur dann eine Entschlüsselungsanforderung an, wenn der verschlüsselte Datenschlüssel mit einem KMS Schlüssel in der Region verschlüsselt wurde, die durch den Region Parameter im Objekt angegeben wurde. `CreateAwsKmsMrkDiscoveryKeyringInput`

In den folgenden Beispielen wird Version 4 verwendet. x der AWS Encryption SDK Form. NET.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk = new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

List<string> account = new List<string> { "111122223333" };

// Create the discovery filter
var filter = DiscoveryFilter = new DiscoveryFilter
{
    AccountIds = account,
    Partition = "aws"
};

var regionalDiscoveryKeyringInput = new CreateAwsKmsMrkDiscoveryKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(RegionEndpoint.USWest2),
    Region = RegionEndpoint.USWest2,
    DiscoveryFilter = filter
};

var kmsRegionalDiscoveryKeyring =
    materialProviders.CreateAwsKmsMrkDiscoveryKeyring(regionalDiscoveryKeyringInput);
```

Sie können KMS Schlüssel auch auf einen bestimmten Wert beschränken, AWS-Region indem Sie in Ihrer Instanz des AWS KMS Clients ([AmazonKeyManagementServiceClient](#)) eine Region angeben. Diese Konfiguration ist jedoch weniger effizient und potenziell kostspieliger als die Verwendung eines Discovery-Schlüsselbundes, der mehrere Regionen berücksichtigt. Anstatt die KMS Schlüssel vor dem Anruf AWS KMS nach Region zu filtern, verwenden Sie das für. AWS Encryption SDK NETruft AWS KMS nach jedem verschlüsselten Datenschlüssel auf (bis er einen

entschlüsselt) und beschränkt sich darauf AWS KMS , die verwendeten KMS Schlüssel auf die angegebene Region zu beschränken.

Im folgenden Beispiel wird Version 4 verwendet. x der AWS Encryption SDK Form. NET.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk = new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

List<string> account = new List<string> { "111122223333" };

// Create the discovery filter,
// but not a AWS KMS key
var createRegionalDiscoveryKeyringInput = new CreateAwsKmsDiscoveryKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(RegionEndpoint.USWest2),
    DiscoveryFilter = new DiscoveryFilter()
    {
        AccountIds = account,
        Partition = "aws"
    }
};

var kmsRegionalDiscoveryKeyring =
    materialProviders.CreateAwsKmsDiscoveryKeyring(createRegionalDiscoveryKeyringInput);
```

JavaScript Browser

```
const clientProvider = getClient(KMS, { credentials })

const discovery = true
const clientProvider = limitRegions(['us-west-2'], getKmsClient)
const keyring = new KmsKeyringBrowser(clientProvider, {
    discovery,
    discoveryFilter: { accountIDs: ['111122223333'], partition: 'aws' }
})
```

JavaScript Node.js

Weitere Informationen zum Anzeigen dieses Schlüsselbunds und der `limitRegions`-Funktion in einem funktionierenden Beispiel finden Sie unter [kms_regional_discovery.ts](#).

```
const discovery = true
```

```
const clientProvider = limitRegions(['us-west-2'], getKmsClient)
const keyring = new KmsKeyringNode({
  clientProvider,
  discovery,
  discoveryFilter: { accountIDs: ['111122223333'], partition: 'aws' }
})
```

Java

```
// Create the discovery filter
DiscoveryFilter discoveryFilter = DiscoveryFilter.builder()
    .partition("aws")
    .accountIds(111122223333)
    .build();
// Create the discovery keyring
CreateAwsKmsMrkDiscoveryMultiKeyringInput createAwsKmsMrkDiscoveryMultiKeyringInput
= CreateAwsKmsMrkDiscoveryMultiKeyringInput.builder()
    .discoveryFilter(discoveryFilter)
    .regions("us-west-2")
    .build();
IKeyring decryptKeyring =
    matProv.CreateAwsKmsMrkDiscoveryMultiKeyring(createAwsKmsMrkDiscoveryMultiKeyringInput);
```

Das exportiert AWS-Verschlüsselungs-SDK for JavaScript auch eine `excludeRegions` Funktion für Node.js und den Browser. Diese Funktion erstellt einen AWS KMS Regional Discovery-Schlüsselbund, der bestimmte Regionen AWS KMS keys auslöst. Im folgenden Beispiel wird ein AWS KMS regionaler Discovery-Schlüsselbund erstellt, der AWS KMS keys im Konto 111122223333 in allen Konten AWS-Region außer US East (Nord-Virginia) (`us-east-1`) verwendet werden kann.

Für AWS-Verschlüsselungs-SDK for C gibt es keine analoge Methode, aber Sie können eine implementieren, indem Sie eine benutzerdefinierte Methode erstellen. [ClientSupplier](#)

Dieses Beispiel zeigt den Code für Node.js.

```
const discovery = true
const clientProvider = excludeRegions(['us-east-1'], getKmsClient)
const keyring = new KmsKeyringNode({
  clientProvider,
  discovery,
  discoveryFilter: { accountIDs: [111122223333], partition: 'aws' }
})
```

AWS KMS Hierarchische Schlüsselanhänger

Important

Der AWS KMS hierarchische Schlüsselbund wird nur von Version 4 unterstützt. x des Forums AWS Encryption SDK . NETund Version 3. x der AWS-Verschlüsselungs-SDK for Java.

Mit dem AWS KMS hierarchischen Schlüsselbund können Sie Ihre kryptografischen Materialien mit einem symmetrischen KMS Verschlüsselungsschlüssel schützen, ohne AWS KMS jedes Mal anrufen zu müssen, wenn Sie Daten ver- oder entschlüsseln. Es ist eine gute Wahl für Anwendungen, bei denen die Anzahl der Aufrufe minimiert werden muss AWS KMS, und für Anwendungen, die kryptografisches Material wiederverwenden können, ohne ihre Sicherheitsanforderungen zu verletzen.

Der hierarchische Schlüsselbund ist eine Caching-Lösung für kryptografisches Material, die die Anzahl der AWS KMS Aufrufe reduziert, indem AWS KMS geschützte Branch-Schlüssel verwendet werden, die in einer Amazon DynamoDB-Tabelle gespeichert sind, und anschließend das für Ver- und Entschlüsselungsvorgänge verwendete Zweigschlüsselmaterial lokal zwischenspeichert. Die DynamoDB-Tabelle dient als Branch-Schlüsselspeicher, der Branch-Schlüssel verwaltet und schützt. Sie speichert den aktiven Branch-Schlüssel und alle vorherigen Versionen des Branch-Schlüssels. Der aktive Zweigschlüssel ist die neueste Version des Zweigschlüssels. Der hierarchische Schlüsselbund verwendet einen eindeutigen Datenschlüssel, um jede Nachricht zu verschlüsseln, und verschlüsselt jeden Datenschlüssel mit einem eindeutigen Umschließungsschlüssel, der vom aktiven Zweigschlüssel abgeleitet wird. Der hierarchische Schlüsselbund hängt von der Hierarchie ab, die zwischen aktiven Zweigschlüsseln und ihren abgeleiteten Umschließungsschlüsseln festgelegt wurde.

Der hierarchische Schlüsselbund verwendet in der Regel jede Version des Zweigschlüssels, um mehrere Anfragen zu erfüllen. Sie kontrollieren jedoch, in welchem Umfang aktive Zweigschlüssel wiederverwendet werden, und bestimmen, wie oft der aktive Zweigschlüssel rotiert wird. Die aktive Version des Abzweigschlüssels bleibt aktiv, bis Sie [ihn drehen](#). Frühere Versionen des aktiven Zweigschlüssels werden nicht zur Ausführung von Verschlüsselungsvorgängen verwendet, sie können jedoch weiterhin abgefragt und bei Entschlüsselungsvorgängen verwendet werden.

Wenn Sie den hierarchischen Schlüsselbund instanziiieren, erstellt er einen lokalen Cache. Sie geben ein [Cache-Limit](#) an, das die maximale Zeitspanne definiert, für die die Branch-Schlüsselmaterialien im lokalen Cache gespeichert werden, bevor sie ablaufen und aus dem Cache entfernt werden.

Der hierarchische Schlüsselbund führt einen AWS KMS Aufruf durch, um den Zweigsschlüssel zu entschlüsseln und die Zweigsschlüsselmaterialien zusammenzustellen, wenn `branch-key-id` zum ersten Mal in einem Vorgang angegeben wird. Anschließend werden die Materialien der Verzweigungsschlüssel im lokalen Cache gespeichert und für alle Verschlüsselungs- und Entschlüsselungsvorgänge, die dies spezifizieren, wiederverwendet, bis das Cache-Limit abläuft. Das Speichern von Zweigsschlüsselmaterialien im lokalen Cache reduziert die Anzahl der Aufrufe. AWS KMS Stellen Sie sich beispielsweise ein Cache-Limit von 15 Minuten vor. Wenn Sie 10.000 Verschlüsselungsvorgänge innerhalb dieses Cache-Limits ausführen, müsste der [herkömmliche AWS KMS Schlüsselbund](#) 10.000 AWS KMS Aufrufe tätigen, um 10.000 Verschlüsselungsvorgänge zu erfüllen. Wenn Sie einen aktiven Schlüsselbund haben `branch-key-id`, muss der hierarchische Schlüsselbund nur einen AWS KMS Aufruf tätigen, um 10.000 Verschlüsselungsvorgänge abzuwickeln.

Der lokale Cache besteht aus zwei Partitionen, einer für Verschlüsselungsvorgänge und einer zweiten für Entschlüsselungsvorgänge. Die Verschlüsselungspartition speichert die aus dem aktiven Zweigsschlüssel zusammengestellten Materialien und verwendet sie für alle Verschlüsselungsvorgänge wieder, bis das Cache-Limit abläuft. In der Entschlüsselungspartition werden die Materialien für Zweigsschlüssel gespeichert, die für andere Versionen von Zweigsschlüsseln zusammengestellt wurden, die bei Entschlüsselungsvorgängen identifiziert wurden. In der Entschlüsselungspartition können mehrere aktive Versionen von Zweigsschlüsselmaterialien gleichzeitig gespeichert werden. Wenn sie so konfiguriert ist, dass sie einen Branch-Schlüssel-ID-Anbieter für eine Umgebung mit mehreren Mandanten verwendet, kann die verschlüsselte Partition auch mehrere Versionen von Branch-Schlüsselmaterialien gleichzeitig speichern. Weitere Informationen finden Sie unter [Verwenden des hierarchischen Schlüsselbunds in Umgebungen mit mehreren Mandanten](#).

Note

Alle Erwähnungen des hierarchischen Schlüsselbundes in der AWS Encryption SDK beziehen sich auf den hierarchischen Schlüsselbund. AWS KMS

Themen

- [Funktionsweise](#)
- [Voraussetzungen](#)
- [Erstellen Sie einen hierarchischen Schlüsselbund](#)
- [Drehe deinen aktiven Filialschlüssel](#)

- [Verwenden des hierarchischen Schlüsselbunds in Umgebungen mit mehreren Mandanten](#)

Funktionsweise

In den folgenden exemplarischen Vorgehensweisen wird beschrieben, wie der hierarchische Schlüsselbund Verschlüsselungs- und Entschlüsselungsmaterialien zusammenstellt und welche verschiedenen Aufrufe der Schlüsselbund für Verschlüsselungs- und Entschlüsselungsvorgänge vornimmt. [Technische Einzelheiten zur Ableitung von Schlüsseln und zur Verschlüsselung von Klartext-Datenschlüsseln finden Sie unter Technische Details zum hierarchischen Schlüsselbund.AWS KMS](#)

Verschlüsseln und signieren

In der folgenden exemplarischen Vorgehensweise wird beschrieben, wie der hierarchische Schlüsselbund Verschlüsselungsmaterialien zusammenstellt und daraus einen eindeutigen Umschließungsschlüssel ableitet.

1. Die Verschlüsselungsmethode fragt den hierarchischen Schlüsselbund nach Verschlüsselungsmaterialien. Der Schlüsselbund generiert einen Klartext-Datenschlüssel und überprüft dann, ob sich im lokalen Cache gültiges Verzweigungsmaterial für die Generierung des Wrapping-Schlüssels befindet. Wenn gültiges Schlüsselmaterial für die Zweige vorhanden ist, fährt der Schlüsselbund mit Schritt 4 fort.
2. Wenn kein gültiges Material für Zweigschlüssel vorhanden ist, fragt der hierarchische Schlüsselbund den Schlüsselspeicher der Filiale nach dem aktiven Zweigschlüssel ab.
 - a. Der Branch-Schlüsselspeicher ruft AWS KMS auf, um den aktiven Branch-Schlüssel zu entschlüsseln, und gibt den aktiven Branch-Schlüssel im Klartext zurück. Daten, die den aktiven Zweigschlüssel identifizieren, werden serialisiert, um zusätzliche authentifizierte Daten (AAD) beim Entschlüsselungsaufruf von bereitzustellen. AWS KMS
 - b. Der Branch-Schlüsselspeicher gibt den Klartext-Zweigschlüssel und Daten, die ihn identifizieren, wie z. B. die Version des Zweigschlüssels, zurück.
3. Der hierarchische Schlüsselbund stellt die Materialien der Verzweigungsschlüssel (die Klartext-Version für den Zweigschlüssel und die Zweigschlüsselversion) zusammen und speichert eine Kopie davon im lokalen Cache.
4. Der hierarchische Schlüsselbund leitet aus dem Klartext-Verzweigungsschlüssel und einem 16-Byte-Zufallssalz einen eindeutigen Umbruchschlüssel ab. Er verwendet den abgeleiteten Umschließungsschlüssel, um eine Kopie des Klartext-Datenschlüssels zu verschlüsseln.

Die Verschlüsselungsmethode verwendet die Verschlüsselungsmaterialien, um die Daten zu verschlüsseln. Weitere Informationen finden Sie unter [So werden AWS Encryption SDK Daten verschlüsselt](#).

Entschlüsseln und verifizieren

In der folgenden exemplarischen Vorgehensweise wird beschrieben, wie der hierarchische Schlüsselbund Entschlüsselungsmaterialien zusammenstellt und den verschlüsselten Datenschlüssel entschlüsselt.

1. Die Entschlüsselungsmethode identifiziert den verschlüsselten Datenschlüssel aus der verschlüsselten Nachricht und leitet ihn an den hierarchischen Schlüsselbund weiter.
2. Der hierarchische Schlüsselbund deserialisiert Daten, die den verschlüsselten Datenschlüssel identifizieren, einschließlich der Version des Zweigsschlüssels, des 16-Byte-Salts und anderer Informationen, die beschreiben, wie der Datenschlüssel verschlüsselt wurde.

Weitere Informationen finden Sie unter [AWS KMS Technische Details zum hierarchischen Schlüsselbund](#).

3. Mit dem hierarchischen Schlüsselbund wird geprüft, ob sich im lokalen Cache gültiges Zweigsschlüsselmaterial befindet, das mit der in Schritt 2 identifizierten Version des Zweigsschlüssels übereinstimmt. Wenn gültiges Schlüsselmaterial für die Zweige vorhanden ist, fährt der Schlüsselbund mit Schritt 6 fort.
4. Wenn kein gültiges Material für Zweigsschlüssel vorhanden ist, fragt der hierarchische Schlüsselbund den Zweigsschlüsselspeicher nach dem Zweigsschlüssel ab, der mit der in Schritt 2 identifizierten Version des Zweigsschlüssels übereinstimmt.
 - a. Der Branch-Schlüsselspeicher ruft AWS KMS auf, um den Branch-Schlüssel zu entschlüsseln, und gibt den aktiven Branch-Schlüssel im Klartext zurück. Daten, die den aktiven Zweigsschlüssel identifizieren, werden serialisiert, um zusätzliche authentifizierte Daten (AAD) beim Entschlüsselungsaufwurf von bereitzustellen. AWS KMS
 - b. Der Branch-Schlüsselspeicher gibt den Klartext-Zweigsschlüssel und Daten, die ihn identifizieren, wie z. B. die Version des Zweigsschlüssels, zurück.
5. Der hierarchische Schlüsselbund stellt die Materialien der Verzweigungsschlüssel (die Klartext-Version für den Zweigsschlüssel und die Zweigsschlüsselversion) zusammen und speichert eine Kopie davon im lokalen Cache.

6. Der hierarchische Schlüsselbund verwendet die zusammengestellten Zweigschlüsselmaterialien und das in Schritt 2 identifizierte 16-Byte-Salt, um den eindeutigen Wrapping-Schlüssel zu reproduzieren, mit dem der Datenschlüssel verschlüsselt wurde.
7. Der hierarchische Schlüsselbund verwendet den reproduzierten Wrapping-Schlüssel, um den Datenschlüssel zu entschlüsseln, und gibt den Klartext-Datenschlüssel zurück.

Die Entschlüsselungsmethode verwendet die Entschlüsselungsmaterialien und den Klartext-Datenschlüssel, um die verschlüsselte Nachricht zu entschlüsseln. Weitere Informationen finden Sie unter [So AWS Encryption SDK entschlüsselt der](#) eine verschlüsselte Nachricht.

Voraussetzungen

Für das AWS Encryption SDK ist kein AWS-Konto und es hängt auch von keinem AWS -Service ab. Der hierarchische Schlüsselbund hängt jedoch von Amazon DynamoDB AWS KMS ab.

[Um einen hierarchischen Schlüsselbund verwenden zu können, benötigen Sie eine symmetrische Verschlüsselung mit kms:Decrypt-Berechtigungen. AWS KMS key Sie können auch einen Schlüssel für mehrere Regionen mit symmetrischer Verschlüsselung verwenden.](#) Ausführliche Informationen zu den Berechtigungen für AWS KMS keys finden Sie unter [Authentifizierung und Zugriffskontrolle](#) im AWS Key Management Service Entwicklerhandbuch.

Bevor Sie einen hierarchischen Schlüsselbund erstellen und verwenden können, müssen Sie Ihren Branch-Schlüsselspeicher erstellen und ihn mit Ihrem ersten aktiven Branch-Schlüssel auffüllen.

Schritt 1: Konfigurieren Sie einen neuen Schlüsselspeicherdienst

Der Schlüsselspeicherdienst bietet verschiedene API Operationen, z. B. `CreateKeyStore` und `CreateKey`, mit denen Sie die Voraussetzungen für den hierarchischen Schlüsselbund zusammenstellen und Ihren Branch-Schlüsselspeicher verwalten können.

Im folgenden Beispiel wird ein Schlüsselspeicherdienst erstellt. Sie müssen einen DynamoDB-Tabellennamen angeben, der als Name Ihres Branch-Schlüsselspeichers dient, einen logischen Namen für den Branch-Schlüsselspeicher und den KMS Schlüssel, der den Schlüssel identifiziertARN, der Ihre KMS Branch-Schlüssel schützt.

Der Name des logischen Schlüsselspeichers ist kryptografisch an alle in der Tabelle gespeicherten Daten gebunden, um DynamoDB-Wiederherstellungsvorgänge zu vereinfachen. Der Name des logischen Schlüsselspeichers kann mit Ihrem DynamoDB-Tabellennamen identisch sein, muss es aber nicht. Wir empfehlen, Ihren DynamoDB-Tabellennamen als

logischen Tabellennamen anzugeben, wenn Sie Ihren Schlüsselspeicherdienst zum ersten Mal konfigurieren. Sie müssen immer denselben logischen Tabellennamen angeben. Falls sich der Name Ihres Branch-Schlüsselspeichers nach der [Wiederherstellung Ihrer DynamoDB-Tabelle aus einer Sicherung](#) ändert, wird der Name des logischen Schlüsselspeichers dem von Ihnen angegebenen DynamoDB-Tabellennamen zugeordnet, um sicherzustellen, dass der hierarchische Schlüsselbund weiterhin auf Ihren Branch-Schlüsselspeicher zugreifen kann.

Note

Der Name des logischen Schlüsselspeichers ist im Verschlüsselungskontext aller Key-Store-Dienstoperationen enthalten, die aufrufen. API AWS KMS Der Verschlüsselungskontext ist nicht geheim, seine Werte — einschließlich des Namens des logischen Schlüsselspeichers — erscheinen im Klartext in Protokollen. AWS CloudTrail

C# / .NET

```
var kmsConfig = new KMSConfiguration { KmsKeyArn = kmsKeyArn };
var keystoreConfig = new KeyStoreConfig
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsConfiguration = kmsConfig,
    DdbTableName = keyStoreName,
    DdbClient = new AmazonDynamoDBClient(),
    LogicalKeyName = logicalKeyName
};
var keystore = new KeyStore(keystoreConfig);
```

Java

```
final KeyStore keystore = KeyStore.builder().KeyStoreConfig(
    KeyStoreConfig.builder()
        .ddbClient(DynamoDbClient.create())
        .ddbTableName(keyStoreName)
        .logicalKeyName(logicalKeyName)
        .kmsClient(KmsClient.create())
        .kmsConfiguration(KMSConfiguration.builder()
            .kmsKeyArn(kmsKeyArn)
            .build())
        .build()).build();
```

Schritt 2: Rufen **CreateKeyStore** Sie auf, um einen Branch-Schlüsselspeicher zu erstellen

Mit dem folgenden Vorgang wird der Branch-Schlüsselspeicher erstellt, der Ihre Filialschlüssel dauerhaft schützt.

C# / .NET

```
var createKeyStoreOutput = keystore.CreateKeyStore(new CreateKeyStoreInput());
```

Java

```
keystore.CreateKeyStore(CreateKeyStoreInput.builder().build());
```

Der `CreateKeyStore` Vorgang erstellt eine DynamoDB-Tabelle mit dem Tabellennamen, den Sie in Schritt 1 angegeben haben, und den folgenden erforderlichen Werten.

	Partitionsschlüssel	Sortierschlüssel
Basistabelle	branch-key-id	type

Note

Sie können die DynamoDB-Tabelle, die als Ihr Branch-Schlüsselspeicher dient, manuell erstellen, anstatt den `CreateKeyStore` Vorgang zu verwenden. Wenn Sie den Branch-Schlüsselspeicher manuell erstellen möchten, müssen Sie die folgenden Zeichenfolgenwerte für die Partitions- und Sortierschlüssel angeben:

- Partitionsschlüssel: branch-key-id
- Sortierschlüssel: type

Schritt 3: Rufen Sie **CreateKey** auf, um einen neuen aktiven Branch-Schlüssel zu erstellen

Der folgende Vorgang erstellt einen neuen aktiven Verzweigungsschlüssel unter Verwendung des KMS Schlüssels, den Sie in Schritt 1 angegeben haben, und fügt den aktiven Verzweigungsschlüssel zu der DynamoDB-Tabelle hinzu, die Sie in Schritt 2 erstellt haben.

Wenn Sie aufrufen `CreateKey`, können Sie wählen, ob Sie die folgenden optionalen Werte angeben möchten.

- Schlüssel-ID für den Zweig: definiert einen benutzerdefinierten Schlüsselbranch-key-id.

Um einen benutzerdefinierten zu erstellenbranch-key-id, müssen Sie dem encryptionContext Parameter auch einen zusätzlichen Verschlüsselungskontext hinzufügen.

- [Verschlüsselungskontext: Definiert einen optionalen Satz nicht geheimer Schlüssel-Wert-Paare, der zusätzliche authentifizierte Daten \(AAD\) in dem Verschlüsselungskontext bereitstellt, der im kms: -Aufruf enthalten ist. GenerateDataKeyWithoutPlaintext](#)

Dieser zusätzliche Verschlüsselungskontext wird mit dem Präfix angezeigt. aws-crypto-ec:

C# / .NET

```
var additionalEncryptionContext = new Dictionary<string, string>();
additionalEncryptionContext.Add("Additional Encryption Context for", "custom
branch key id");

var branchKeyId = keystore.CreateKey(new CreateKeyInput
{
    BranchKeyIdentifier = "custom-branch-key-id", // OPTIONAL
    EncryptionContext = additionalEncryptionContext // OPTIONAL
});
```

Java

```
final Map<String, String> additionalEncryptionContext =
    Collections.singletonMap("Additional Encryption Context for",
        "custom branch key id");

final String BranchKey = keystore.CreateKey(
    CreateKeyInput.builder()
        .branchKeyIdentifier("custom-branch-key-id") //OPTIONAL
        .encryptionContext(additionalEncryptionContext) //OPTIONAL
        .build()).branchKeyIdentifier();
```

Zunächst generiert die CreateKey Operation die folgenden Werte.

- Ein [Universally Unique Identifier](#) (UUID) der Version 4 für branch-key-id (sofern Sie keinen benutzerdefinierten Bezeichner angegeben habenbranch-key-id).
- Eine Version 4 UUID für die Branch Key-Version
- A timestamp im [Datums- und Uhrzeitformat ISO 8601](#) in koordinierter Weltzeit (UTC).

Anschließend ruft der CreateKey Vorgang [kms: GenerateDataKeyWithoutPlaintext](#) mit der folgenden Anforderung auf.

```
{
  "EncryptionContext": {
    "branch-key-id" : "branch-key-id",
    "type" : "type",
    "create-time" : "timestamp",
    "logical-key-store-name" : "the logical table name for your branch key store",
    "kms-arn" : the KMS key ARN,
    "hierarchy-version" : "1",
    "aws-crypto-ec:contextKey" : "contextValue"
  },
  "KeyId": "the KMS key ARN you specified in Step 1",
  "NumberOfBytes": "32"
}
```

Als Nächstes ruft der CreateKey Vorgang [kms: ReEncrypt](#) auf, um einen aktiven Datensatz für den Branch-Schlüssel zu erstellen, indem der Verschlüsselungskontext aktualisiert wird.

Zuletzt ruft der CreateKey Vorgang [ddb: TransactWriteItems](#) auf, um ein neues Element zu schreiben, das den Verzweigungsschlüssel in der Tabelle, die Sie in Schritt 2 erstellt haben, beibehält. Das Element hat die folgenden Attribute.

```
{
  "branch-key-id" : branch-key-id,
  "type" : "branch:ACTIVE",
  "enc" : the branch key returned by the GenerateDataKeyWithoutPlaintext call,
  "version": "branch:version:the branch key version UUID",
  "create-time" : "timestamp",
  "kms-arn" : "the KMS key ARN you specified in Step 1",
  "hierarchy-version" : "1",
  "aws-crypto-ec:contextKey" : "contextValue"
}
```

Erstellen Sie einen hierarchischen Schlüsselbund

Um den hierarchischen Schlüsselbund zu initialisieren, müssen Sie die folgenden Werte angeben:

- Name eines Zweigschlüsselspeichers

Der Name der DynamoDB-Tabelle, die Sie als Branch-Schlüsselspeicher erstellt haben.

-

Ein Cache-Limit für die Gültigkeitsdauer () TTL

Die Zeitspanne in Sekunden, in der ein Eintrag für Materialien in einem Branch Key innerhalb des lokalen Caches verwendet werden kann, bevor er abläuft. Dieser Wert muss größer als null sein. Wenn das Cache-Limit TTL abläuft, wird der Eintrag aus dem lokalen Cache entfernt.

- Eine Schlüssel-ID für die Verzweigung

Der `branch-key-id`, der den aktiven Filialschlüssel in Ihrem Filialschlüsselspeicher identifiziert.

Note

Um den hierarchischen Schlüsselbund für die Verwendung durch mehrere Mandanten zu initialisieren, müssen Sie anstelle eines einen Lieferanten für die Filialschlüssel-ID angeben. `branch-key-id` Weitere Informationen finden Sie unter [Verwenden des hierarchischen Schlüsselbunds in Umgebungen mit mehreren Mandanten](#).

- (Optional) Ein Cache

Wenn Sie Ihren Cachetyp oder die Anzahl der Einträge für Branch-Schlüsselmaterialien, die im lokalen Cache gespeichert werden können, anpassen möchten, geben Sie den Cachetyp und die Eintragskapazität an, wenn Sie den Schlüsselbund initialisieren.

Der Cachetyp definiert das Threading-Modell. Der hierarchische Schlüsselbund bietet drei Cachetypen, die Umgebungen mit mehreren Mandanten unterstützen: Standard,,. MultiThreaded StormTracking

Wenn Sie keinen Cache angeben, verwendet der hierarchische Schlüsselbund automatisch den Standard-Cachetyp und legt die Eintragskapazität auf 1000 fest.

Default (Recommended)

Für die meisten Benutzer erfüllt der Standard-Cache ihre Threading-Anforderungen. Der Standard-Cache ist so konzipiert, dass er Umgebungen mit hohem Multithreading-Anteil unterstützt. Wenn ein Eintrag für Branch-Schlüsselmaterialien abläuft, verhindert der Standard-Cache den Aufruf mehrerer Threads AWS KMS und Amazon DynamoDB, indem ein Thread 10 Sekunden im Voraus darüber informiert wird, dass der Eintrag für Branch-Schlüsselmaterialien

abläuft. Dadurch wird sichergestellt, dass nur ein Thread eine Anfrage AWS KMS zur Aktualisierung des Caches sendet.

Um Ihren hierarchischen Schlüsselbund mit einem Standard-Cache zu initialisieren, geben Sie den folgenden Wert an:

- **Eintragskapazität:** Schränkt die Anzahl der Einträge für Branch-Schlüsselmaterialien ein, die im lokalen Cache gespeichert werden können.

C#/. NET

```
CacheType defaultCache = new CacheType
{
    Default = new DefaultCache{EntryCapacity = 100}
};
```

Java

```
.cache(CacheType.builder()
    .Default(DefaultCache.builder()
    .entryCapacity(100)
    .build())
```

Default und StormTracking Caches unterstützen dasselbe Threading-Modell, aber Sie müssen nur die Eingabekapazität angeben, um den hierarchischen Schlüsselbund mit dem Standard-Cache zu initialisieren. Für detailliertere Cache-Anpassungen verwenden Sie den Cache.

StormTracking

MultiThreaded

Der MultiThreaded Cache kann sicher in Multithread-Umgebungen verwendet werden, bietet jedoch keine Funktionen zur Minimierung AWS KMS von Amazon DynamoDB DynamoDB-Aufrufen. Daher werden alle Threads gleichzeitig benachrichtigt, wenn ein Eintrag für wichtige Materialien in einer Branche abläuft. Dies kann zu mehreren AWS KMS Aufrufen führen, um den Cache zu aktualisieren.

Um Ihren hierarchischen Schlüsselbund mit einem MultiThreaded Cache zu initialisieren, geben Sie die folgenden Werte an:

- **Eintragskapazität:** Schränkt die Anzahl der Einträge für Branch-Schlüsselmaterialien ein, die im lokalen Cache gespeichert werden können.

- Größe des Endstücks des Eintrags: Definiert die Anzahl der Einträge, die beschnitten werden müssen, wenn die Eingangskapazität erreicht ist.

C#. NET

```
CacheType multithreadedCache = new CacheType
{
    MultiThreaded = new MultiThreadedCache
    {
        EntryCapacity = 100,
        EntryPruningTailSize = 1
    }
};
```

Java

```
.cache(CacheType.builder()
    .MultiThreaded(MultiThreadedCache.builder()
        .entryCapacity(100)
        .entryPruningTailSize(1)
        .build())
```

StormTracking

Der StormTracking Cache ist so konzipiert, dass er Umgebungen mit hohem Multithreading-Anteil unterstützt. Wenn ein Eintrag für Branch-Schlüsselmaterialien abläuft, verhindert der StormTracking Cache, dass mehrere Threads Amazon DynamoDB aufrufen AWS KMS , indem ein Thread im Voraus darüber informiert wird, dass der Eintrag für Branch-Schlüsselmaterialien abläuft. Dadurch wird sichergestellt, dass nur ein Thread eine Anfrage AWS KMS zur Aktualisierung des Caches sendet.

Um Ihren hierarchischen Schlüsselbund mit einem StormTracking Cache zu initialisieren, geben Sie die folgenden Werte an:

- Eintragskapazität: Schränkt die Anzahl der Einträge für Branch-Schlüsselmaterialien ein, die im lokalen Cache gespeichert werden können.
- Größe des Endstücks des Eintrags: Definiert die Anzahl der Einträge für das Schlüsselmaterial der Branche, die gleichzeitig beschnitten werden sollen.

Standardwert: 1 Eintrag

- **Übergangszeit:** Definiert die Anzahl der Sekunden vor Ablauf, nach der versucht wird, die wichtigsten Materialien der Branche zu aktualisieren.

Standardwert: 10 Sekunden

- **Verlängerungsintervall:** Definiert die Anzahl der Sekunden zwischen Versuchen, die Schlüsselmaterialien der Filiale zu aktualisieren.

Standardwert: 1 Sekunde

- **Fan-out:** Definiert die Anzahl der gleichzeitigen Versuche, die wichtigsten Materialien der Filiale zu aktualisieren.

Standardwert: 20 Versuche

- **In flight time to live (TTL):** Definiert die Anzahl der Sekunden, bis beim Versuch, die Schlüsselmaterialien der Filiale zu aktualisieren, eine Zeitüberschreitung eintritt. Jedes Mal, wenn der Cache als Antwort auf eine zurückkehrt `NoSuchEntryGetCacheEntry`, gilt dieser Verzweigungsschlüssel als aktiv, bis derselbe Schlüssel zusammen mit einem `PutCache` Eintrag geschrieben wird.

Standardwert: 20 Sekunden

- **Ruhezustand:** Definiert die Anzahl der Sekunden, die ein Thread in den Ruhezustand versetzen soll, wenn der Wert überschritten `fanOut` wird.

Standardwert: 20 Millisekunden

C#. NET

```
CacheType stormTrackingCache = new CacheType
{
    StormTracking = new StormTrackingCache
    {
        EntryCapacity = 100,
        EntryPruningTailSize = 1,
        FanOut = 20,
        GraceInterval = 1,
        GracePeriod = 10,
        InFlightTTL = 20,
        SleepMilli = 20
    }
};
```

Java

```
.cache(CacheType.builder()
    .MultiThreaded(MultiThreadedCache.builder()
        .entryCapacity(100)
        .entryPruningTailSize(1)
        .gracePeriod(10)
        .graceInterval(1)
        .fanOut(20)
        .inFlightTTL(20)
        .sleepMilli(20)
        .build())
```

- (Optional) Eine Liste von Grant-Tokens

Wenn Sie den Zugriff auf den KMS Schlüssel in Ihrem hierarchischen Schlüsselbund mit [Grants](#) steuern, müssen Sie bei der Initialisierung des Schlüsselbunds alle erforderlichen Grant-Token angeben.

Im folgenden Beispiel wird ein hierarchischer Schlüsselbund mit einem Cache-Limit TLL von 600 Sekunden und einer Eingabekapazität von 1000 initialisiert.

C# / .NET

```
// Instantiate the AWS Encryption SDK and material providers
var mpl = new MaterialProviders(new MaterialProvidersConfig());
var esdk = new ESDK(new AwsEncryptionSdkConfig());

// Instantiate the keyring
var createKeyringInput = new CreateAwsKmsHierarchicalKeyringInput
{
    KeyStore = branchKeyStoreName,
    BranchKeyId = branch-key-id,
    Cache = new CacheType { Default = new DefaultCache { EntryCapacity = 1000 } },
    TtlSeconds = 600
};
```

Java

```
final MaterialProviders matProv = MaterialProviders.builder()
    .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
```

```
        .build();
final CreateAwsKmsHierarchicalKeyringInput keyringInput =
    CreateAwsKmsHierarchicalKeyringInput.builder()
        .keyStore(branchKeyName)
        .branchKeyId(branch-key-id)
        .ttlSeconds(600)
        .cache(CacheType.builder() //OPTIONAL
            .Default(DefaultCache.builder()
                .entryCapacity(1000)
                .build())
            .build())
        .build();
final Keyring hierarchicalKeyring =
    matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);
```

Drehe deinen aktiven Filialschlüssel

Für jeden Filialschlüssel kann es jeweils nur eine aktive Version geben. Der hierarchische Schlüsselbund verwendet in der Regel jede aktive Version des Zweigschlüssels, um mehrere Anfragen zu erfüllen. Sie kontrollieren jedoch, in welchem Umfang aktive Zweigschlüssel wiederverwendet werden, und bestimmen, wie oft der aktive Zweigschlüssel rotiert wird.

Zweigschlüssel werden nicht zur Verschlüsselung von Klartext-Datenschlüsseln verwendet. Sie werden verwendet, um die eindeutigen Wrapping-Schlüssel abzuleiten, mit denen Klartext-Datenschlüssel verschlüsselt werden. Bei der [Ableitung des Wrapping-Schlüssels](#) wird ein einzigartiger 32-Byte-Wrapping-Schlüssel mit 28 Byte Zufälligkeit erzeugt. Das bedeutet, dass aus einem Zweigschlüssel mehr als 79 Oktillionen oder 2^{96} einzigartige Wrapping-Schlüssel abgeleitet werden können, bevor es zu einem kryptografischen Verschleiß kommt. Trotz dieses sehr geringen Risikos der Datener schöpfung müssen Sie Ihre aktiven Filialschlüssel möglicherweise aufgrund von Geschäfts- oder Vertragsbestimmungen oder behördlichen Vorschriften wechseln.

Die aktive Version des Zweigschlüssels bleibt aktiv, bis Sie ihn rotieren. Frühere Versionen des aktiven Zweigschlüssels werden nicht zur Ausführung von Verschlüsselungsvorgängen verwendet und können auch nicht zum Ableiten neuer Umschließungsschlüssel verwendet werden. Sie können jedoch weiterhin abgefragt werden und bieten Umschließungsschlüssel zum Entschlüsseln der Datenschlüssel, die sie verschlüsselt haben, während sie aktiv waren.

Verwenden Sie den Schlüsselspeicherdienst `VersionKey`, um Ihren aktiven Filialschlüssel zu rotieren. Wenn Sie den aktiven Zweigschlüssel rotieren, wird ein neuer Zweigschlüssel erstellt, der die vorherige Version ersetzt. Das `branch-key-id` ändert sich nicht, wenn Sie den aktiven

Abzweigschlüssel drehen. Sie müssen den Schlüssel angeben `branch-key-id`, der den aktuell aktiven Abzweigschlüssel identifiziert, wenn Sie anrufen `VersionKey`.

C# / .NET

```
keystore.VersionKey(new VersionKeyInput{BranchKeyIdentifier = branchKeyId});
```

Java

```
keystore.VersionKey(  
    VersionKeyInput.builder()  
        .branchKeyIdentifier("branch-key-id")  
        .build()  
);
```

Verwenden des hierarchischen Schlüsselbunds in Umgebungen mit mehreren Mandanten

Sie können die zwischen aktiven Zweigschlüsseln und ihren abgeleiteten Umschließungsschlüsseln eingerichtete Schlüsselhierarchie verwenden, um Umgebungen mit mehreren Mandanten zu unterstützen, indem Sie für jeden Mandanten in Ihrer Umgebung einen Verzweigungsschlüssel erstellen. Der hierarchische Schlüsselbund verschlüsselt dann alle Daten für einen bestimmten Mandanten mit seinem eindeutigen Zweigschlüssel. Auf diese Weise können Sie Mandantendaten nach Zweigschlüsseln isolieren.

Jeder Mandant hat seinen eigenen Zweigschlüssel, der durch einen eindeutigen Schlüssel definiert ist `branch-key-id`. Es kann jeweils nur eine aktive Version von `branch-key-id` jeder Version geben.

Bevor Sie Ihren hierarchischen Schlüsselbund für die Verwendung durch mehrere Mandanten initialisieren können, müssen Sie für jeden Mandanten einen Branch-Schlüssel und einen Branch-Schlüssel-ID-Lieferanten erstellen. Verwenden Sie den Anbieter für die Filialschlüssel-ID, um einen benutzerfreundlichen Namen für Sie `branch-key-ids` zu erstellen, damit Sie den richtigen `branch-key-id` Namen für einen Mandanten leichter erkennen können. Mit dem Anzeigenamen können Sie beispielsweise auf einen Filialschlüssel als `tenant1` statt auf `verweisenb3f61619-4d35-48ad-a275-050f87e15122`.

Für Entschlüsselungsvorgänge können Sie entweder einen einzelnen hierarchischen Schlüsselbund statisch konfigurieren, um die Entschlüsselung auf einen einzelnen Mandanten zu beschränken, oder Sie können den Branch-Schlüssel-ID-Anbieter verwenden, um zu ermitteln, welcher Mandant für die Entschlüsselung einer Nachricht verantwortlich ist.

[Folgen Sie zunächst Schritt 1 und Schritt 2 der Verfahren mit den Voraussetzungen.](#) Verwenden Sie dann die folgenden Verfahren, um einen Zweigschlüssel für jeden Mandanten zu erstellen, einen Anbieter für die Zweigschlüssel-ID zu erstellen und Ihren hierarchischen Schlüsselbund für die Verwendung durch mehrere Mandanten zu initialisieren.

Schritt 1: Erstellen Sie einen Zweigschlüssel für jeden Mandanten in Ihrer Umgebung

Rufen Sie `CreateKey` für jeden Mandanten an.

Der folgende Vorgang erstellt zwei Verzweigungsschlüssel unter Verwendung des KMS Schlüssels, den Sie bei der Erstellung Ihres Schlüssel Speicherdienstes angegeben haben, und fügt die Verzweigungsschlüssel zu der DynamoDB-Tabelle hinzu, die Sie als Branch-Schlüsselspeicher erstellt haben. Derselbe KMS Schlüssel muss alle Zweigschlüssel schützen.

C# / .NET

```
var branchKeyId1 = keystore.CreateKey(new CreateKeyInput());
var branchKeyId2 = keystore.CreateKey(new CreateKeyInput());
```

Java

```
CreateKeyOutput branchKeyId1 =
    keystore.CreateKey(CreateKeyInput.builder().build());
CreateKeyOutput branchKeyId2 =
    keystore.CreateKey(CreateKeyInput.builder().build());
```

Schritt 2: Erstellen Sie einen Lieferanten für die Filialschlüssel-ID

Im folgenden Beispiel wird ein Lieferant für die Filialschlüssel-ID erstellt.

C# / .NET

```
var branchKeySupplier =
    new ExampleBranchKeySupplier(branchKeyId1.BranchKeyIdentifier,
    branchKeyId2.BranchKeyIdentifier);
```

Java

```
IBranchKeyIdSupplier branchKeyIdSupplier = new ExampleBranchKeyIdSupplier(
    branchKeyId1.branchKeyIdIdentifier(), branchKeyId2.branchKeyIdIdentifier());
```

Schritt 3: Initialisieren Sie Ihren hierarchischen Schlüsselbund mit dem Branchenschlüssel-ID-Lieferanten

Um den hierarchischen Schlüsselbund zu initialisieren, müssen Sie die folgenden Werte angeben:

- Name eines Zweigschlüsselspeichers
- Ein [Cache-Limit für die Gültigkeitsdauer \(TTL\)](#)
- Ein Lieferant für die Schlüssel-ID der Branche
- (Optional) Ein Cache

Wenn Sie Ihren Cachetyp oder die Anzahl der Einträge für Branch-Schlüsselmaterialien, die im lokalen Cache gespeichert werden können, anpassen möchten, geben Sie den Cachetyp und die Eintragskapazität an, wenn Sie den Schlüsselbund initialisieren.

Der Cachetyp definiert das Threading-Modell. Der hierarchische Schlüsselbund bietet drei Cachetypen, die Umgebungen mit mehreren Mandanten unterstützen: Standard,, MultiThreaded StormTracking

Wenn Sie keinen Cache angeben, verwendet der hierarchische Schlüsselbund automatisch den Standard-Cachetyp und legt die Eintragskapazität auf 1000 fest.

Default (Recommended)

Für die meisten Benutzer erfüllt der Standard-Cache ihre Threading-Anforderungen. Der Standard-Cache ist so konzipiert, dass er Umgebungen mit hohem Multithreading-Anteil unterstützt. Wenn ein Eintrag für Branch-Schlüsselmaterialien abläuft, verhindert der Standard-Cache den Aufruf mehrerer Threads AWS KMS und Amazon DynamoDB, indem ein Thread 10 Sekunden im Voraus darüber informiert wird, dass der Eintrag für Branch-Schlüsselmaterialien abläuft. Dadurch wird sichergestellt, dass nur ein Thread eine Anfrage AWS KMS zur Aktualisierung des Caches sendet.

Um Ihren hierarchischen Schlüsselbund mit einem Standard-Cache zu initialisieren, geben Sie den folgenden Wert an:

- Eintragskapazität: Schränkt die Anzahl der Einträge für Branch-Schlüsselmaterialien ein, die im lokalen Cache gespeichert werden können.

C#/. NET

```
CacheType defaultCache = new CacheType
{
    Default = new DefaultCache{EntryCapacity = 100}
};
```

Java

```
.cache(CacheType.builder()
        .Default(DefaultCache.builder()
                .entryCapacity(100)
                .build())
```

Default und StormTracking Caches unterstützen dasselbe Threading-Modell, aber Sie müssen nur die Eingabekapazität angeben, um den hierarchischen Schlüsselbund mit dem Standard-Cache zu initialisieren. Für detailliertere Cache-Anpassungen verwenden Sie den Cache. StormTracking

MultiThreaded

Der MultiThreaded Cache kann sicher in Multithread-Umgebungen verwendet werden, bietet jedoch keine Funktionen zur Minimierung AWS KMS von Amazon DynamoDB DynamoDB-Aufrufen. Daher werden alle Threads gleichzeitig benachrichtigt, wenn ein Eintrag für wichtige Materialien in einer Branche abläuft. Dies kann zu mehreren AWS KMS Aufrufen führen, um den Cache zu aktualisieren.

Um Ihren hierarchischen Schlüsselbund mit einem MultiThreaded Cache zu initialisieren, geben Sie die folgenden Werte an:

- **Eintragskapazität:** Schränkt die Anzahl der Einträge für Branch-Schlüsselmaterialien ein, die im lokalen Cache gespeichert werden können.
- **Größe des Endstücks des Eintrags:** Definiert die Anzahl der Einträge, die beschnitten werden müssen, wenn die Eingangskapazität erreicht ist.

C#/. NET

```
CacheType multithreadedCache = new CacheType
{
    MultiThreaded = new MultiThreadedCache
    {
```

```
        EntryCapacity = 100,  
        EntryPruningTailSize = 1  
    }  
};
```

Java

```
.cache(CacheType.builder()  
    .MultiThreaded(MultiThreadedCache.builder()  
    .entryCapacity(100)  
    .entryPruningTailSize(1)  
    .build())
```

StormTracking

Der StormTracking Cache ist so konzipiert, dass er Umgebungen mit hohem Multithreading-Anteil unterstützt. Wenn ein Eintrag für Branch-Schlüsselmaterialien abläuft, verhindert der StormTracking Cache, dass mehrere Threads Amazon DynamoDB aufrufen AWS KMS, indem ein Thread im Voraus darüber informiert wird, dass der Eintrag für Branch-Schlüsselmaterialien abläuft. Dadurch wird sichergestellt, dass nur ein Thread eine Anfrage AWS KMS zur Aktualisierung des Caches sendet.

Um Ihren hierarchischen Schlüsselbund mit einem StormTracking Cache zu initialisieren, geben Sie die folgenden Werte an:

- **Eintragskapazität:** Schränkt die Anzahl der Einträge für Branch-Schlüsselmaterialien ein, die im lokalen Cache gespeichert werden können.
- **Größe des Endstücks des Eintrags:** Definiert die Anzahl der Einträge für das Schlüsselmaterial der Branche, die gleichzeitig beschnitten werden sollen.

Standardwert: 1 Eintrag

- **Übergangszeit:** Definiert die Anzahl der Sekunden vor Ablauf, nach der versucht wird, die wichtigsten Materialien der Branche zu aktualisieren.

Standardwert: 10 Sekunden

- **Verlängerungsintervall:** Definiert die Anzahl der Sekunden zwischen Versuchen, die Schlüsselmaterialien der Filiale zu aktualisieren.

Standardwert: 1 Sekunde

- Fan-out: Definiert die Anzahl der gleichzeitigen Versuche, die wichtigsten Materialien der Filiale zu aktualisieren.

Standardwert: 20 Versuche

- In flight time to live (TTL): Definiert die Anzahl der Sekunden, bis beim Versuch, die Schlüsselmaterialien der Filiale zu aktualisieren, eine Zeitüberschreitung eintritt. Jedes Mal, wenn der Cache als Antwort auf eine zurückkehrt `NoSuchEntryGetCacheEntry`, gilt dieser Verzweigungsschlüssel als aktiv, bis derselbe Schlüssel zusammen mit einem `PutCache` Eintrag geschrieben wird.

Standardwert: 20 Sekunden

- Ruhezustand: Definiert die Anzahl der Sekunden, die ein Thread in den Ruhezustand versetzen soll, wenn der Wert überschritten `fanOut` wird.

Standardwert: 20 Millisekunden

C#/. NET

```
CacheType stormTrackingCache = new CacheType
{
    StormTracking = new StormTrackingCache
    {
        EntryCapacity = 100,
        EntryPruningTailSize = 1,
        FanOut = 20,
        GraceInterval = 1,
        GracePeriod = 10,
        InFlightTTL = 20,
        SleepMilli = 20
    }
};
```

Java

```
.cache(CacheType.builder()
    .MultiThreaded(MultiThreadedCache.builder()
    .entryCapacity(100)
    .entryPruningTailSize(1)
    .gracePeriod(10)
    .graceInterval(1)
    .fanOut(20)
```

```

    .inFlightTTL(20)
    .sleepMilli(20)
    .build()

```

- (Optional) Eine Liste von Grant-Tokens

Wenn Sie den Zugriff auf den KMS Schlüssel in Ihrem hierarchischen Schlüsselbund mit [Grants](#) steuern, müssen Sie bei der Initialisierung des Schlüsselbunds alle erforderlichen Grant-Token angeben.

Im folgenden Beispiel wird ein hierarchischer Schlüsselbund mit dem in Schritt 2 erstellten Branch-Schlüssel-ID-Lieferanten, einem Cache-Limit TLL von 600 Sekunden und einer Eingabekapazität von 1000 initialisiert.

C# / .NET

```

var createKeyringInput = new CreateAwsKmsHierarchicalKeyringInput
{
    KeyStore = keystore,
    BranchKeyIdSupplier = branchKeySupplier,
    Cache = new CacheType { Default = new DefaultCache{EntryCapacity = 1000} },
    TtlSeconds = 600
};
var keyring = mpl.CreateAwsKmsHierarchicalKeyring(createKeyringInput);

```

Java

```

final MaterialProviders matProv = MaterialProviders.builder()
    .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
    .build();
final CreateAwsKmsHierarchicalKeyringInput keyringInput =
    CreateAwsKmsHierarchicalKeyringInput.builder()
        .keyStore(branchKeyStoreName)
        .branchKeyIdSupplier(branchKeyIdSupplier)
        .ttlSeconds(600)
        .cache(CacheType.builder() //OPTIONAL
            .Default(DefaultCache.builder()
                .entryCapacity(100)
                .build())
            .build());
final IKeyring hierarchicalKeyring =
    matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);

```

Schritt 4: Erstellen Sie benutzerfreundliche Namen für jeden Zweigschlüssel

Im folgenden Beispiel werden benutzerfreundliche Namen für die beiden in Schritt 1 erstellten Verzweigungsschlüssel erstellt. Der AWS Encryption SDK verwendet Verschlüsselungskontexte, um den von Ihnen definierten Anzeigenamen dem zugehörigen Namen zuzuordnen `branch-key-id`.

C#/.NET

```
// Create encryption contexts for the two branch keys created in Step 1
var encryptionContextA = new Dictionary<string, string>()
{
    // We will encrypt with branchKeyTenantA
    {"tenant", "TenantA"},
    {"encryption", "context"},
    {"is not", "secret"},
    {"but adds", "useful metadata"},
    {"that can help you", "be confident that"},
    {"the data you are handling", "is what you think it is"}
};
var encryptionContextB = new Dictionary<string, string>()
{
    // We will encrypt with branchKeyTenantB
    {"tenant", "TenantB"},
    {"encryption", "context"},
    {"is not", "secret"},
    {"but adds", "useful metadata"},
    {"that can help you", "be confident that"},
    {"the data you are handling", "is what you think it is"}
};

// Instantiate the AWS Encryption SDK var esdk = new ESDK(new
    AwsEncryptionSdkConfig());

var encryptInputA = new EncryptInput
{
    Plaintext = plaintext,
    Keyring = keyring,
    // Encrypt with branchKeyId1
    EncryptionContext = encryptionContextA
};

var encryptInputB = new EncryptInput
```

```
{
    Plaintext = plaintext,
    Keyring = keyring,
    // Encrypt with branchKeyId2
    EncryptionContext = encryptionContextB
};

var encryptOutput = esdk.Encrypt(encryptInputA);
encryptOutput = esdk.Encrypt(encryptInputB);

// Use the encryption contexts to define friendly names for each branch key
public class ExampleBranchKeySupplier : IBranchKeyIdSupplier
{
    private string branchKeyTenantA;
    private string branchKeyTenantB;

    public ExampleBranchKeySupplier(string branchKeyTenantA, string
branchKeyTenantB)
    {
        this.branchKeyTenantA = branchKeyTenantA;
        this.branchKeyTenantB = branchKeyTenantB;
    }

    public GetBranchKeyIdOutput GetBranchKeyId(GetBranchKeyIdInput input)
    {
        Dictionary<string, string> encryptionContext = input.EncryptionContext;

        if (!encryptionContext.ContainsKey("tenant"))
        {
            throw new Exception("EncryptionContext invalid, does not contain
expected tenant key value pair.");
        }

        string tenant = encryptionContext["tenant"];
        string branchKeyId;

        if (tenant.Equals("TenantA"))
        {
            GetBranchKeyIdOutput output = new GetBranchKeyIdOutput();
            output.BranchKeyId = branchKeyTenantA;
            return output;
        } else if (tenant.Equals("TenantB"))
        {
            GetBranchKeyIdOutput output = new GetBranchKeyIdOutput();
```

```
        output.BranchKeyId = branchKeyTenantB;
        return output;
    }
    else
    {
        throw new Exception("Item does not have a valid tenantID.");
    }
}
}
```

Java

```
// Create encryption context for branchKeyTenantA
Map<String, String> encryptionContextA = new HashMap<>();
encryptionContextA.put("tenant", "TenantA");
encryptionContextA.put("encryption", "context");
encryptionContextA.put("is not", "secret");
encryptionContextA.put("but adds", "useful metadata");
encryptionContextA.put("that can help you", "be confident that");
encryptionContextA.put("the data you are handling", "is what you think it is");

// Create encryption context for branchKeyTenantB
Map<String, String> encryptionContextB = new HashMap<>();
encryptionContextB.put("tenant", "TenantB");
encryptionContextB.put("encryption", "context");
encryptionContextB.put("is not", "secret");
encryptionContextB.put("but adds", "useful metadata");
encryptionContextB.put("that can help you", "be confident that");
encryptionContextB.put("the data you are handling", "is what you think it is");

// Instantiate the AWS Encryption SDK
final AwsCrypto crypto = AwsCrypto.builder().build();

final CryptoResult<byte[], ?> encryptResultA = crypto.encryptData(keyring,
    plaintext, encryptionContextA);

final CryptoResult<byte[], ?> encryptResultB = crypto.encryptData(keyring,
    plaintext, encryptionContextB);

// Use the encryption contexts to define friendly names for each branch key
public class ExampleBranchKeyIdSupplier implements IBranchKeyIdSupplier {
    private static String branchKeyIdForTenantA;
```

```
private static String branchKeyIdForTenantB;

public ExampleBranchKeyIdSupplier(String tenant1Id, String tenant2Id) {
    this.branchKeyIdForTenantA = tenant1Id;
    this.branchKeyIdForTenantB = tenant2Id;
}

@Override
public GetBranchKeyIdOutput GetBranchKeyId(GetBranchKeyIdInput input) {

    Map<String, String> encryptionContext = input.encryptionContext();

    if (!encryptionContext.containsKey("tenant"))
    {
        throw new IllegalArgumentException("EncryptionContext invalid, does
not contain expected tenant key value pair.");
    }

    String tenantKeyId = encryptionContext.get("tenant");
    String branchKeyId;

    if (tenantKeyId.equals("TenantA")) {
        branchKeyId = branchKeyIdForTenantA;
    } else if (tenantKeyId.equals("TenantB")) {
        branchKeyId = branchKeyIdForTenantB;
    } else {
        throw new IllegalArgumentException("Item does not contain valid
tenant ID");
    }

    return GetBranchKeyIdOutput.builder().branchKeyId(branchKeyId).build();
}
}
```

AWS KMS ECDH-Schlüsselbünde

Important

Der AWS KMS ECDH Schlüsselbund ist nur mit Version 4 verfügbar. x der Form AWS Encryption SDK . NETund Version 3. x der AWS-Verschlüsselungs-SDK for Java. Der AWS KMS ECDH Schlüsselbund wurde in Version 1.5.0 der Material Providers Library eingeführt.

Ein AWS KMS ECDH Schlüsselbund verwendet eine asymmetrische Schlüsselvereinbarung, [AWS KMS keys](#)um einen gemeinsamen symmetrischen Wrapping-Schlüssel zwischen zwei Parteien abzuleiten. Zunächst verwendet der Schlüsselbund den Elliptic Curve Diffie-Hellman (ECDH) - Schlüsselvereinbarungsalgorithmus, um ein gemeinsames Geheimnis aus dem privaten Schlüssel im key pair des Absenders und dem öffentlichen KMS Schlüssel des Empfängers abzuleiten. Anschließend leitet der Schlüsselbund anhand des gemeinsamen geheimen Schlüssels den gemeinsamen Wrapping-Schlüssel ab, der Ihre Datenverschlüsselungsschlüssel schützt. [Die Schlüsselableitungsfunktion, die \(KDF_CTR_HMAC_SHA384\) AWS Encryption SDK verwendet, um den gemeinsamen Wrapping-Schlüssel abzuleiten, entspricht den Empfehlungen für die Schlüsselableitung. NIST](#)

Die Funktion zur Schlüsselableitung gibt 64 Byte an Schlüsselmaterial zurück. Um sicherzustellen, dass beide Parteien das richtige Schlüsselmaterial verwenden, AWS Encryption SDK verwendet die ersten 32 Byte als Commitment-Schlüssel und die letzten 32 Byte als gemeinsamen Wrapping-Schlüssel. Wenn der Schlüsselbund beim Entschlüsseln nicht denselben Commitment-Schlüssel und denselben gemeinsamen Wrapping-Schlüssel reproduzieren kann, die im Chiffretext der Nachrichtenkopfzeile gespeichert sind, schlägt der Vorgang fehl. Wenn Sie beispielsweise Daten mit einem Schlüsselbund verschlüsseln, der mit Alices privatem Schlüssel und Bobs öffentlichem Schlüssel konfiguriert ist, reproduziert ein Schlüsselbund, der mit Bobs privatem Schlüssel und Alices öffentlichem Schlüssel konfiguriert ist, denselben Commitment-Schlüssel und gemeinsamen Wrapping-Schlüssel und kann die Daten entschlüsseln. Wenn Bobs öffentlicher Schlüssel nicht von einem KMS key pair stammt, kann Bob einen [ECDHRaw-Schlüsselbund](#) erstellen, um die Daten zu entschlüsseln.

Der AWS KMS ECDH Schlüsselbund verschlüsselt Daten mit einem symmetrischen Schlüssel unter Verwendung von -. AES GCM Der Datenschlüssel wird dann mit dem abgeleiteten gemeinsamen Wrapping-Schlüssel unter Verwendung von - umhüllt. AES GCM [Jeder AWS KMS ECDH Schlüsselbund kann nur einen gemeinsamen Wickelschlüssel haben, aber Sie können mehrere Schlüsselbunde, allein oder zusammen mit AWS KMS ECDH anderen Schlüsselbunden, zu einem Mehrfachschlüsselbund zusammenfassen.](#)

Themen

- [AWS KMS ECDHErforderliche Berechtigungen für Schlüsselanhänger](#)
- [Einen Schlüsselbund erstellen AWS KMS ECDH](#)
- [Einen AWS KMS ECDH Discovery-Schlüsselbund erstellen](#)

AWS KMS ECDHErforderliche Berechtigungen für Schlüsselanhänger

Für das AWS Encryption SDK ist kein AWS Konto erforderlich und es ist auch nicht von einem AWS Dienst abhängig. Um einen AWS KMS ECDH Schlüsselbund verwenden zu können, benötigen Sie jedoch ein AWS Konto und die folgenden Mindestberechtigungen für AWS KMS keys den Schlüsselbund. Die Berechtigungen variieren je nachdem, welches Schlüsselvereinbarungsschema Sie verwenden.

- Um Daten mithilfe des `KmsPrivateKeyToStaticPublicKey` Schlüsselvereinbarungsschemas zu verschlüsseln und zu entschlüsseln, benötigen Sie [kms: GetPublicKey](#) und [kms: DeriveSharedSecret](#) für das asymmetrische KMS key pair des Absenders. Wenn Sie den DER -codierten öffentlichen Schlüssel des Absenders direkt angeben, wenn Sie Ihren Schlüsselbund instanziiieren, benötigen Sie nur die [kms: DeriveSharedSecret](#) -Berechtigung für das asymmetrische key pair des Absenders. KMS
- Um Daten mithilfe des `KmsPublicKeyDiscovery` Schlüsselvereinbarungsschemas zu entschlüsseln, benötigen Sie die `GetPublicKey` Berechtigungen [kms: DeriveSharedSecret](#) und [kms:](#) für das angegebene asymmetrische KMS key pair.

Einen Schlüsselbund erstellen AWS KMS ECDH

Um einen AWS KMS ECDH Schlüsselbund zu erstellen, der Daten ver- und entschlüsselt, müssen Sie das `KmsPrivateKeyToStaticPublicKey` Schlüsselvereinbarungsschema verwenden. Um einen AWS KMS ECDH Schlüsselbund mit dem Schlüsselvereinbarungsschema zu initialisieren, `KmsPrivateKeyToStaticPublicKey` geben Sie die folgenden Werte an:

- ID des Absenders AWS KMS key

Muss ein asymmetrisches NIST — empfohlenes elliptisches KMS Curve-Schlüsselpaar (ECC) mit einem `KeyUsage` Wert von `identify`. `KEY_AGREEMENT` Der private Schlüssel des Absenders wird verwendet, um den gemeinsamen geheimen Schlüssel abzuleiten.

- (Optional) Der öffentliche Schlüssel des Absenders

[Muss ein DER -codierter öffentlicher X.509-Schlüssel sein, auch bekannt als `SubjectPublicKeyInfo` \(SPKI\), wie in 5280 definiert. RFC](#)

Die AWS KMS `GetPublicKey` Operation gibt den öffentlichen Schlüssel eines asymmetrischen KMS key pair im erforderlichen DER -codierten Format zurück.

Um die Anzahl der AWS KMS Anrufe zu reduzieren, die Ihr Schlüsselbund tätigt, können Sie den öffentlichen Schlüssel des Absenders direkt angeben. Wenn kein Wert für den öffentlichen Schlüssel des Absenders angegeben wird, ruft der Schlüsselbund auf, AWS KMS um den öffentlichen Schlüssel des Absenders abzurufen.

- Der öffentliche Schlüssel des Empfängers

[Sie müssen den DER -codierten öffentlichen X.509-Schlüssel des Empfängers, auch bekannt als SubjectPublicKeyInfo \(SPKI\), wie in 5280 definiert, angeben. RFC](#)

Die AWS KMS [GetPublicKey](#)Operation gibt den öffentlichen Schlüssel eines asymmetrischen KMS key pair im erforderlichen DER -codierten Format zurück.

- Kurvenspezifikation

Identifiziert die Spezifikation für elliptische Kurven in den angegebenen Schlüsselpaaren. Sowohl die Schlüsselpaare des Absenders als auch des Empfängers müssen dieselbe Kurvenspezifikation haben.

Zulässige Werte: ECC_NIST_P256, ECC_NIS_P384, ECC_NIST_P512

- (Optional) Eine Liste von Grant-Tokens

Wenn Sie den Zugriff auf den KMS Schlüssel in Ihrem AWS KMS ECDH Schlüsselbund mit [Grants](#) kontrollieren, müssen Sie bei der Initialisierung des Schlüsselbunds alle erforderlichen Grant-Token angeben.

C# / .NET

Im folgenden Beispiel wird ein AWS KMS ECDH Schlüsselbund mit dem Schlüssel des Absenders, dem öffentlichen KMS Schlüssel des Absenders und dem öffentlichen Schlüssel des Empfängers erstellt. In diesem Beispiel wird der optionale `SenderPublicKey` Parameter verwendet, um den öffentlichen Schlüssel des Absenders bereitzustellen. Wenn Sie den öffentlichen Schlüssel des Absenders nicht angeben, ruft der Schlüsselbund auf, AWS KMS um den öffentlichen Schlüssel des Absenders abzurufen. Sowohl die Schlüsselpaare des Absenders als auch des Empfängers befinden sich auf der ECC_NIST_P256 Kurve.

```
// Instantiate material providers
var materialProviders = new MaterialProviders(new MaterialProvidersConfig());

// Must be DER-encoded X.509 public keys
```

```

var BobPublicKey = new MemoryStream(new byte[] { });
var AlicePublicKey = new MemoryStream(new byte[] { });

// Create the AWS KMS ECDH static keyring
var staticConfiguration = new KmsEcdhStaticConfigurations
{
    KmsPrivateKeyToStaticPublicKey = new KmsPrivateKeyToStaticPublicKeyInput
    {
        SenderKmsIdentifier = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        SenderPublicKey = BobPublicKey,
        RecipientPublicKey = AlicePublicKey
    }
};

var createKeyringInput = new CreateAwsKmsEcdhKeyringInput
{
    CurveSpec = ECDHCurveSpec.ECC_NIST_P256,
    KmsClient = new AmazonKeyManagementServiceClient(),
    KeyAgreementScheme = staticConfiguration
};

var keyring = materialProviders.CreateAwsKmsEcdhKeyring(createKeyringInput);

```

Java

Im folgenden Beispiel wird ein AWS KMS ECDH Schlüsselbund mit dem KMS Schlüssel des Absenders, dem öffentlichen Schlüssel des Absenders und dem öffentlichen Schlüssel des Empfängers erstellt. In diesem Beispiel wird der optionale `senderPublicKey` Parameter verwendet, um den öffentlichen Schlüssel des Absenders bereitzustellen. Wenn Sie den öffentlichen Schlüssel des Absenders nicht angeben, ruft der Schlüsselbund auf, AWS KMS um den öffentlichen Schlüssel des Absenders abzurufen. Sowohl die Schlüsselpaare des Absenders als auch des Empfängers befinden sich auf der `ECC_NIST_P256` Kurve.

```

// Retrieve public keys
// Must be DER-encoded X.509 public keys
ByteBuffer BobPublicKey = getPublicKeyBytes("arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab");
    ByteBuffer AlicePublicKey = getPublicKeyBytes("arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321");

// Create the AWS KMS ECDH static keyring

```

```

final CreateAwsKmsEcdhKeyringInput senderKeyringInput =
    CreateAwsKmsEcdhKeyringInput.builder()
        .kmsClient(KmsClient.create())
        .curveSpec(ECDHCurveSpec.ECC_NIST_P256)
        .keyAgreementScheme(
            KmsEcdhStaticConfigurations.builder()
                .kmsPrivateKeyToStaticPublicKey(
                    KmsPrivateKeyToStaticPublicKeyInput.builder()
                        .senderKmsIdentifier("arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab")
                        .senderPublicKey(BobPublicKey)
                        .recipientPublicKey(AlicePublicKey)
                        .build()).build()).build();

```

Einen AWS KMS ECDH Discovery-Schlüsselbund erstellen

Beim Entschlüsseln empfiehlt es sich, die Schlüssel anzugeben, die sie verwenden AWS Encryption SDK können. Um dieser bewährten Methode zu folgen, verwenden Sie einen AWS KMS ECDH Schlüsselbund mit dem `KmsPrivateKeyToStaticPublicKey` Schlüsselvereinbarungsschema. Sie können jedoch auch einen AWS KMS ECDH Discovery-Schlüsselbund erstellen, d. h. einen AWS KMS ECDH Schlüsselbund, der jede Nachricht entschlüsseln kann, bei der der öffentliche Schlüssel des angegebenen Schlüsselpaars mit dem öffentlichen KMS Schlüssel des Empfängers übereinstimmt, der im Nachrichtenchiffretext gespeichert ist.

Important

Wenn Sie Nachrichten mithilfe des `KmsPublicKeyDiscovery` Schlüsselvereinbarungsschemas entschlüsseln, akzeptieren Sie alle öffentlichen Schlüssel, unabhängig davon, wem sie gehören.

Um einen AWS KMS ECDH Schlüsselbund mit dem Schlüsselvereinbarungsschema zu initialisieren, geben Sie die folgenden Werte an: `KmsPublicKeyDiscovery`

- ID des Empfängers AWS KMS key

Muss ein asymmetrisches NIST — empfohlenes elliptisches KMS Curve-Schlüsselpaar (ECC) mit einem `KeyUsage` Wert von `identifizieren`. `KEY_AGREEMENT`

- Kurvenspezifikation

Identifiziert die elliptische Kurvenspezifikation im KMS key pair des Empfängers.

Zulässige Werte: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

- (Optional) Eine Liste von Grant-Tokens

Wenn Sie den Zugriff auf den KMS Schlüssel in Ihrem AWS KMS ECDH Schlüsselbund mit [Grants](#) kontrollieren, müssen Sie bei der Initialisierung des Schlüsselbunds alle erforderlichen Grant-Token angeben.

C# / .NET

Im folgenden Beispiel wird ein AWS KMS ECDH Discovery-Schlüsselbund mit einem KMS key pair auf der `ECC_NIST_P256` Kurve erstellt. Sie müssen über die `DeriveSharedSecret` Berechtigungen [kms: GetPublicKey](#) und [kms:](#) für das angegebene KMS key pair verfügen. Dieser Schlüsselbund kann jede Nachricht entschlüsseln, bei der der öffentliche Schlüssel des angegebenen key pair mit dem öffentlichen KMS Schlüssel des Empfängers übereinstimmt, der im Nachrichtenchiffretext gespeichert ist.

```
// Instantiate material providers
var materialProviders = new MaterialProviders(new MaterialProvidersConfig());

// Create the AWS KMS ECDH discovery keyring
var discoveryConfiguration = new KmsEcdhStaticConfigurations
{
    KmsPublicKeyDiscovery = new KmsPublicKeyDiscoveryInput
    {
        RecipientKmsIdentifier = "arn:aws:kms:us-
west-2:111122223333:key/0987dcb-a-09fe-87dc-65ba-ab0987654321"
    }
};
var createKeyringInput = new CreateAwsKmsEcdhKeyringInput
{
    CurveSpec = ECDHCurveSpec.ECC_NIST_P256,
    KmsClient = new AmazonKeyManagementServiceClient(),
    KeyAgreementScheme = discoveryConfiguration
};
var keyring = materialProviders.CreateAwsKmsEcdhKeyring(createKeyringInput);
```

Java

Im folgenden Beispiel wird ein AWS KMS ECDH Discovery-Schlüsselbund mit einem KMS key pair auf der ECC_NIST_P256 Kurve erstellt. Sie müssen über die `DeriveSharedSecret` Berechtigungen [kms: GetPublicKey](#) und [kms:](#) für das angegebene KMS key pair verfügen. Dieser Schlüsselbund kann jede Nachricht entschlüsseln, bei der der öffentliche Schlüssel des angegebenen key pair mit dem öffentlichen KMS Schlüssel des Empfängers übereinstimmt, der im Nachrichtenchiffretext gespeichert ist.

```
// Create the AWS KMS ECDH discovery keyring
final CreateAwsKmsEcdhKeyringInput recipientKeyringInput =
    CreateAwsKmsEcdhKeyringInput.builder()
        .kmsClient(KmsClient.create())
        .curveSpec(ECDHCurveSpec.ECC_NIST_P256)
        .keyAgreementScheme(
            KmsEcdhStaticConfigurations.builder()
                .kmsPublicKeyDiscovery(
                    KmsPublicKeyDiscoveryInput.builder()
                        .recipientKmsIdentifier("arn:aws:kms:us-
west-2:111122223333:key/0987dcb-a-09fe-87dc-65ba-ab0987654321").build()
                ).build()
            ).build();
```

Unformatierte Schlüsselanhänger AES

AWS Encryption SDK Damit können Sie einen AES symmetrischen Schlüssel verwenden, den Sie als Umschließungsschlüssel angeben, der Ihren Datenschlüssel schützt. Sie müssen das Schlüsselmaterial generieren, speichern und schützen, vorzugsweise in einem Hardware-Sicherheitsmodul (HSM) oder einem Schlüsselverwaltungssystem. Verwenden Sie einen AES Raw-Schlüsselbund, wenn Sie den Wrap-Schlüssel bereitstellen und die Datenschlüssel lokal oder offline verschlüsseln müssen.

Der AES Raw-Schlüsselbund verschlüsselt Daten mithilfe des AES GCM -Algorithmus und eines Umschließungsschlüssels, den Sie als Byte-Array angeben. [Sie können in jedem AES Raw-Schlüsselbund nur einen Wrap-Schlüssel angeben, Sie können jedoch mehrere AES Raw-Schlüsselbunde einzeln oder zusammen mit anderen Schlüsselbunden in einen Mehrfachschlüsselbund aufnehmen.](#)

Der AES Raw-Schlüsselbund entspricht der Klasse in der und der [JceMasterKey](#)-Klasse in der AWS-Verschlüsselungs-SDK for Java und interagiert mit ihnen, AWS-Verschlüsselungs-SDK for Python wenn sie mit einem Verschlüsselungsschlüssel verwendet werden. [RawMasterKey](#)-AES Sie können Ihre Daten mit einer Programmiersprachen-Implementierung verschlüsseln und mit jeder beliebigen anderen Sprachenimplementierung unter Verwendung desselben Umhüllungsschlüssels entschlüsseln. Details hierzu finden Sie unter [Schlüsselbund-Kompatibilität](#).

Wichtige Namespaces und Namen

Um den AES Schlüssel in einem Schlüsselbund zu identifizieren, verwendet der AES Raw-Schlüsselbund einen Schlüsselnamespace und einen Schlüsselnamen, die Sie angeben. Diese Werte sind nicht geheim. Sie erscheinen im Klartext in der Kopfzeile der [verschlüsselten Nachricht](#), die der Verschlüsselungsvorgang zurückgibt. Wir empfehlen, einen Schlüsselnamespace für Ihr HSM oder Ihr Schlüsselverwaltungssystem und einen Schlüsselnamen zu verwenden, der den AES Schlüssel in diesem System identifiziert.

Note

Der Schlüsselnamespace und der Schlüsselname entsprechen den Feldern Provider-ID (oder Provider) und Key-ID in den `JceMasterKey` Feldern und `RawMasterKey` Das AWS-Verschlüsselungs-SDK for C und AWS Encryption SDK für .NET reserviert den `aws-kms` Schlüssel-namespace-Wert für KMS Schlüssel. Verwenden Sie diesen namespace-Wert nicht in einem AES Raw-Schlüsselbund oder Raw-Schlüsselbund mit diesen RSA Bibliotheken.

Wenn Sie verschiedene Schlüsselbunde zum Verschlüsseln und Entschlüsseln einer bestimmten Nachricht erstellen, sind die namespace- und Namenswerte entscheidend. Wenn der Schlüsselnamespace und der Schlüsselname im Schlüsselbund für die Entschlüsselung nicht exakt und unter Berücksichtigung der Groß- und Kleinschreibung mit dem Schlüsselnamespace und dem Schlüsselnamen im Verschlüsselungsschlüsselbund übereinstimmen, wird der Schlüsselbund nicht verwendet, auch wenn die Schlüsselmaterial-Bytes identisch sind.

Sie könnten beispielsweise einen AES Raw-Schlüsselbund mit Schlüsselnamespace und Schlüsselnamen definieren. `HSM_01_AES_256_012` Anschließend verwenden Sie diesen Schlüsselbund, um einige Daten zu verschlüsseln. Um diese Daten zu entschlüsseln, erstellen Sie einen AES Raw-Schlüsselbund mit demselben Schlüsselnamespace, demselben Schlüsselnamen und demselben Schlüsselmaterial.

Die folgenden Beispiele zeigen, wie Sie einen Raw-Schlüsselbund erstellen. AES Die `AESWrappingKey` Variable steht für das von Ihnen bereitgestellte Schlüsselmaterial.

C

Um einen AES Raw-Schlüsselbund in der zu instanzieren, verwenden Sie AWS-Verschlüsselungs-SDK for C. `aws_cryptosdk_raw_aes_keyring_new()` [Ein vollständiges Beispiel finden Sie unter `raw_aes_keyring.c`.](#)

```
struct aws_allocator *alloc = aws_default_allocator();

AWS_STATIC_STRING_FROM_LITERAL(wrapping_key_namespace, "HSM_01");
AWS_STATIC_STRING_FROM_LITERAL(wrapping_key_name, "AES_256_012");

struct aws_cryptosdk_keyring *raw_aes_keyring = aws_cryptosdk_raw_aes_keyring_new(
    alloc, wrapping_key_namespace, wrapping_key_name, aes_wrapping_key,
    wrapping_key_len);
```

C# / .NET

Um einen Raw-Schlüsselbund in für zu erstellen. AES AWS Encryption SDK NET, verwende die `materialProviders.CreateRawAesKeyring()` Methode. Ein vollständiges Beispiel finden Sie unter [R `awAESKeyring Example.cs`.](#)

Das folgende Beispiel verwendet Version 4. x der AWS Encryption SDK Form. NET.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk = new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

var keyNamespace = "HSM_01";
var keyName = "AES_256_012";

// This example uses the key generator in Bouncy Castle to generate the key
// material.
// In production, use key material from a secure source.
var aesWrappingKey = new
    MemoryStream(GeneratorUtilities.GetKeyGenerator("AES256").GenerateKey());

// Create the keyring that determines how your data keys are protected.
var createKeyringInput = new CreateRawAesKeyringInput
{
```

```
    KeyNamespace = keyNamespace,  
    KeyName = keyName,  
    WrappingKey = aesWrappingKey,  
    WrappingAlg = AesWrappingAlg.ALG_AES256_GCM_IV12_TAG16  
};  
  
var keyring = materialProviders.CreateRawAesKeyring(createKeyringInput);
```

JavaScript Browser

Das AWS-Verschlüsselungs-SDK for JavaScript im Browser bezieht seine kryptografischen Primitive aus dem [WebCrypto](#) API. Bevor Sie den Schlüsselbund erstellen, müssen Sie ihn verwenden, `RawAesKeyringWebCrypto.importCryptoKey()` um das Rohschlüsselmaterial in das Backend zu importieren. WebCrypto Dadurch wird sichergestellt, dass der Schlüsselbund vollständig ist, obwohl alle Aufrufe asynchron sind. WebCrypto

Verwenden Sie dann die Methode, um einen AES Raw-Schlüsselbund zu instanziiieren. `RawAesKeyringWebCrypto()` Sie müssen den AES Wrapping-Algorithmus („Wrapping Suite“) auf der Grundlage der Länge Ihres Schlüsselmaterials angeben. Ein vollständiges Beispiel finden Sie unter [aes_simple.ts](#) (Browser). JavaScript

```
const keyNamespace = 'HSM_01'  
const keyName = 'AES_256_012'  
  
const wrappingSuite =  
  RawAesWrappingSuiteIdentifier.AES256_GCM_IV12_TAG16_NO_PADDING  
  
/* Import the plaintext AES key into the WebCrypto backend. */  
const aesWrappingKey = await RawAesKeyringWebCrypto.importCryptoKey(  
  rawAesKey,  
  wrappingSuite  
)  
  
const rawAesKeyring = new RawAesKeyringWebCrypto({  
  keyName,  
  keyNamespace,  
  wrappingSuite,  
  aesWrappingKey  
})
```


JavaScript Node.js

Um einen AES Raw-Schlüsselbund in der AWS-Verschlüsselungs-SDK for JavaScript for Node.js zu instanziiieren, erstellen Sie eine Instanz der Klasse `RawAesKeyringNode`. Sie müssen den AES Wrapping-Algorithmus („Wrapping Suite“) auf der Grundlage der Länge Ihres Schlüsselmaterials angeben. Ein vollständiges Beispiel finden Sie unter [aes_simple.ts](#) (Node.js).

JavaScript

```
const keyName = 'AES_256_012'  
const keyNamespace = 'HSM_01'  
  
const wrappingSuite =  
  RawAesWrappingSuiteIdentifier.AES256_GCM_IV12_TAG16_NO_PADDING  
  
const rawAesKeyring = new RawAesKeyringNode({  
  keyName,  
  keyNamespace,  
  aesWrappingKey,  
  wrappingSuite,  
})
```

Java

Um einen Raw-Schlüsselbund in der zu instanziiieren, verwenden Sie `AES AWS-Verschlüsselungs-SDK for JavamatProv.CreateRawAesKeyring()`

```
final CreateRawAesKeyringInput keyringInput = CreateRawAesKeyringInput.builder()  
  .keyName("AES_256_012")  
  .keyNamespace("HSM_01")  
  .wrappingKey(AESWrappingKey)  
  .wrappingAlg(AesWrappingAlg.ALG_AES256_GCM_IV12_TAG16)  
  .build();  
final MaterialProviders matProv = MaterialProviders.builder()  
  .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())  
  .build();  
IKeyring rawAesKeyring = matProv.CreateRawAesKeyring(keyringInput);
```

RSARaw-Schlüsselringe

Der RSA Raw-Schlüsselbund führt eine asymmetrische Verschlüsselung und Entschlüsselung von Datenschlüsseln im lokalen Speicher mit RSA öffentlichen und privaten Schlüsseln durch,

die Sie bereitstellen. Sie müssen den privaten Schlüssel generieren, speichern und schützen, vorzugsweise in einem Hardware-Sicherheitsmodul (HSM) oder einem Schlüsselverwaltungssystem. Die Verschlüsselungsfunktion verschlüsselt den Datenschlüssel unter dem RSA öffentlichen Schlüssel. Die Entschlüsselungsfunktion entschlüsselt den Datenschlüssel mithilfe des privaten Schlüssels. Sie können zwischen den verschiedenen [RSAPadding-Modi wählen](#).

Ein RSA Raw-Schlüsselbund, der verschlüsselt und entschlüsselt, muss ein asymmetrisches Paar aus öffentlichem Schlüssel und privatem Schlüssel enthalten. Sie können Daten jedoch mit einem RSA Raw-Schlüsselbund verschlüsseln, der nur über einen öffentlichen Schlüssel verfügt, und Sie können Daten mit einem RSA Raw-Schlüsselbund entschlüsseln, der nur über einen privaten Schlüssel verfügt. [Sie können einen beliebigen Raw-Schlüsselbund in einen RSA Mehrfachschlüsselbund aufnehmen](#). Wenn Sie einen RSA Raw-Schlüsselbund mit einem öffentlichen und einem privaten Schlüssel konfigurieren, stellen Sie sicher, dass sie Teil desselben key pair sind. Einige Sprachimplementierungen von AWS Encryption SDK werden keinen RSA Raw-Schlüsselbund mit Schlüsseln aus verschiedenen Paaren erstellen. Andere verlassen sich darauf, dass Sie überprüfen, ob Ihre Schlüssel von demselben key pair stammen.

Der RSA Raw-Schlüsselbund entspricht in der und der [JceMasterKeyRawMasterKey](#) in der AWS-Verschlüsselungs-SDK for Java und interagiert mit ihnen, AWS-Verschlüsselungs-SDK for Python wenn sie mit RSA asymmetrischen Verschlüsselungsschlüsseln verwendet werden. Sie können Ihrer Daten mit einer Programmiersprachen-Implementierung verschlüsseln und mit jeder beliebigen anderen Sprachenimplementierung unter Verwendung desselben Umhüllungsschlüssels entschlüsseln. Details hierzu finden Sie unter [Schlüsselbund-Kompatibilität](#).

Note

Der RSA Raw-Schlüsselbund unterstützt keine asymmetrischen Schlüssel. KMS Wenn Sie asymmetrische RSA KMS Schlüssel verwenden möchten, Version 4. x der Form AWS Encryption SDK . NET und Version 3. x der AWS-Verschlüsselungs-SDK for Java unterstützten AWS KMS Schlüsselbunde, die symmetrische Verschlüsselung (SYMMETRIC_DEFAULT) oder asymmetrische Verschlüsselung verwenden. RSA AWS KMS keys

Wenn Sie Daten mit einem RSA Raw-Schlüsselbund verschlüsseln, der den öffentlichen Schlüssel eines Schlüssels enthält, kann weder der noch den RSA KMS AWS Encryption SDK Schlüssel entschlüsseln. AWS KMS Sie können den privaten Schlüssel eines AWS KMS asymmetrischen Schlüssels nicht in einen KMS Raw-Schlüsselbund exportieren. RSA Der

AWS KMS Entschlüsselungsvorgang kann die [verschlüsselte Nachricht](#), die zurückgegeben wird, nicht entschlüsseln. AWS Encryption SDK

Achten Sie beim Aufbau eines RSA Raw-Schlüsselbunds in der darauf AWS-Verschlüsselungs-SDK for C, den Inhalt der PEM Datei, die jeden Schlüssel enthält, als Null-terminierte C-Zeichenfolge anzugeben, nicht als Pfad oder Dateiname. [Achten Sie bei der Erstellung eines RSA Raw-Schlüsselbundes auf mögliche Inkompatibilitäten mit JavaScript anderen Sprachimplementierungen.](#)

Namespaces und Namen

Um das RSA Schlüsselmaterial in einem Schlüsselbund zu identifizieren, verwendet der RSA Raw-Schlüsselbund einen Schlüsselnamespace und einen Schlüsselnamen, die Sie angeben. Diese Werte sind nicht geheim. Sie erscheinen im Klartext in der Kopfzeile der [verschlüsselten Nachricht](#), die der Verschlüsselungsvorgang zurückgibt. Wir empfehlen, den Schlüsselnamespace und den Schlüsselnamen zu verwenden, die das RSA key pair (oder seinen privaten Schlüssel) in Ihrem HSM oder dem Schlüsselverwaltungssystem identifizieren.

Note

Der Schlüsselnamespace und der Schlüsselname entsprechen den Feldern Provider-ID (oder Provider) und Key-ID in den `JceMasterKey` Feldern und `RawMasterKey`. Der AWS-Verschlüsselungs-SDK for C reserviert den `aws-kms` Schlüsselnamespace-Wert für KMS Schlüssel. Verwenden Sie ihn nicht in einem AES Raw-Schlüsselbund oder einem RSA Raw-Schlüsselbund mit dem. AWS-Verschlüsselungs-SDK for C

Wenn Sie verschiedene Schlüsselbunde zum Verschlüsseln und Entschlüsseln einer bestimmten Nachricht erstellen, sind der Namespace und die Namenswerte entscheidend. Wenn der Schlüsselnamespace und der Schlüsselname im Entschlüsselungsschlüsselbund nicht exakt und unter Berücksichtigung der Groß- und Kleinschreibung für den Schlüsselnamespace und den Schlüsselnamen im Verschlüsselungsschlüsselbund übereinstimmen, wird der Entschlüsselungsschlüsselbund nicht verwendet, auch wenn die Schlüssel aus demselben key pair stammen.

Der Schlüsselnamespace und der Schlüsselname des Schlüsselmaterials in den Verschlüsselungs- und Entschlüsselungsschlüsselbunden müssen identisch sein, unabhängig davon, ob der Schlüsselbund den RSA öffentlichen Schlüssel, den RSA privaten Schlüssel oder beide Schlüssel

im key pair enthält. Nehmen wir beispielsweise an, Sie verschlüsseln Daten mit einem RSA Raw-Schlüsselbund für einen RSA öffentlichen Schlüssel mit Schlüsselnamespace und Schlüsselname. HSM_01 RSA_2048_06 Um diese Daten zu entschlüsseln, erstellen Sie einen RSA Raw-Schlüsselbund mit dem privaten Schlüssel (oder key pair) und demselben Schlüsselnamespace und Namen.

Padding-Modus

Sie müssen einen Füllmodus für RSA Raw-Schlüsselringe angeben, die zur Verschlüsselung und Entschlüsselung verwendet werden, oder Funktionen Ihrer Sprachimplementierung verwenden, die ihn für Sie spezifizieren.

Der AWS Encryption SDK unterstützt die folgenden Füllmodi, die den Einschränkungen der einzelnen Sprachen unterliegen. Wir empfehlen einen Füllmodus, insbesondere bei SHA -256 und OAEP MGF1 bei SHA -256 [OAEP](#)Padding. Der [PKCS1](#)Padding-Modus wird nur aus Gründen der Abwärtskompatibilität unterstützt.

- OAEPmit SHA -1 und MGF1 mit SHA -1 Padding
- OAEPmit SHA -256 und MGF1 mit -256 Padding SHA
- OAEPmit SHA -384 und mit -384 Padding MGF1 SHA
- OAEPmit SHA -512 und mit -512 Padding MGF1 SHA
- PKCS1v1.5 Polsterung

Die folgenden Beispiele zeigen, wie Sie einen RSA Raw-Schlüsselbund mit dem öffentlichen und privaten Schlüssel eines RSA key pair und den Padding-Modus OAEP mit SHA -256 und MGF1 mit SHA -256 erstellen. Die `RSAPrivateKey` Variablen `RSAPublicKey` und stehen für das von Ihnen bereitgestellte Schlüsselmaterial.

C

Um einen RSA Raw-Schlüsselbund in der zu erstellen AWS-Verschlüsselungs-SDK for C, verwenden Sie `aws_cryptosdk_raw_rsa_keyring_new`.

Achten Sie beim Aufbau eines RSA Raw-Schlüsselbunds in der darauf AWS-Verschlüsselungs-SDK for C, dass Sie den Inhalt der PEM Datei, die jeden Schlüssel enthält, als Null-terminierte C-Zeichenfolge angeben, nicht als Pfad oder Dateiname. [Ein vollständiges Beispiel finden Sie unter `raw_rsa_keyring.c`.](#)

```
struct aws_allocator *alloc = aws_default_allocator();
```

```
AWS_STATIC_STRING_FROM_LITERAL(key_namespace, "HSM_01");
AWS_STATIC_STRING_FROM_LITERAL(key_name, "RSA_2048_06");

struct aws_cryptosdk_keyring *rawRsaKeyring = aws_cryptosdk_raw_rsa_keyring_new(
    alloc,
    key_namespace,
    key_name,
    private_key_from_pem,
    public_key_from_pem,
    AWS_CRYPTOSDK_RSA_OAEP_SHA256_MGF1);
```

C# / .NET

Um einen Raw-Schlüsselbund in der Form zu instanziiieren. RSA AWS Encryption SDK NET, verwende die Methode `materialProviders.CreateRawRsaKeyring()` Ein vollständiges Beispiel finden Sie unter [R awRSAKeyring Example.cs](#).

Das folgende Beispiel verwendet Version 4. x der AWS Encryption SDK Form. NET.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk = new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

var keyNamespace = "HSM_01";
var keyName = "RSA_2048_06";

// Get public and private keys from PEM files
var publicKey = new
    MemoryStream(System.IO.File.ReadAllBytes("RSAKeyringExamplePublicKey.pem"));
var privateKey = new
    MemoryStream(System.IO.File.ReadAllBytes("RSAKeyringExamplePrivateKey.pem"));

// Create the keyring input
var createRawRsaKeyringInput = new CreateRawRsaKeyringInput
{
    KeyNamespace = keyNamespace,
    KeyName = keyName,
    PaddingScheme = PaddingScheme.OAEP_SHA512_MGF1,
    PublicKey = publicKey,
    PrivateKey = privateKey
};
```

```
// Create the keyring
var rawRsaKeyring = materialProviders.CreateRawRsaKeyring(createRawRsaKeyringInput);
```

JavaScript Browser

Das AWS-Verschlüsselungs-SDK for JavaScript im Browser bezieht seine kryptografischen Primitive aus der [WebCrypto](#) Bibliothek. Bevor Sie den Schlüsselbund erstellen, müssen Sie das Rohschlüsselmaterial verwenden `importPublicKey()` und/oder `importPrivateKey()` in das Backend importieren. WebCrypto Dadurch wird sichergestellt, dass der Schlüsselbund vollständig ist, obwohl alle Aufrufe asynchron sind. WebCrypto Das Objekt, das die Importmethoden verwenden, beinhaltet den Wrapping-Algorithmus und seinen Padding-Modus.

Verwenden Sie nach dem Import des Schlüsselmaterials die `RawRsaKeyringWebCrypto()` Methode, um den Schlüsselbund zu instanziiieren. Beachten Sie bei der Erstellung eines RSA Raw-Schlüsselbundes die [mögliche Inkompatibilität](#) mit anderen Sprachimplementierungen.

JavaScript

Ein vollständiges Beispiel finden Sie unter [rsa_simple.ts](#) (Browser). JavaScript

```
const privateKey = await RawRsaKeyringWebCrypto.importPrivateKey(
  privateRsaJwkKey
)

const publicKey = await RawRsaKeyringWebCrypto.importPublicKey(
  publicRsaJwkKey
)

const keyNamespace = 'HSM_01'
const keyName = 'RSA_2048_06'

const keyring = new RawRsaKeyringWebCrypto({
  keyName,
  keyNamespace,
  publicKey,
  privateKey,
})
```

JavaScript Node.js

Um einen RSA Raw-Schlüsselbund in AWS-Verschlüsselungs-SDK for JavaScript Node.js zu instanziiieren, erstellen Sie eine neue Instanz der Klasse. `RawRsaKeyringNode` Der `wrapKey`

Parameter enthält den öffentlichen Schlüssel. Der `unwrapKey` Parameter enthält den privaten Schlüssel. Der `RawRsaKeyringNode` Konstruktor berechnet einen Standard-Füllmodus für Sie, obwohl Sie einen bevorzugten Füllmodus angeben können.

Achten Sie bei der Erstellung eines unformatierten RSA Schlüsselbundes auf [mögliche JavaScript Inkompatibilitäten](#) mit anderen Sprachimplementierungen.

Ein vollständiges Beispiel finden Sie unter [rsa_simple.ts](#) (Node.js). JavaScript

```
const keyNamespace = 'HSM_01'
const keyName = 'RSA_2048_06'

const keyring = new RawRsaKeyringNode({ keyName, keyNamespace, rsaPublicKey,
  rsaPrivateKey})
```

Java

```
final CreateRawRsaKeyringInput keyringInput = CreateRawRsaKeyringInput.builder()
    .keyName("RSA_2048_06")
    .keyNamespace("HSM_01")
    .paddingScheme(PaddingScheme.OAEP_SHA256_MGF1)
    .publicKey(RSAPublicKey)
    .privateKey(RSAPrivateKey)
    .build();
final MaterialProviders matProv = MaterialProviders.builder()
    .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
    .build();
IKeyring rawRsaKeyring = matProv.CreateRawRsaKeyring(keyringInput);
```

Rohe Schlüsselanhänger ECDH

Important

Der ECDH Raw-Schlüsselring ist nur mit Version 4 erhältlich. x der Form AWS Encryption SDK . NET und Version 3. x der AWS-Verschlüsselungs-SDK for Java. Der ECDH Raw-Schlüsselbund wurde in Version 1.5.0 der Material Providers Library eingeführt.

Der ECDH Raw-Schlüsselbund verwendet die öffentlich-privaten Schlüsselpaare mit elliptischer Kurve, die Sie angeben, um einen gemeinsamen Wrapping-Schlüssel zwischen zwei Parteien

abzuleiten. Zunächst leitet der Schlüsselbund mithilfe des privaten Schlüssels des Absenders, des öffentlichen Schlüssels des Empfängers und des Elliptic Curve Diffie-Hellman () - Schlüsselvereinbarungsalgorithmus ein gemeinsames Geheimnis ab. ECDH Anschließend leitet der Schlüsselbund anhand des gemeinsamen geheimen Schlüssels den gemeinsamen Wrapping-Schlüssel ab, der Ihre Datenverschlüsselungsschlüssel schützt. [Die Schlüsselableitungsfunktion, die \(KDF_CTR_HMAC_SHA384\) AWS Encryption SDK verwendet, um den gemeinsamen Wrapping-Schlüssel abzuleiten, entspricht den Empfehlungen für die Schlüsselableitung. NIST](#)

Die Funktion zur Schlüsselableitung gibt 64 Byte an Schlüsselmaterial zurück. Um sicherzustellen, dass beide Parteien das richtige Schlüsselmaterial verwenden, AWS Encryption SDK verwendet die ersten 32 Byte als Commitment-Schlüssel und die letzten 32 Byte als gemeinsamen Wrapping-Schlüssel. Wenn der Schlüsselbund beim Entschlüsseln nicht denselben Commitment-Schlüssel und denselben gemeinsamen Wrapping-Schlüssel reproduzieren kann, die im Chiffretext der Nachrichtenkopfzeile gespeichert sind, schlägt der Vorgang fehl. Wenn Sie beispielsweise Daten mit einem Schlüsselbund verschlüsseln, der mit Alices privatem Schlüssel und Bobs öffentlichem Schlüssel konfiguriert ist, reproduziert ein Schlüsselbund, der mit Bobs privatem Schlüssel und Alices öffentlichem Schlüssel konfiguriert ist, denselben Commitment-Schlüssel und gemeinsamen Wrapping-Schlüssel und kann die Daten entschlüsseln. Wenn Bobs öffentlicher Schlüssel aus einem AWS KMS key Paar stammt, kann Bob einen [AWS KMS ECDH Schlüsselbund](#) erstellen, um die Daten zu entschlüsseln.

Der ECDH Raw-Schlüsselbund verschlüsselt Daten mit einem symmetrischen Schlüssel unter Verwendung von -. AES GCM Der Datenschlüssel wird dann mit dem abgeleiteten gemeinsamen Wrapping-Schlüssel unter Verwendung von - umhüllverschlüsselt. AES GCM [Jeder ECDH Raw-Schlüsselbund kann nur einen gemeinsamen Wrap-Schlüssel haben, aber Sie können mehrere ECDH Raw-Schlüsselanhänger, einzeln oder zusammen mit anderen Schlüsselbunden, in einen Mehrfachschlüsselbund aufnehmen.](#)

Sie sind dafür verantwortlich, Ihre privaten Schlüssel zu generieren, zu speichern und zu schützen, vorzugsweise in einem Hardware-Sicherheitsmodul (HSM) oder einem Schlüsselverwaltungssystem. Die Schlüsselpaare des Absenders und des Empfängers müssen sich auf derselben elliptischen Kurve befinden. Der AWS Encryption SDK unterstützt die folgenden Spezifikationen für elliptische Kurven:

- ECC_NIST_P256
- ECC_NIST_P384
- ECC_NIST_P512

Einen Raw-Schlüsselbund erstellen ECDH

Der ECDH Raw-Schlüsselbund unterstützt drei wichtige Vereinbarungsschemata: `RawPrivateKeyToStaticPublicKey`, `EphemeralPrivateKeyToStaticPublicKey` und `PublicKeyDiscovery`. Das von Ihnen gewählte Schlüsselvereinbarungsschema bestimmt, welche kryptografischen Operationen Sie ausführen können und wie die Schlüsselmaterialien zusammengestellt werden.

Themen

- [RawPrivateKeyToStaticPublicKey](#)
- [EphemeralPrivateKeyToStaticPublicKey](#)
- [PublicKeyDiscovery](#)

RawPrivateKeyToStaticPublicKey

Verwenden Sie das `RawPrivateKeyToStaticPublicKey` Schlüsselvereinbarungsschema, um den privaten Schlüssel des Absenders und den öffentlichen Schlüssel des Empfängers im Schlüsselbund statisch zu konfigurieren. Dieses Schlüsselvereinbarungsschema kann Daten ver- und entschlüsseln.

Um einen ECDH Raw-Schlüsselbund mit dem Schlüsselvereinbarungsschema zu initialisieren, geben Sie die `RawPrivateKeyToStaticPublicKey` folgenden Werte an:

- Der private Schlüssel des Absenders

[Sie müssen den in 5958 definierten PEM privaten Schlüssel des Absenders \(PKCS#8 PrivateKeyInfo -Strukturen\) angeben. RFC](#)

- Der öffentliche Schlüssel des Empfängers

[Sie müssen den DER -codierten öffentlichen X.509-Schlüssel des Empfängers, auch bekannt als SubjectPublicKeyInfo \(SPKI\), wie in 5280 definiert, angeben. RFC](#)

Sie können den öffentlichen Schlüssel eines Schlüsselpaars mit asymmetrischer KMS Schlüsselvereinbarung oder den öffentlichen Schlüssel eines außerhalb von AWS generierten key pair angeben.

- Kurvenspezifikation

Identifiziert die Spezifikation für elliptische Kurven in den angegebenen Schlüsselpaaren. Sowohl die Schlüsselpaare des Absenders als auch des Empfängers müssen dieselbe Kurvenspezifikation haben.

Zulässige Werte: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

C# / .NET

```
// Instantiate material providers
var materialProviders = new MaterialProviders(new MaterialProvidersConfig());
var BobPrivateKey = new MemoryStream(new byte[] { });
var AlicePublicKey = new MemoryStream(new byte[] { });

// Create the Raw ECDH static keyring
var staticConfiguration = new RawEcdhStaticConfigurations()
{
    RawPrivateKeyToStaticPublicKey = new RawPrivateKeyToStaticPublicKeyInput
    {
        SenderStaticPrivateKey = BobPrivateKey,
        RecipientPublicKey = AlicePublicKey
    }
};

var createKeyringInput = new CreateRawEcdhKeyringInput()
{
    CurveSpec = ECDHCurveSpec.ECC_NIST_P256,
    KeyAgreementScheme = staticConfiguration
};

var keyring = materialProviders.CreateRawEcdhKeyring(createKeyringInput);
```

Java

Das folgende Java-Beispiel verwendet das `RawPrivateKeyToStaticPublicKey` Schlüsselvereinbarungsschema, um den privaten Schlüssel des Absenders und den öffentlichen Schlüssel des Empfängers statisch zu konfigurieren. Beide Schlüsselpaare befinden sich auf der `ECC_NIST_P256` Kurve.

```
private static void StaticRawKeyring() {
    // Instantiate material providers
    final MaterialProviders materialProviders =
```

```

MaterialProviders.builder()
    .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
    .build();

KeyPair senderKeys = GetRawEccKey();
KeyPair recipient = GetRawEccKey();

// Create the Raw ECDH static keyring
final CreateRawEcdhKeyringInput rawKeyringInput =
    CreateRawEcdhKeyringInput.builder()
        .curveSpec(ECDHCurveSpec.ECC_NIST_P256)
        .KeyAgreementScheme(
            RawEcdhStaticConfigurations.builder()
                .RawPrivateKeyToStaticPublicKey(
                    RawPrivateKeyToStaticPublicKeyInput.builder()
                        // Must be a PEM-encoded private key

                )
                .senderStaticPrivateKey(ByteBuffer.wrap(senderKeys.getPrivate().getEncoded()))
                // Must be a DER-encoded X.509 public key

                .recipientPublicKey(ByteBuffer.wrap(recipient.getPublic().getEncoded()))
                .build()
            )
            .build()
        ).build();

final IKeyring staticKeyring =
materialProviders.CreateRawEcdhKeyring(rawKeyringInput);
}

```

EphemeralPrivateKeyToStaticPublicKey

Mit dem Schlüsselvereinbarungsschema konfigurierte

`EphemeralPrivateKeyToStaticPublicKey` Schlüsselringe erstellen lokal ein neues key pair und leiten für jeden Verschlüsselungsaufwurf einen eindeutigen gemeinsamen Wrapping-Schlüssel ab.

Dieses Schlüsselvereinbarungsschema kann nur Nachrichten verschlüsseln. Um Nachrichten zu entschlüsseln, die mit dem `EphemeralPrivateKeyToStaticPublicKey` Schlüsselvereinbarungsschema verschlüsselt wurden, müssen Sie ein Discovery-Schlüsselvereinbarungsschema verwenden, das mit dem öffentlichen Schlüssel desselben Empfängers konfiguriert ist. Zum Entschlüsseln können Sie einen ECDH Raw-Schlüsselbund

mit dem [PublicKeyDiscovery](#) Schlüsselvereinbarungsalgorithmus verwenden. Wenn der öffentliche Schlüssel des Empfängers aus einem asymmetrischen key pair für die Schlüsselvereinbarung stammt, können Sie einen KMS AWS KMS ECDH Schlüsselbund mit dem [KmsPublicKeyDiscovery](#) Schlüsselvereinbarungsschema verwenden.

Um einen ECDH Raw-Schlüsselbund mit dem Schlüsselvereinbarungsschema zu initialisieren, geben Sie die `EphemeralPrivateKeyToStaticPublicKey` folgenden Werte an:

- Der öffentliche Schlüssel des Empfängers

[Sie müssen den DER -codierten öffentlichen X.509-Schlüssel des Empfängers, auch bekannt als SubjectPublicKeyInfo \(SPKI\), wie in 5280 definiert, angeben. RFC](#)

Sie können den öffentlichen Schlüssel eines Schlüsselpaars mit asymmetrischer KMS Schlüsselvereinbarung oder den öffentlichen Schlüssel eines außerhalb von AWS generierten key pair angeben.

- Kurvenspezifikation

Identifiziert die Spezifikation für elliptische Kurven im angegebenen öffentlichen Schlüssel.

Beim Verschlüsseln erstellt der Schlüsselbund ein neues key pair auf der angegebenen Kurve und verwendet den neuen privaten Schlüssel und den angegebenen öffentlichen Schlüssel, um einen gemeinsamen Wrapping-Schlüssel abzuleiten.

Zulässige Werte: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

C# / .NET

Im folgenden Beispiel wird ein ECDH Raw-Schlüsselbund mit dem `EphemeralPrivateKeyToStaticPublicKey` Schlüsselvereinbarungsschema erstellt. Beim Verschlüsseln erstellt der Schlüsselbund lokal auf der angegebenen `ECC_NIST_P256` Kurve ein neues key pair.

```
// Instantiate material providers
var materialProviders = new MaterialProviders(new MaterialProvidersConfig());
    var AlicePublicKey = new MemoryStream(new byte[] { });

// Create the Raw ECDH ephemeral keyring
var ephemeralConfiguration = new RawEcdhStaticConfigurations()
{
```

```
    EphemeralPrivateKeyToStaticPublicKey = new
    EphemeralPrivateKeyToStaticPublicKeyInput
    {
        RecipientPublicKey = AlicePublicKey
    }
};

var createKeyringInput = new CreateRawEcdhKeyringInput()
{
    CurveSpec = ECDHCurveSpec.ECC_NIST_P256,
    KeyAgreementScheme = ephemeralConfiguration
};

var keyring = materialProviders.CreateRawEcdhKeyring(createKeyringInput);
```

Java

Im folgenden Beispiel wird ein ECDH Raw-Schlüsselbund mit dem Schlüsselvereinbarungsschema `EphemeralPrivateKeyToStaticPublicKey` erstellt. Beim Verschlüsseln erstellt der Schlüsselbund lokal auf der angegebenen `ECC_NIST_P256` Kurve ein neues key pair.

```
private static void EphemeralRawEcdhKeyring() {
    // Instantiate material providers
    final MaterialProviders materialProviders =
        MaterialProviders.builder()
            .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
            .build();

    ByteBuffer recipientPublicKey = getPublicKeyBytes();

    // Create the Raw ECDH ephemeral keyring
    final CreateRawEcdhKeyringInput ephemeralInput =
        CreateRawEcdhKeyringInput.builder()
            .curveSpec(ECDHCurveSpec.ECC_NIST_P256)
            .KeyAgreementScheme(
                RawEcdhStaticConfigurations.builder()
                    .EphemeralPrivateKeyToStaticPublicKey(
                        EphemeralPrivateKeyToStaticPublicKeyInput.builder()
                            .recipientPublicKey(recipientPublicKey)
                            .build()
                    )
                )
            .build()
}
```

```
    ).build();

    final IKeyring ephemeralKeyring =
materialProviders.CreateRawEcdhKeyring(ephemeralInput);
}
```

PublicKeyDiscovery

Beim Entschlüsseln empfiehlt es sich, die Umschließungsschlüssel anzugeben, die sie verwenden können. AWS Encryption SDK Um dieser bewährten Methode zu folgen, verwenden Sie einen ECDH Schlüsselbund, der sowohl den privaten Schlüssel des Absenders als auch den öffentlichen Schlüssel des Empfängers angibt. Sie können jedoch auch einen Raw ECDH Discovery-Schlüsselbund erstellen, d. h. einen ECDH Raw-Schlüsselbund, mit dem jede Nachricht entschlüsselt werden kann, bei der der öffentliche Schlüssel des angegebenen Schlüssels mit dem öffentlichen Schlüssel des Empfängers übereinstimmt, der im Chiffretext der Nachricht gespeichert ist. Dieses Schlüsselvereinbarungsschema kann nur Nachrichten entschlüsseln.

Important

Wenn Sie Nachrichten mithilfe des PublicKeyDiscovery Schlüsselvereinbarungsschemas entschlüsseln, akzeptieren Sie alle öffentlichen Schlüssel, unabhängig davon, wem sie gehören.

Um einen ECDH Raw-Schlüsselbund mit dem PublicKeyDiscovery Schlüsselvereinbarungsschema zu initialisieren, geben Sie die folgenden Werte an:

- Der statische private Schlüssel des Empfängers

[Sie müssen den PEM -codierten privaten Schlüssel des Empfängers \(PKCS#8 PrivateKeyInfo - Strukturen\) angeben, wie in RFC 5958 definiert.](#)

- Spezifikation der Kurve

Identifiziert die Spezifikation für elliptische Kurven im angegebenen privaten Schlüssel. Sowohl die Schlüsselpaare des Absenders als auch des Empfängers müssen dieselbe Kurvenspezifikation haben.

Zulässige Werte: ECC_NIST_P256, ECC_NIS_P384, ECC_NIST_P512

C# / .NET

Im folgenden Beispiel wird ein ECDH Raw-Schlüsselbund mit dem `PublicKeyDiscovery` Schlüsselvereinbarungsschema erstellt. Dieser Schlüsselbund kann jede Nachricht entschlüsseln, bei der der öffentliche Schlüssel des angegebenen privaten Schlüssels mit dem öffentlichen Schlüssel des Empfängers übereinstimmt, der im Chiffretext der Nachricht gespeichert ist.

```
// Instantiate material providers
var materialProviders = new MaterialProviders(new MaterialProvidersConfig());
    var AlicePrivateKey = new MemoryStream(new byte[] { });

// Create the Raw ECDH discovery keyring
var discoveryConfiguration = new RawEcdhStaticConfigurations()
{
    PublicKeyDiscovery = new PublicKeyDiscoveryInput
    {
        RecipientStaticPrivateKey = AlicePrivateKey
    }
};

var createKeyringInput = new CreateRawEcdhKeyringInput()
{
    CurveSpec = ECDHCurveSpec.ECC_NIST_P256,
    KeyAgreementScheme = discoveryConfiguration
};

var keyring = materialProviders.CreateRawEcdhKeyring(createKeyringInput);
```

Java

Im folgenden Beispiel wird ein ECDH Raw-Schlüsselbund mit dem Schlüsselvereinbarungsschema erstellt. `PublicKeyDiscovery` Dieser Schlüsselbund kann jede Nachricht entschlüsseln, bei der der öffentliche Schlüssel des angegebenen privaten Schlüssels mit dem öffentlichen Schlüssel des Empfängers übereinstimmt, der im Chiffretext der Nachricht gespeichert ist.

```
private static void RawEcdhDiscovery() {
    // Instantiate material providers
    final MaterialProviders materialProviders =
        MaterialProviders.builder()
            .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
            .build();
```

```
KeyPair recipient = GetRawEccKey();

// Create the Raw ECDH discovery keyring
final CreateRawEcdhKeyringInput rawKeyringInput =
    CreateRawEcdhKeyringInput.builder()
        .curveSpec(ECDHCurveSpec.ECC_NIST_P256)
        .KeyAgreementScheme(
            RawEcdhStaticConfigurations.builder()
                .PublicKeyDiscovery(
                    PublicKeyDiscoveryInput.builder()
                        // Must be a PEM-encoded private key
                )
            )
        .recipientStaticPrivateKey(ByteBuffer.wrap(sender.getPrivate().getEncoded()))
        .build()
    ).build();

final IKeyring publicKeyDiscovery =
    materialProviders.CreateRawEcdhKeyring(rawKeyringInput);
}
```

Multi-Schlüsselbunde

Sie können Schlüsselbunde zu einem Multi-Schlüsselbund kombinieren. Ein Multi-Schlüsselbund ist ein Schlüsselbund, der aus einem oder mehreren einzelnen Schlüsselbunden desselben oder eines anderen Typs besteht. Das hat den gleichen Effekt wie die Verwendung von mehreren Schlüsselbunden in einer Reihe. Wenn Sie einen Multi-Schlüsselbund verwenden, um Daten zu verschlüsseln, können alle Umhüllungsschlüssel in einem seiner Schlüsselbunde diese Daten entschlüsseln.

Wenn Sie einen Multi-Schlüsselbund erstellen, um Daten zu verschlüsseln, geben Sie einen der Schlüsselbunde als Generator-Schlüsselbund an. Alle anderen Schlüsselbunde werden als untergeordnete Schlüsselbunde bezeichnet. Der Generator-Schlüsselbund generiert und verschlüsselt den Klartext-Datenschlüssel. Anschließend verschlüsseln alle Umhüllungsschlüssel in den untergeordneten Schlüsselbunden den gleichen Klartext-Datenschlüssel. Der Multi-Schlüsselbund gibt den Klartext-Datenschlüssel und einen verschlüsselten Datenschlüssel für jeden Umhüllungsschlüssel im Multi-Schlüsselbund zurück. Wenn der Generator-Schlüsselbund ein [KMSSchlüsselbund](#) ist, generiert und verschlüsselt der Generatorschlüssel im AWS KMS

Schlüsselbund den Klartext-Schlüssel. Dann verschlüsseln alle zusätzlichen Schlüssel AWS KMS keys im AWS KMS Schlüsselbund und alle Schlüssel in allen untergeordneten Schlüsselbunden im Mehrfachschlüsselbund denselben Klartext-Schlüssel.

Wenn Sie einen Mehrfachschlüsselbund ohne Generatorschlüsselbund erstellen, können Sie ihn eigenständig zum Entschlüsseln von Daten verwenden, aber nicht zum Verschlüsseln. Oder wenn Sie bei Verschlüsselungsvorgängen einen Mehrfachschlüsselbund ohne Generatorschlüsselbund verwenden möchten, können Sie ihn als untergeordneten Schlüsselbund in einem anderen Schlüsselbund angeben. Ein Mehrfachschlüsselbund ohne Generatorschlüsselbund kann nicht als Generator-Schlüsselbund in einem anderen Schlüsselbund mit mehreren Schlüsseln bezeichnet werden.

Beim Entschlüsseln versucht der AWS Encryption SDK anhand der Schlüsselbunde, einen der verschlüsselten Datenschlüssel zu entschlüsseln. Die Schlüsselbunde werden in der Reihenfolge aufgerufen, in der sie im Multi-Schlüsselbund angegeben sind. Die Verarbeitung stoppt, sobald ein Schlüssel in einem Schlüsselbund einen verschlüsselten Datenschlüssel entschlüsseln kann.

[Ab Version 1.7.](#) x, wenn ein verschlüsselter Datenschlüssel unter einem AWS Key Management Service (AWS KMS) -Schlüsselbund (oder einem Hauptschlüsselanbieter) verschlüsselt wird, übergibt der AWS Encryption SDK immer den Schlüssel ARN von AWS KMS key an den `KeyId` Parameter der AWS KMS [Decrypt-Operation](#). Dies ist eine AWS KMS bewährte Methode, die sicherstellt, dass Sie den verschlüsselten Datenschlüssel mit dem Umschließungsschlüssel entschlüsseln, den Sie verwenden möchten.

Ein funktionierendes Beispiel für einen Multi-Schlüsselbund finden Sie unter:

- C: [multi_keyring.cpp](#)
- C#/. NET: [.cs MultiKeyringExample](#)
- JavaScript [Node.js: multi_keyring.ts](#)
- JavaScript Browser: [multi_keyring.ts](#)
- Java [MultiKeyringExample: .java](#)

Zum Erstellen eines Multi-Schlüsselbunds müssen Sie zuerst die untergeordneten Schlüsselbunde instanzieren. In diesem Beispiel verwenden wir einen AWS KMS Schlüsselbund und einen AES Raw-Schlüsselbund, aber Sie können jeden unterstützten Schlüsselbund in einem Mehrfachschlüsselbund kombinieren.

C

```

/* Define an AWS KMS keyring. For details, see string.cpp */
struct aws_cryptosdk_keyring *kms_keyring =
    Aws::Cryptosdk::KmsKeyring::Builder().Build(example_key);

// Define a Raw AES keyring. For details, see raw\_aes\_keyring.c */
struct aws_cryptosdk_keyring *aes_keyring = aws_cryptosdk_raw_aes_keyring_new(
    alloc, wrapping_key_namespace, wrapping_key_name, wrapping_key,
    AWS_CRYPTOSDK_AES256);

```

C# / .NET

```

// Define an AWS KMS keyring. For details, see AwsKmsKeyringExample.cs.
var kmsKeyring = materialProviders.CreateAwsKmsKeyring(createKmsKeyringInput);

// Define a Raw AES keyring. For details, see RawAESKeyringExample.cs.
var aesKeyring = materialProviders.CreateRawAesKeyring(createAesKeyringInput);

```

JavaScript Browser

```

const clientProvider = getClient(KMS, { credentials })

// Define an AWS KMS keyring. For details, see kms\_simple.ts.
const kmsKeyring = new KmsKeyringBrowser({ generatorKeyId: exampleKey })

// Define a Raw AES keyring. For details, see aes\_simple.ts.
const aesKeyring = new RawAesKeyringWebCrypto({ keyName, keyNamespace,
    wrappingSuite, masterKey })

```

JavaScript Node.js

```

// Define an AWS KMS keyring. For details, see kms\_simple.ts.
const kmsKeyring = new KmsKeyringNode({ generatorKeyId: exampleKey })

// Define a Raw AES keyring. For details, see raw\_aes\_keyring\_node.ts.
const aesKeyring = new RawAesKeyringNode({ keyName, keyNamespace, wrappingSuite,
    unencryptedMasterKey })

```

Java

```

// Define the raw AES keyring.

```

```

final MaterialProviders matProv = MaterialProviders.builder()
    .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
    .build();
final CreateRawAesKeyringInput createRawAesKeyringInput =
    CreateRawAesKeyringInput.builder()
        .keyName("AES_256_012")
        .keyNamespace("HSM_01")
        .wrappingKey(AESWrappingKey)
        .wrappingAlg(AesWrappingAlg.ALG_AES256_GCM_IV12_TAG16)
        .build();
IKeyring rawAesKeyring = matProv.CreateRawAesKeyring(createRawAesKeyringInput);

// Define the AWS KMS keyring.
final CreateAwsKmsMrkMultiKeyringInput createAwsKmsMrkMultiKeyringInput =
    CreateAwsKmsMrkMultiKeyringInput.builder()
        .generator(kmsKeyArn)
        .build();
IKeyring awsKmsMrkMultiKeyring =
    matProv.CreateAwsKmsMrkMultiKeyring(createAwsKmsMrkMultiKeyringInput);

```

Erstellen Sie als Nächstes den Multi-Schlüsselbund und geben Sie seinen Generator-Schlüsselbund an, falls vorhanden. In diesem Beispiel erstellen wir einen Mehrfachschlüsselbund, bei dem der Schlüsselbund der Generator-Schlüsselbund und der AWS KMS Schlüsselbund der untergeordnete Schlüsselbund ist. AES

C

Im Multi-Schlüsselbund-Konstruktor in C geben Sie nur den Generator-Schlüsselbund an.

```

struct aws_cryptosdk_keyring *multi_keyring = aws_cryptosdk_multi_keyring_new(alloc,
    kms_keyring);

```

Verwenden Sie die `aws_cryptosdk_multi_keyring_add_child`-Methode, um einen untergeordneten Schlüsselbund zu Ihrem Multi-Schlüsselbund hinzuzufügen. Sie müssen die Methode einmal für jeden untergeordneten Schlüsselbund aufrufen, den Sie hinzufügen.

```

// Add the Raw AES keyring (C only)
aws_cryptosdk_multi_keyring_add_child(multi_keyring, aes_keyring);

```

C# / .NET

Der `NETCreateMultiKeyringInput` Mit dem Konstruktor können Sie einen Generator-Schlüsselbund und untergeordnete Schlüsselringe definieren. Das resultierende `CreateMultiKeyringInput` Objekt ist unveränderlich.

```
var createMultiKeyringInput = new CreateMultiKeyringInput
{
    Generator = kmsKeyring,
    ChildKeyrings = new List<IKeyring>() {aesKeyring}
};

var multiKeyring = materialProviders.CreateMultiKeyring(createMultiKeyringInput);
```

JavaScript Browser

JavaScript Mehrfachschlüsselringe sind unveränderlich. Mit dem Konstruktor für JavaScript mehrere Schlüsselbunde können Sie den Generator-Schlüsselbund und mehrere untergeordnete Schlüsselringe angeben.

```
const clientProvider = getClient(KMS, { credentials })

const multiKeyring = new MultiKeyringWebCrypto(generator: kmsKeyring, children:
[aesKeyring]);
```

JavaScript Node.js

JavaScript Schlüsselanhänger mit mehreren Schlüsseln sind unveränderlich. Mit dem Konstruktor für JavaScript mehrere Schlüsselbunde können Sie den Generator-Schlüsselbund und mehrere untergeordnete Schlüsselringe angeben.

```
const multiKeyring = new MultiKeyringNode(generator: kmsKeyring, children:
[aesKeyring]);
```

Java

Mit dem `CreateMultiKeyringInput` Java-Konstruktor können Sie einen Generator-Schlüsselbund und untergeordnete Schlüsselringe definieren. Das resultierende `createMultiKeyringInput` Objekt ist unveränderlich.

```
final CreateMultiKeyringInput createMultiKeyringInput =  
    CreateMultiKeyringInput.builder()  
        .generator(awsKmsMrkMultiKeyring)  
        .childKeyrings(Collections.singletonList(rawAesKeyring))  
        .build();  
IKeyring multiKeyring = matProv.CreateMultiKeyring(createMultiKeyringInput);
```

Jetzt können Sie mit dem Multi-Schlüsselbund Daten ver- und entschlüsseln.

AWS Encryption SDK-Programmiersprachen

Der AWS Encryption SDK für die folgenden Programmiersprachen verfügbar. Alle Sprachimplementierungen sind interoperabel. Sie können mit einer Sprachimplementierung verschlüsseln und mit einer anderen entschlüsseln. Die Interoperabilität ist möglicherweise von Spracheinschränkungen abhängig. Wenn dies der Fall ist, werden diese Einschränkungen im Thema zur Sprachimplementierung beschrieben. Außerdem müssen Sie beim Verschlüsseln und Entschlüsseln kompatible Schlüsselbünde oder Masterschlüssel und Masterschlüsselanbieter verwenden. Details hierzu finden Sie unter [the section called “Schlüsselbund-Kompatibilität”](#).

Themen

- [AWS-Verschlüsselungs-SDK for C](#)
- [AWS Encryption SDK für .NET](#)
- [AWS-Verschlüsselungs-SDK for Java](#)
- [AWS-Verschlüsselungs-SDK for JavaScript](#)
- [AWS-Verschlüsselungs-SDK for Python](#)
- [AWS Encryption SDK-Befehlszeilenschnittstelle](#)

AWS-Verschlüsselungs-SDK for C

Das AWS-Verschlüsselungs-SDK for C bietet eine clientseitige Verschlüsselungsbibliothek für Entwickler, die Anwendungen in C schreiben. Es dient auch als Grundlage für Implementierungen des AWS Encryption SDK in übergeordneten Programmiersprachen.

Wie alle Implementierungen des AWS Encryption SDK bietet auch die AWS-Verschlüsselungs-SDK for C erweiterte Datenschutzfunktionen. Dazu gehören die [Envelope-Verschlüsselung](#), zusätzliche authentifizierte Daten (AAD) und [Algorithmen-Pakete](#) mit sicherem, authentifiziertem, symmetrischem Schlüssel, wie z. B. 256-Bit-AES-GCM mit Schlüsselableitung und Signatur.

Alle sprachspezifischen Implementierungen des AWS Encryption SDK sind vollständig interoperabel. Beispielsweise können Sie Daten mit der verschlüsseln AWS-Verschlüsselungs-SDK for C und entschlüsseln Sie es mit [jede unterstützte Sprachimplementierung](#), einschließlich der [AWSCLI für Verschlüsselung](#) aus.

Die AWS-Verschlüsselungs-SDK for C erfordert das AWS SDK for C++ um mit zu interaktion AWS Key Management Service (AWS KMS) enthalten. Sie müssen es nur verwenden, wenn Sie das optionale

verwenden [AWS KMSSchlüsselring](#) aus. Das AWS Encryption SDK ist jedoch nicht von AWS KMS oder einem anderen AWS-Service abhängig.

Weitere Informationen

- Für Details über die Programmierung mit der AWS-Verschlüsselungs-SDK for C, finden Sie unter [C-Beispiele](#), der [Beispiele im aws-encryption-sdk-c Repository](#) auf GitHub, und das [AWS-Verschlüsselungs-SDK for C API-Dokumentation](#) aus.
- Für eine Diskussion darüber, wie man die AWS-Verschlüsselungs-SDK for C um Daten zu verschlüsseln, damit Sie sie in mehreren entschlüsseln können AWS-Regionen, finden Sie unter [Wie entschlüsselt man Chiffretexte in mehreren Regionen mit AWS Encryption SDK in C](#) im AWS Blog-Sicherheit

Themen

- [Installieren des AWS-Verschlüsselungs-SDK for C](#)
- [Verwendung der AWS-Verschlüsselungs-SDK for C](#)
- [AWS-Verschlüsselungs-SDK for C Beispiele für](#)

Installieren des AWS-Verschlüsselungs-SDK for C

Installieren der neuesten Version von AWS-Verschlüsselungs-SDK for C.

Note

Alle Versionen der AWS-Verschlüsselungs-SDK for C früher als 2.0.0 [end-of-support Phase](#). Sie können sicher von Version 2.0 aktualisieren und später zur neuesten Version des AWS-Verschlüsselungs-SDK for C ohne Code- oder Datenänderungen. Allerdings [neue Sicherheitsfunktionen](#) eingeführt in Version 2.0.x sind nicht abwärtskompatibel. Um von Versionen vor 1.7 zu aktualisieren auf Version 2.0.x und später müssen Sie zuerst auf die neueste Version 1 aktualisieren Version des AWS-Verschlüsselungs-SDK for C. Details hierzu finden Sie unter [Migrieren Sie Ihre AWS Encryption SDK](#).

Sie finden eine detaillierte Anleitung für die Installation und den Bau des AWS-Verschlüsselungs-SDK for C in der [README-Datei](#) der [aws-encryption-sdk-c](#) Repository. Es enthält Anweisungen zum Aufbau auf Amazon Linux-, Ubuntu-, MacOS- und Windows-Plattformen.

Bevor Sie beginnen, entscheiden Sie, ob Sie [AWS KMSSchlüsselringe](#) in der AWS Encryption SDK. Bei Verwendung einer AWS KMSSchlüsselring, müssen Sie das AWS SDK for C++. Die AWS SDK ist für die Interaktion mit erforderlich [AWS Key Management Service](#) (AWS KMS) enthalten. Bei Verwendung AWS KMSSchlüsselringe, das AWS Encryption SDK Verwendungszwecke AWS KMS um die Verschlüsselungsschlüssel zu generieren und zu schützen, die Ihre Daten schützen.

Es ist nicht erforderlich, das AWS SDK for C++ wenn Sie einen anderen Schlüsselanhängertyp verwenden, z. B. einen unbearbeiteten AES-Schlüsselbund, einen RSA-RSA-Schlüsselbund oder einen Mehrfachschlüsselbund, der keine AWS KMSSchlüsselring. Wenn Sie jedoch einen Raw-Schlüsselbundtyp verwenden, müssen Sie Ihre eigenen unverarbeiteten Wrapping-Schlüssel generieren und schützen.

Hilfe bei der Entscheidung, welche Schlüsselbundtypen verwendet werden sollen, finden Sie unter [the section called "Einen Schlüsselbund wählen"](#).

Wenn Probleme mit der Installation auftreten, [ein Problem einreichen](#) in der `aws-encryption-sdk`-Archivierung oder einen der Feedback-Links auf dieser Seite verwenden.

Verwendung der AWS-Verschlüsselungs-SDK for C

Dieses Thema erklärt einige der Funktionen des AWS-Verschlüsselungs-SDK for C, die in Implementierungen anderer Programmiersprachen nicht unterstützt werden.

Die Beispiele in diesem Abschnitt zeigen, wie Sie verwenden können [Version 2.0.x](#) und später von der AWS-Verschlüsselungs-SDK for C aus. Für Beispiele, die frühere Versionen verwenden, finden Sie Ihre Version im [Versionen](#)-Liste des [aws-encryption-sdk-c Repository](#)-Repository auf GitHub.

Weitere Informationen zur Programmierung mit dem AWS-Verschlüsselungs-SDK for C finden Sie in den [C-Beispielen](#), in den [Beispielen](#) im GitHub-Repository [aws-encryption-sdk-c](#) und in der [AWS-Verschlüsselungs-SDK for C- API-Dokumentation](#).

Weitere Informationen finden Sie auch unter: [Verwenden von Schlüsselbunden](#).

Themen

- [Muster zum Ver- und Entschlüsseln von Daten](#)
- [Referenzzählung](#)

Muster zum Ver- und Entschlüsseln von Daten

Wenn Sie das AWS-Verschlüsselungs-SDK for C verwenden, folgen Sie einem Muster, das in etwa so aussieht: Erstellen eines [Schlüsselbunds](#), Erstellen eines [CMM](#), der den Schlüsselbund verwendet, Erstellen einer Sitzung, die den CMM (und Schlüsselbund) verwendet, und Verarbeiten der Sitzung.

1. Ladefehlerzeichenfolgen.

Rufen Sie `denaws_cryptosdk_load_error_strings()`-Methode in Ihrem C- oder C++-Code. Es lädt Fehlerinformationen, die für das Debuggen sehr nützlich sind.

Sie müssen es nur einmal anrufen, z. B. in Ihrer `main`-Methode.

```
/* Load error strings for debugging */
aws_cryptosdk_load_error_strings();
```

2. Erstellen Sie einen Schlüsselbund.

Konfigurieren Sie Ihren [Schlüsselbund](#) mit dem Umhüllungsschlüssel, den Sie verwenden möchten, um Ihre Daten zu verschlüsseln. In diesem Beispiel verwenden wir einen [AWS KMSSchlüsselring](#) mit einem AWS KMS key, den Sie verwenden können, Sie können aber jede Art von Schlüsselbund verwenden.

So identifizieren Sie ein AWS KMS key in einem Verschlüsselungsschlüsselbund im AWS-Verschlüsselungs-SDK for C, geben Sie ein [Schlüssel-ARN](#) oder [Alias-ARN](#) aus. In einem Entschlüsselungsschlüsselbund müssen Sie einen Schlüssel-ARN verwenden. Details hierzu finden Sie unter [Identifizierung AWS KMS keys in einem AWS KMS Schlüsselbund](#).

```
const char * KEY_ARN = "arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"  
struct aws_cryptosdk_keyring *kms_keyring =  
    Aws::Cryptosdk::KmsKeyring::Builder().Build(KEY_ARN);
```

3. Erstellen Sie eine Sitzung.

Im AWS-Verschlüsselungs-SDK for C verwenden Sie eine Sitzung zum Verschlüsseln einer einzelnen Klartextnachricht oder zum Entschlüsseln einer einzelnen Verschlüsselungstext-Nachricht, unabhängig von ihrer Größe. Die Sitzung behält den Status der Nachricht während der Verarbeitung.

Konfigurieren Sie Ihre Sitzung mit einem Allocator, einem Schlüsselbund und einem Modus: `AWS_CRYPTOSDK_ENCRYPT` oder `AWS_CRYPTOSDK_DECRYPT`. Wenn Sie den Modus der Sitzung ändern müssen, verwenden Sie die `aws_cryptosdk_session_reset`-Methode.

Wenn Sie eine Sitzung mit einem Schlüsselbund erstellen, wird der AWS-Verschlüsselungs-SDK für C erstellt automatisch einen Standard-Manager von kryptographischen Materialien (CMM) für Sie. Sie müssen dieses Objekt nicht erstellen, pflegen oder zerstören.

Beispiel: Die folgende Sitzung verwendet den Allocator und den Schlüsselbund, der in Schritt 1 definiert wurde. Wenn Sie Daten verschlüsseln, lautet der Modus `AWS_CRYPTOSDK_ENCRYPT`.

```
struct aws_cryptosdk_session * session =
    aws_cryptosdk_session_new_from_keyring_2(allocator, AWS_CRYPTOSDK_ENCRYPT,
        kms_keyring);
```

4. Verschlüsseln oder entschlüsseln Sie die Daten.

Um die Daten in der Sitzung zu verarbeiten, verwenden Sie die `aws_cryptosdk_session_process`-Methode. Wenn der Eingabepuffer groß genug ist, um den gesamten Klartext zu speichern und der Ausgabepuffer groß genug ist, um den gesamten Verschlüsselungstext zu speichern, können Sie `aws_cryptosdk_session_process_full` aufrufen. Wenn Sie jedoch Streaming-Daten verarbeiten müssen, können Sie `aws_cryptosdk_session_process` in einer Schleife aufrufen. Ein Beispiel finden Sie im Beispiel [file_streaming.cpp](#). Die `aws_cryptosdk_session_process_full` wird in den AWS Encryption SDK Versionen 1.9.x und 2.2.x aus.

Wenn die Sitzung für das Verschlüsseln von Daten konfiguriert ist, beschreiben die Klartextfelder die Eingabe und die Verschlüsselungstext-Felder die Ausgabe. Das `plaintext`-Feld enthält die Nachricht, die Sie verschlüsseln möchten, und das `ciphertext`-Feld ruft die [verschlüsselte Nachricht](#) ab, die die Verschlüsselungsmethode zurückgibt.

```
/* Encrypting data */
aws_cryptosdk_session_process_full(session,
                                   ciphertext,
                                   ciphertext_buffer_size,
                                   &ciphertext_length,
                                   plaintext,
                                   plaintext_length)
```

Wenn die Sitzung für das Entschlüsseln von Daten konfiguriert ist, beschreiben die Verschlüsselungstext-Felder die Eingabe und die Klartextfelder die Ausgabe. Das `ciphertext`-Feld enthält die [verschlüsselte Nachricht](#), die die Verschlüsselungsmethode zurückgegeben hat, und das `plaintext`-Feld ruft die Klartextnachricht ab, die die Entschlüsselungsmethode zurückgibt.

Um die Daten zu entschlüsseln, rufen Sie die `aws_cryptosdk_session_process_full`-Methode auf.

```
/* Decrypting data */
aws_cryptosdk_session_process_full(session,
                                   plaintext,
                                   plaintext_buffer_size,
                                   &plaintext_length,
                                   ciphertext,
                                   ciphertext_length)
```

Referenzzählung

Um Lecks im Arbeitsspeicher zu verhindern, müssen Sie Ihre Referenzen an alle Objekte freigeben, die Sie erstellen, wenn Sie mit diesen fertig sind. Andernfalls erhalten Sie Lecks im Arbeitsspeicher. Das SDK bietet Methoden, um diese Aufgabe zu vereinfachen.

Wenn Sie ein übergeordnetes Objekt mit einem der folgenden untergeordneten Objekte erstellen, erhält und behält das übergeordnete Objekt einen Verweis auf das untergeordnete Objekt wie folgt bei:

- Ein [Schlüsselbund](#), z. B. das Erstellen einer Sitzung mit einem Schlüsselbund
- Eine Standard-[Manager kryptographischer Materialien](#)(CMM), z. B. das Erstellen einer Sitzung oder eines benutzerdefinierten CMM mit einem Standard-CMM
- Ein [Datenschlüssel-Cache](#), z. B. das Erstellen eines Caching-CMM mit einem Schlüsselbund und Cache

Wenn Sie keinen unabhängigen Verweis auf das untergeordnete Objekt benötigen, können Sie den Verweis auf das untergeordnete Objekt freigeben, sobald Sie das übergeordnete Objekt erstellen.

Der verbleibende Verweis auf das untergeordnete Objekt wird freigegeben, wenn das übergeordnete Objekt zerstört wird. Durch dieses Muster wird sichergestellt, dass Sie die Referenz auf jedes Objekt nur so lange wie nötig behalten. Es kommt auch nicht zu einem Leck im Arbeitsspeicher aufgrund von nicht freigegebenen Referenzen.

Sie sind nur dafür verantwortlich, Verweise auf die untergeordneten Objekte freizugeben, die Sie explizit erstellen. Sie sind nicht verantwortlich für die Verwaltung von Verweisen auf Objekte, die das SDK für Sie erstellt. Wenn das SDK ein Objekt erstellt, z. B. das Standard-CMM, das `aws_cryptosdk_caching_cmm_new_from_keyring`-Methode fügt zu einer Sitzung hinzu, das SDK verwaltet die Erstellung und Zerstörung des Objekts und seiner Referenzen.

Wenn Sie im folgenden Beispiel eine Sitzung mit einem [Schlüsselbund](#) erstellen, erhält die Sitzung einen Verweis auf den Schlüsselbund und behält diesen Verweis bei, bis die Sitzung zerstört wird. Wenn Sie keinen zusätzlichen Verweis auf den Schlüsselbund beibehalten müssen, können Sie die `aws_cryptosdk_keyring_release`-Methode verwenden, um das Schlüsselbundobjekt freizugeben, sobald die Sitzung erstellt wird. Diese Methode verringert die Referenzanzahl für den Schlüsselbund. Der Verweis der Sitzung auf den Schlüsselbund wird freigegeben, wenn Sie `aws_cryptosdk_session_destroy` aufrufen, um die Sitzung zu zerstören.

```
// The session gets a reference to the keyring.
struct aws_cryptosdk_session *session =
    aws_cryptosdk_session_new_from_keyring_2(alloc, AWS_CRYPTOSDK_ENCRYPT, keyring);

// After you create a session with a keyring, release the reference to the keyring
// object.
aws_cryptosdk_keyring_release(keyring);
```

Für komplexere Aufgaben, z. B. die Wiederverwendung eines Schlüsselbunds für mehrere Sitzungen oder das Angeben eines Algorithmen-Pakets in einem CMM, müssen Sie möglicherweise einen unabhängigen Verweis auf das Objekt beibehalten. Wenn ja, rufen Sie die Freigabemethoden nicht sofort auf. Geben Sie stattdessen Ihre Referenzen frei, wenn Sie die Objekte nicht mehr verwenden, zusätzlich zum Löschen der Sitzung.

Diese Referenzzähltechnik funktioniert auch, wenn Sie alternative CMMs verwenden, z. B. den Caching-[Datenschlüssel-Caching](#)aus. Wenn Sie einen Caching- CMM aus einem Cache und einem Schlüsselbund erstellen, erhält der Caching- einen Verweis auf beide Objekte. Wenn Sie sie nicht für eine andere Aufgabe benötigen, können Sie Ihre unabhängigen Referenzen auf den Cache und Schlüsselbund freigeben, sobald der Caching- erstellt wird. Wenn Sie dann eine Sitzung mit dem Caching- CMM erstellen, können Sie Ihren Verweis auf den Caching- CMM freigeben.

Beachten Sie, dass Sie nur für die Freigabe von Verweisen auf Objekte verantwortlich sind, die Sie explizit erstellen. Objekte, die von den Methoden für Sie erstellt werden, z. B. das Standard-CMM, das dem Caching- CMM zugrunde liegt, werden von der Methode verwaltet.

```
/ Create the caching CMM from a cache and a keyring.
struct aws_cryptosdk_cmm *caching_cmm =
    aws_cryptosdk_caching_cmm_new_from_keyring(allocator, cache, kms_keyring, NULL, 60,
    AWS_TIMESTAMP_SECS);

// Release your references to the cache and the keyring.
aws_cryptosdk_materials_cache_release(cache);
aws_cryptosdk_keyring_release(kms_keyring);

// Create a session with the caching CMM.
struct aws_cryptosdk_session *session = aws_cryptosdk_session_new_from_cmm_2(allocator,
    AWS_CRYPTOSDK_ENCRYPT, caching_cmm);

// Release your references to the caching CMM.
aws_cryptosdk_cmm_release(caching_cmm);

// ...

aws_cryptosdk_session_destroy(session);
```

AWS-Verschlüsselungs-SDK for C Beispiele für

Die folgenden Beispiele zeigen Ihnen, wie Sie mit dem AWS-Verschlüsselungs-SDK for C Daten verschlüsseln und entschlüsseln.

Die Beispiele in diesem Abschnitt zeigen, wie Sie Versionen 2.0 verwenden und später von der AWS-Verschlüsselungs-SDK for C wechseln. Für Beispiele, die frühere Versionen verwenden, finden Sie Ihre Version in der [Versionen](#)-Liste der [aws-encryption-sdk-c-Repository](#)-Repository auf GitHub aus.

Wenn Sie das AWS-Verschlüsselungs-SDK for C installieren und erstellen, ist der Quellcode für dieses und andere Beispiele im `examples`-Unterverzeichnis enthalten. Sie werden kompiliert und im `build`-Verzeichnis erstellt. Sie finden sie auch im [Beispiele](#)-Unterverzeichnis des [aws-encryption-sdk-c-Repository](#) auf GitHub aus.

Themen

- [Verschlüsseln und Entschlüsseln von Zeichenfolgen](#)

Verschlüsseln und Entschlüsseln von Zeichenfolgen

Das folgende Beispiel zeigt Ihnen, wie Sie mit dem AWS-Verschlüsselungs-SDK for C eine Zeichenfolge verschlüsseln und entschlüsseln.

In diesem Beispiel wird die [AWS KMSSchlüsselring](#), eine Art Schlüsselbund, der ein AWS KMS key in [AWS Key Management Service \(AWS KMS\)](#) um Datenschlüssel zu generieren und zu verschlüsseln. Das Beispiel enthält Code, der in C++ geschrieben wurde. Die AWS-Verschlüsselungs-SDK for C erfordert das AWS SDK for C++. Rufen Sie [AWS KMS](#) bei Verwendung von [AWS KMSSchlüsselring](#). Wenn Sie einen Schlüsselbund verwenden, der nicht mit [interagiert AWS KMS](#) wie ein roher AES-Schlüsselanhänger, ein roher RSA-Schlüsselring oder ein Multi-Schlüsselring, der keinen [AWS KMSSchlüsselring](#), der [AWS SDK for C++](#) ist nicht erforderlich.

Hilfe beim Erstellen eines [AWS KMS key](#) finden Sie unter [Erstellen von Schlüsseln](#) im [AWS Key Management Service Entwicklerhandbuch](#). Hilfe bei der Identifizierung des [AWS KMS key](#) in einem [AWS KMSSchlüsselring](#), siehe [Identifizierung AWS KMS keys in einem AWS KMS Schlüsselring](#) aus.

Das vollständige Codebeispiel finden Sie unter: [string.cpp](#)

Themen

- [Verschlüsseln einer Zeichenfolge](#)
- [Entschlüsseln einer Zeichenfolge](#)

Verschlüsseln einer Zeichenfolge

Im ersten Teil dieses Beispiels wird ein [AWS KMSSchlüsselring](#) mit einem [AWS KMS key](#) um eine Klartext-Zeichenfolge zu verschlüsseln.

Schritt 1. Laden Sie Fehlerzeichenfolgen

Rufen Sie die `aws_cryptosdk_load_error_strings()`-Methode in Ihrem C- oder C++-Code. Es lädt Fehlerinformationen, die für das Debuggen sehr nützlich sind.

Sie müssen es nur einmal aufrufen, z. B. in Ihrer `main`-Methode

```
/* Load error strings for debugging */
aws_cryptosdk_load_error_strings();
```

Schritt 2: Erstellen Sie den Schlüsselbund.

Erstellen eines AWS KMS Schlüsselbund für die Verschlüsselung. Der Schlüsselbund in diesem Beispiel ist mit einem konfiguriert AWS KMS key, aber Sie können ein AWS KMS Schlüsselbund mit mehreren AWS KMS keys, einschließlich AWS KMS keys in verschiedenen AWS-Regionen und verschiedene Konten.

So identifizieren Sie ein AWS KMS key in einem Verschlüsselungsschlüsselbund im AWS-Verschlüsselungs-SDK for C, geben Sie ein [Schlüssel-ARN](#) oder [Alias-ARN](#) aus. In einem Entschlüsselungsschlüsselbund müssen Sie einen Schlüssel-ARN verwenden. Details hierzu finden Sie unter [Identifizierung AWS KMS keys in einem AWS KMS Schlüsselbund](#).

[Identifizierung AWS KMS keys in einem AWS KMS Schlüsselbund](#)

Wenn Sie einen Schlüsselbund mit mehreren erstellen AWS KMS keys geben Sie die AWS KMS key Wird verwendet, um den Klartext-Datenschlüssel zu generieren und zu verschlüsseln, sowie ein optionales Array von zusätzlichen AWS KMS keys Die den gleichen Klartext-Datenschlüssel verschlüsseln. In diesem Fall geben Sie nur den Generator an AWS KMS key aus.

Ersetzen Sie vor Ausführung dieses Codes den ARN des Beispiel-Schlüssels durch einen gültigen.

```
const char * key_arn = "arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";  
  
struct aws_cryptosdk_keyring *kms_keyring =  
    Aws::Cryptosdk::KmsKeyring::Builder().Build(key_arn);
```

Schritt 3: Erstellen Sie eine Sitzung.

Erstellen Sie eine Sitzung mithilfe des Allocators, eines Modus-Enumerators und des Schlüsselbunds.

Jede Sitzung erfordert eine Modus: entweder `AWS_CRYPTOSDK_ENCRYPT` zum Verschlüsseln oder `AWS_CRYPTOSDK_DECRYPT` zum Entschlüsseln. Um den Modus einer vorhandenen Sitzung zu ändern, verwenden Sie die `aws_cryptosdk_session_reset`-Methode.

Nach dem Erstellen einer Sitzung mit dem Schlüsselbund können Sie Ihre Referenz auf den Schlüsselbund unter Verwendung der Methode, die das SDK bietet, freigeben. Die Sitzung behält während ihrer Lebensdauer einen Verweis auf das Schlüsselbundobjekt bei. Verweise auf den

Schlüsselbund und die Sitzungsobjekte werden freigegeben, wenn Sie die Sitzung zerstören. Diese [Referenzzähltechnik](#) hilft, Lecks im Arbeitsspeicher zu verhindern, und sorgt dafür, dass die Objekte nicht freigegeben werden, während sie verwendet werden.

```
struct aws_cryptosdk_session *session =
    aws_cryptosdk_session_new_from_keyring_2(alloc, AWS_CRYPTOSDK_ENCRYPT,
    kms_keyring);

/* When you add the keyring to the session, release the keyring object */
aws_cryptosdk_keyring_release(kms_keyring);
```

Schritt 4: Legen Sie den Verschlüsselungskontext fest.

Ein [Verschlüsselungskontext](#) ist eine Art zufälliger, nicht geheimer, zusätzlich authentifizierter Daten. Wenn Sie beim Verschlüsseln einen Verschlüsselungskontext angeben, bindet das AWS Encryption SDK den Verschlüsselungskontext kryptografisch an den Verschlüsselungstext. Daher ist zum Entschlüsseln der Daten derselbe Verschlüsselungskontext erforderlich. Die Verwendung eines Verschlüsselungskontexts ist optional, aber wir empfehlen dies als eine bewährte Methode.

Erstellen Sie zuerst eine Hash-Tabelle, die die Zeichenfolgen des Verschlüsselungskontexts enthält.

```
/* Allocate a hash table for the encryption context */
int set_up_enc_ctx(struct aws_allocator *alloc, struct aws_hash_table *my_enc_ctx)

// Create encryption context strings
AWS_STATIC_STRING_FROM_LITERAL(enc_ctx_key1, "Example");
AWS_STATIC_STRING_FROM_LITERAL(enc_ctx_value1, "String");
AWS_STATIC_STRING_FROM_LITERAL(enc_ctx_key2, "Company");
AWS_STATIC_STRING_FROM_LITERAL(enc_ctx_value2, "MyCryptoCorp");

// Put the key-value pairs in the hash table
aws_hash_table_put(my_enc_ctx, enc_ctx_key1, (void *)enc_ctx_value1, &was_created)
aws_hash_table_put(my_enc_ctx, enc_ctx_key2, (void *)enc_ctx_value2, &was_created)
```

Rufen Sie einen veränderlichen Zeiger auf den Verschlüsselungskontext in der Sitzung ab. Verwenden Sie anschließend die `aws_cryptosdk_enc_ctx_clone`-Funktion zum Kopieren des Verschlüsselungskontexts in die Sitzung. Halten Sie die Kopie in `my_enc_ctx`, damit Sie den Wert nach der Entschlüsselung der Daten validieren können.

Der Verschlüsselungskontext ist Teil der Sitzung, nicht ein Parameter, der an die Sitzungs-Verarbeitungsfunktion übergeben wird. Dadurch wird sichergestellt, dass derselbe Verschlüsselungskontext für jedes Segment einer Nachricht verwendet wird, auch wenn die Sitzungs-Verarbeitungsfunktion mehrmals aufgerufen wird, um die gesamte Nachricht zu verschlüsseln.

```
struct aws_hash_table *session_enc_ctx =
    aws_cryptosdk_session_get_enc_ctx_ptr_mut(session);

aws_cryptosdk_enc_ctx_clone(alloc, session_enc_ctx, my_enc_ctx)
```

Schritt 5: Verschlüsseln Sie die Zeichenfolge.

Verwenden Sie zum Verschlüsseln der Klartext-Zeichenfolge die `aws_cryptosdk_session_process_full`-Methode, wobei sich die Sitzung im Verschlüsselungsmodus befinden muss. Diese Methode wurde eingeführt in AWS Encryption SDK-Versionen 1.9x und 2.2x, ist für die Nicht-Streaming-Verschlüsselung und Entschlüsselung konzipiert. Um mit Streaming-Daten umzugehen, rufen Sie die `aws_cryptosdk_session_process` in einer Schleife.

Beim Verschlüsseln sind die Klartextfelder Eingabefelder. Die Verschlüsselungstext-Felder sind Ausgabefelder. Wenn die Verarbeitung abgeschlossen ist, enthält das `ciphertext_output`-Feld die [verschlüsselte Nachricht](#), einschließlich des tatsächlichen Verschlüsselungstexts, der verschlüsselten Datenschlüssel und des Verschlüsselungskontexts. Sie können diese verschlüsselte Nachricht durch Verwenden des AWS Encryption SDK für jede unterstützte Programmiersprache entschlüsseln.

```
/* Gets the length of the plaintext that the session processed */
size_t ciphertext_len_output;
if (AWS_OP_SUCCESS != aws_cryptosdk_session_process_full(session,
                                                         ciphertext_output,
                                                         ciphertext_buf_sz_output,
                                                         &ciphertext_len_output,
                                                         plaintext_input,
                                                         plaintext_len_input)) {
    aws_cryptosdk_session_destroy(session);
    return 8;
}
```

Schritt 6: Bereinigen Sie die Sitzung.

Im letzten Schritt wird die Sitzung einschließlich Referenzen auf das CMM und den Schlüsselbund zerstört.

Wenn Sie die Sitzung nicht zerstören möchten, können Sie sie mit demselben Schlüsselbund und dem CMM zum Entschlüsseln der Zeichenfolge oder zum Verschlüsseln oder Entschlüsseln von anderen Nachrichten verwenden. Um die Sitzung zum Entschlüsseln zu verwenden, wenden Sie die `aws_cryptosdk_session_reset`-Methode an, um den Modus in `AWS_CRYPTOSDK_DECRYPT` zu ändern.

Entschlüsseln einer Zeichenfolge

Im zweiten Teil dieses Beispiels wird eine verschlüsselte Nachricht entschlüsselt, die den Verschlüsselungstext der ursprünglichen Zeichenfolge enthält.

Schritt 1: Laden Sie Fehlerzeichenfolgen

Rufen Sie die `aws_cryptosdk_load_error_strings()`-Methode in Ihrem C- oder C++-Code. Es lädt Fehlerinformationen, die für das Debuggen sehr nützlich sind.

Sie müssen es nur einmal aufrufen, z. B. in Ihrer `main`-Methode

```
/* Load error strings for debugging */  
aws_cryptosdk_load_error_strings();
```

Schritt 2: Erstellen Sie den Schlüsselbund.

Wenn Sie Daten in AWS KMS entschlüsseln, übergeben Sie die [verschlüsselte Nachricht](#), die die API zurückgegeben hat. Die [API entschlüsseln](#) nimmt kein AWS KMS key als Eingabe. Stattdessen verwendet AWS KMS denselben AWS KMS key zum Entschlüsseln des Verschlüsselungstexts, der auch für die Verschlüsselung verwendet wurde. Allerdings ist der AWS Encryption SDK können Sie eine angeben AWS KMS Schlüsselbund mit AWS KMS keys beim Verschlüsseln und Entschlüsseln von.

Beim Entschlüsseln können Sie einen Schlüsselbund konfigurieren, der nur die AWS KMS keys enthält, die Sie zum Entschlüsseln der verschlüsselten Nachricht verwenden möchten. Beispielsweise möchten Sie einen Schlüsselbund mit nur dem AWS KMS key erstellen, der von einer bestimmten Rolle in Ihrer Organisation verwendet wird. Die AWS Encryption SDK wird niemals ein benutzen AWS KMS key. Es sei denn, er erscheint im Entschlüsselungsschlüsselbund.

Wenn das SDK die verschlüsselten Datenschlüssel nicht mithilfe der AWS KMS keys im von Ihnen bereitgestellten Schlüsselbund entschlüsseln kann, weil entweder keiner der AWS KMS keys im Schlüsselbund zum Verschlüsseln der Datenschlüssel verwendet wurde oder weil der Aufrufer die AWS KMS keys im Schlüsselbund nicht für die Entschlüsselung verwenden darf, schlägt die Entschlüsselung fehl.

Wenn Sie ein AWS KMS key für einen Entschlüsselungsschlüsselbund müssen Sie dessen [Schlüssel-ARN](#) aus [Alias-ARNs](#) sind nur in Verschlüsselungsschlüsselbunden zulässig. Hilfe bei der Identifizierung des AWS KMS keys in einem AWS KMS Schlüsselbund, siehe [Identifizierung AWS KMS keys in einem AWS KMS Schlüsselbund](#) aus.

In diesem Beispiel geben wir einen Schlüsselbund an, der mit demselben konfiguriert wurde AWS KMS key wird verwendet, um die Zeichenfolge zu verschlüsseln. Ersetzen Sie vor Ausführung dieses Codes den ARN des Beispiel-Schlüssels durch einen gültigen.

```
const char * key_arn = "arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"  
  
struct aws_cryptosdk_keyring *kms_keyring =  
    Aws::Cryptosdk::KmsKeyring::Builder().Build(key_arn);
```

Schritt 3: Erstellen Sie eine Sitzung.

Erstellen Sie eine Sitzung unter Verwendung des Allocators und des Schlüsselbunds. Zum Konfigurieren der Sitzung für die Entschlüsselung konfigurieren Sie die Sitzung mit dem `AWS_CRYPTOSDK_DECRYPT`-Modus.

Nach dem Erstellen einer Sitzung mit einem Schlüsselbund können Sie Ihre Referenz auf den Schlüsselbund unter Verwendung der Methode, die das SDK bietet, freigeben. Die Sitzung behält während ihrer Lebensdauer einen Verweis auf das Schlüsselbundobjekt bei und sowohl die Sitzung als auch der Schlüsselbund werden freigegeben, wenn Sie die Sitzung zerstören. Diese Referenzzähltechnik hilft, Lecks im Arbeitsspeicher zu verhindern, und sorgt dafür, dass die Objekte nicht freigegeben werden, während sie verwendet werden.

```
struct aws_cryptosdk_session *session =  
    aws_cryptosdk_session_new_from_keyring_2(alloc, AWS_CRYPTOSDK_DECRYPT,  
    kms_keyring);  
  
/* When you add the keyring to the session, release the keyring object */  
aws_cryptosdk_keyring_release(kms_keyring);
```

Schritt 4: Entschlüsseln Sie die Zeichenfolge.

Verwenden Sie zum Entschlüsseln der Zeichenfolge die `aws_cryptosdk_session_process_full`-Methode mit der Sitzung, die für die Entschlüsselung konfiguriert ist. Diese Methode wurde eingeführt in AWS Encryption SDK-Versionen 1.9x und 2.2x, ist für die Nicht-Streaming-Verschlüsselung und Entschlüsselung konzipiert. Um mit Streaming-Daten umzugehen, rufen Sie die `aws_cryptosdk_session_process` in einer Schleife.

Beim Entschlüsseln sind die Verschlüsselungstext-Felder Eingabefelder. Die Klartextfelder sind Ausgabefelder. Das `ciphertext_input`-Feld enthält die [verschlüsselte Nachricht](#), die die Verschlüsselungsmethode zurückgegeben hat. Wenn die Verarbeitung abgeschlossen ist, enthält das `plaintext_output`-Feld die Klartext-Zeichenfolge (entschlüsselt).

```
size_t plaintext_len_output;

if (AWS_OP_SUCCESS != aws_cryptosdk_session_process_full(session,
                                                         plaintext_output,
                                                         plaintext_buf_sz_output,
                                                         &plaintext_len_output,
                                                         ciphertext_input,
                                                         ciphertext_len_input)) {
    aws_cryptosdk_session_destroy(session);
    return 13;
}
```

Schritt 5: Überprüfen Sie den Verschlüsselungskontext.

Stellen Sie sicher, dass der eigentliche Verschlüsselungskontext - der zum Entschlüsseln der Nachricht verwendet wurde - den Verschlüsselungskontext enthält, den Sie beim Verschlüsseln der Nachricht angegeben haben. Der tatsächliche Verschlüsselungskontext kann zusätzliche Paare enthalten, da der [Manager von kryptographischen Materialien](#) (CMM) Paare zum angegebenen Verschlüsselungskontext hinzufügen kann, bevor die Nachricht verschlüsselt wird.

Im AWS-Verschlüsselungs-SDK for C müssen Sie beim Entschlüsseln keinen Verschlüsselungskontext angeben, da der Verschlüsselungskontext in der verschlüsselten Nachricht enthalten ist, die das SDK zurückgibt. Aber bevor sie die Klartext-Nachricht zurückgibt, sollte Ihre Entschlüsselungsfunktion überprüfen, dass alle Paare im angegebenen Verschlüsselungskontext in dem Verschlüsselungskontext erscheinen, der zum Entschlüsseln der Nachricht verwendet wurde.

Rufen Sie zuerst einen schreibgeschützten Zeiger auf die Hash-Tabelle in der Sitzung ab. Diese Hash-Tabelle enthält den Verschlüsselungskontext, der zum Entschlüsseln der Nachricht verwendet wurde.

```
const struct aws_hash_table *session_enc_ctx =
    aws_cryptosdk_session_get_enc_ctx_ptr(session);
```

Durchlaufen Sie anschließend den Verschlüsselungskontext in der `my_enc_ctx`-Hash-Tabelle, die Sie beim Verschlüsseln kopiert haben. Überprüfen Sie, dass alle Paare in der `my_enc_ctx`-Hash-Tabelle, die zum Verschlüsseln verwendet wurde, in der `session_enc_ctx`-Hash-Tabelle erscheinen, die zur Entschlüsselung verwendet wurde. Wenn ein Schlüssel nicht vorhanden ist oder dieser Schlüssel einen anderen Wert hat, beenden Sie die Verarbeitung und schreiben Sie eine Fehlermeldung.

```
for (struct aws_hash_iter iter = aws_hash_iter_begin(my_enc_ctx); !
aws_hash_iter_done(&iter);
    aws_hash_iter_next(&iter)) {
    struct aws_hash_element *session_enc_ctx_kv_pair;
    aws_hash_table_find(session_enc_ctx, iter.element.key,
&session_enc_ctx_kv_pair)

    if (!session_enc_ctx_kv_pair ||
        !aws_string_eq(
            (struct aws_string *)iter.element.value, (struct aws_string
*)session_enc_ctx_kv_pair->value)) {
        fprintf(stderr, "Wrong encryption context!\n");
        abort();
    }
}
```

Schritt 6: Bereinigen Sie die Sitzung.

Nachdem Sie den Verschlüsselungskontext überprüft haben, können Sie die Sitzung zerstören oder wiederverwenden. Wenn Sie die Sitzung neu konfigurieren müssen, verwenden Sie die `aws_cryptosdk_session_reset`-Methode

```
aws_cryptosdk_session_destroy(session);
```

AWS Encryption SDK für .NET

Die AWS Encryption SDK for .NET ist eine clientseitige Verschlüsselungsbibliothek für Entwickler, die Anwendungen in C# und anderen .NET-Programmiersprachen schreiben. Sie wird unter Windows, macOS und Linux unterstützt.

Alle [Programmiersprachenimplementierungen](#) von sind vollständig interoperabel. AWS Encryption SDK Wenn Sie jedoch Daten mit dem [erforderlichen Verschlüsselungskontext CMM](#) in Version 4 verschlüsseln. x von AWS Encryption SDK für .NET, Sie können es nur mit Version 4 entschlüsseln. x der AWS Encryption SDK für .NET oder Version 3. x der AWS-Verschlüsselungs-SDK for Java.

Note

Version 4.0.0 von AWS Encryption SDK für .NET weicht von der AWS Encryption SDK Nachrichtenspezifikation ab. Aus diesem Grund können Nachrichten, die mit Version 4.0.0 verschlüsselt wurden, nur mit Version 4.0.0 oder höher von for.NET entschlüsselt werden. AWS Encryption SDK Sie können mit keiner anderen Programmiersprachenimplementierung entschlüsselt werden.

Version 4.0.1 von AWS Encryption SDK for .NET schreibt Nachrichten gemäß der AWS Encryption SDK Nachrichtenspezifikation und ist mit anderen Programmiersprachenimplementierungen interoperabel. Standardmäßig kann Version 4.0.1 Nachrichten lesen, die mit Version 4.0.0 verschlüsselt wurden. Wenn Sie jedoch mit Version 4.0.0 verschlüsselte Nachrichten nicht entschlüsseln möchten, können Sie die [NetV4_0_0_RetryPolicy](#)Eigenschaft so angeben, dass der Client diese Nachrichten nicht lesen kann. Weitere Informationen finden Sie in den [Versionshinweisen zu Version 4.0.1 im Repository](#) unter. [aws-encryption-sdk-dafny](#) GitHub

Das AWS Encryption SDK für.NET unterscheidet sich von einigen anderen Programmiersprachenimplementierungen AWS Encryption SDK in folgenden Punkten:

- Keine Unterstützung für das Zwischenspeichern [von Datenschlüsseln](#)

Note

Version 4. x of the AWS Encryption SDK for .NET unterstützt den [AWS KMS hierarchischen Schlüsselbund](#), eine alternative Lösung zum Zwischenspeichern kryptografischer Materialien.

- Keine Unterstützung für Streaming-Daten
- [Keine Protokollierung oder Stack-Traces](#) von AWS Encryption SDK für .NET
- [Benötigt den AWS SDK for .NET](#)

Das AWS Encryption SDK für .NET enthält alle Sicherheitsfunktionen, die in Version 2.0 eingeführt wurden. x und höher von anderen Sprachimplementierungen von AWS Encryption SDK. Wenn Sie jedoch for.NET verwenden, AWS Encryption SDK um Daten zu entschlüsseln, die mit einer Version vor 2.0 verschlüsselt wurden. x-Version einer anderen Sprachimplementierung von AWS Encryption SDK, möglicherweise müssen Sie Ihre [Verpflichtungsrichtlinie](#) anpassen. Details hierzu finden Sie unter [Wie legen Sie Ihre Verpflichtungsrichtlinie fest](#).

Die AWS Encryption SDK für .NET-Version ist ein Produkt von AWS Encryption SDK In [Dafny](#), einer formalen Überprüfungssprache, in der Sie Spezifikationen, den Code zu ihrer Implementierung und die Beweise, um sie zu testen, schreiben. Das Ergebnis ist eine Bibliothek, die die Funktionen von AWS Encryption SDK in einem Framework implementiert, das die funktionale Korrektheit gewährleistet.

Weitere Informationen

- Beispiele für die Konfiguration von Optionen in der AWS Encryption SDK, z. B. die Angabe einer alternativen Algorithmussuite, die Beschränkung verschlüsselter Datenschlüssel und die Verwendung von Schlüsseln für AWS KMS mehrere Regionen, finden Sie unter [Konfiguration der AWS Encryption SDK](#)
- Einzelheiten zur Programmierung mit dem AWS Encryption SDK für .NET finden Sie im [aws-encryption-sdk-net](#) Verzeichnis des aws-encryption-sdk-dafny Repositorys unter GitHub.

Themen

- [Installation von AWS Encryption SDK für .NET](#)
- [Debuggen des AWS Encryption SDK für .NET](#)

- [AWS KMSSchlüsselanhänger im AWS Encryption SDK für.NET](#)
- [Erforderliche Verschlüsselungskontexte in Version 4.x](#)
- [AWS Encryption SDK für .NET-Beispiele](#)

Installation von AWS Encryption SDK für.NET

Das AWS Encryption SDK für .NET ist als [AWS.Cryptography.EncryptionSDK](#) Paket in verfügbar NuGet. Einzelheiten zur Installation und Erstellung von AWS Encryption SDK für.NET finden Sie in der Datei [README.md](#) im Repository. `aws-encryption-sdk-net`

Version 3.x

Ausführung 3. x of the AWS Encryption SDK für .NET unterstützt .NET Framework 4.5.2 — 4.8 nur unter Windows. Es unterstützt .NET Core 3.0+ und .NET 5.0 und höher auf allen unterstützten Betriebssystemen.

Version 4.x

Ausführung 4. x of the AWS Encryption SDK für .NET unterstützt .NET 6.0 und .NET Framework net48 und höher.

AWS Encryption SDK für .NET sind die Schlüssel AWS SDK for .NET auch dann erforderlich, wenn Sie nicht AWS Key Management Service (AWS KMS) verwenden. Es ist mit dem NuGet Paket installiert. Sofern Sie jedoch keine AWS KMS Schlüssel verwenden, benötigt AWS Encryption SDK für.NET keine AWS Anmeldeinformationen und keine Interaktion mit einem AWS Dienst. AWS-Konto Hilfe beim Einrichten eines AWS Kontos finden Sie bei Bedarf unter [Verwenden der AWS Encryption SDK mit AWS KMS](#).

Debuggen des AWS Encryption SDK für .NET

Das AWS Encryption SDK für .NET generiert keine Protokolle. Ausnahmen im AWS Encryption SDK für.NET erzeugen eine Ausnahmemeldung, aber keine Stack-Traces.

Um Ihnen beim Debuggen zu helfen, stellen Sie sicher, dass Sie die AWS SDK for .NET Anmeldung bei aktivieren. Mithilfe der Protokolle und Fehlermeldungen von AWS SDK for .NET können Sie Fehler, die in der auftreten, AWS SDK for .NET von denen in AWS Encryption SDK für.NET unterscheiden. Hilfe zur AWS SDK for .NET Protokollierung finden Sie [AWSLogging](#) im AWS SDK for .NET Entwicklerhandbuch. (Um das Thema zu lesen, erweitern Sie den Abschnitt Öffnen, um .NET Framework-Inhalte anzuzeigen.)

AWS KMSSchlüsselanhänger im AWS Encryption SDK für.NET

Die grundlegenden AWS KMS Schlüsselbunde im AWS Encryption SDK für.NET benötigen nur einen KMS-Schlüssel. Sie benötigen außerdem einen AWS KMS Client, sodass Sie den Client für den AWS-Region KMS-Schlüssel konfigurieren können.

Um einen AWS KMS Schlüsselbund mit einem oder mehreren Schlüsseln zu erstellen, verwenden Sie einen Mehrfachschlüsselbund. Der AWS Encryption SDK für.NET hat einen speziellen Mehrfachschlüsselbund, der einen oder mehrere AWS KMS Schlüssel akzeptiert, und einen Standard-Mehrfachschlüsselbund, der einen oder mehrere Schlüsselbunde eines beliebigen unterstützten Typs akzeptiert. Manche Programmierer ziehen es vor, eine Methode mit mehreren Schlüsselbunden zu verwenden, um all ihre Schlüsselbunde zu erstellen, und for.NET unterstützt diese Strategie. AWS Encryption SDK

[Die AWS Encryption SDK für.NET bietet grundlegende Einzelschlüsselringe und Mehrfachschlüsselringe für alle typischen Anwendungsfälle, einschließlich Schlüssel für mehrere Regionen. AWS KMS](#)

Um beispielsweise einen AWS KMS Schlüsselbund mit einem Schlüssel zu erstellen, können Sie die Methode verwenden. `AWS KMS CreateAwsKmsKeyring()`

Version 3.x

Im folgenden Beispiel wird Version 3 verwendet. x von AWS Encryption SDK für .NET, um einen AWS KMS Standardclient für die Region zu erstellen, der den angegebenen Schlüssel enthält.

```
// Instantiate the AWS Encryption SDK and material providers
var encryptionSdk = AwsEncryptionSdkFactory.CreateDefaultAwsEncryptionSdk();
var materialProviders =

    AwsCryptographicMaterialProvidersFactory.CreateDefaultAwsCryptographicMaterialProviders();

string keyArn = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

// Instantiate the keyring input object
var kmsKeyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = keyArn
};
```

```
// Create the keyring
var keyring = materialProviders.CreateAwsKmsKeyring(kmsKeyringInput);
```

Version 4.x

Das folgende Beispiel verwendet Version 4. x von AWS Encryption SDK für .NET, um einen AWS KMS Client für die Region zu erstellen, der den angegebenen Schlüssel enthält.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk = new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

string keyArn = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

// Instantiate the keyring input object
var createKeyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = kmsArn
};

// Create the keyring
var kmsKeyring = mpl.CreateAwsKmsKeyring(createKeyringInput);
```

Verwenden Sie die `CreateAwsKmsMultiKeyring()` Methode, um einen Schlüsselbund mit einem oder mehreren AWS KMS Schlüsseln zu erstellen. In diesem Beispiel werden zwei AWS KMS Schlüssel verwendet. Verwenden Sie nur den `Generator` Parameter, um einen KMS-Schlüssel anzugeben. Der `KmsKeyIds` Parameter, der zusätzliche KMS-Schlüssel angibt, ist optional.

Die Eingabe für diesen Schlüsselbund benötigt keinen AWS KMS Client. Stattdessen AWS Encryption SDK verwendet der den AWS KMS Standardclient für jede Region, die durch einen KMS-Schlüssel im Schlüsselbund repräsentiert wird. Befindet sich der KMS-Schlüssel, der durch den Wert des `Generator` Parameters identifiziert wird, beispielsweise in der Region USA West (Oregon) (`us-west-2`), AWS Encryption SDK wird ein AWS KMS Standardclient für die `us-west-2` Region erstellt. Wenn Sie den AWS KMS Client anpassen müssen, verwenden Sie die `CreateAwsKmsKeyring()` Methode.

Im folgenden Beispiel wird Version 4 verwendet. x von AWS Encryption SDK für .NET und die `CreateAwsKmsKeyring()` Methode zum Anpassen des AWS KMS Clients.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk = new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

string generatorKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
List<string> additionalKeys = new List<string> { "arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321" };

// Instantiate the keyring input object
var createEncryptKeyringInput = new CreateAwsKmsMultiKeyringInput
{
    Generator = generatorKey,
    KmsKeyIds = additionalKeys
};

var kmsEncryptKeyring =
    materialProviders.CreateAwsKmsMultiKeyring(createEncryptKeyringInput);
```

Version 4. x of the AWS Encryption SDK für .NET unterstützt AWS KMS Schlüsselringe, die symmetrische Verschlüsselung (SYMMETRIC_DEFAULT) oder asymmetrische RSA-KMS-Schlüssel verwenden. AWS KMS Schlüsselringe, die mit asymmetrischen RSA-KMS-Schlüsseln erstellt wurden, können nur ein key pair enthalten.

Um mit einem asymmetrischen AWS KMS RSA-Schlüsselbund zu verschlüsseln, benötigen Sie [kms:GenerateDataKey](#) oder [kms:Encrypt](#) nicht, da Sie bei der Erstellung des Schlüsselbunds das Material der öffentlichen Schlüssel angeben müssen, das Sie für die Verschlüsselung verwenden möchten. Bei der Verschlüsselung mit diesem Schlüsselbund werden keine Anrufe getätigt. AWS KMS [Um mit einem asymmetrischen AWS KMS RSA-Schlüsselbund zu entschlüsseln, benötigen Sie die kms:Decrypt-Berechtigung.](#)

Um einen asymmetrischen AWS KMS RSA-Schlüsselbund zu erstellen, müssen Sie den öffentlichen Schlüssel und den privaten Schlüssel ARN aus Ihrem asymmetrischen RSA-KMS-Schlüssel angeben. Der öffentliche Schlüssel muss PEM-codiert sein. Im folgenden Beispiel wird ein AWS KMS Schlüsselbund mit einem asymmetrischen RSA-Schlüsselpaar erstellt.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk = new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());
```

```
var publicKey = new MemoryStream(Encoding.UTF8.GetBytes(AWS KMS RSA public key));

// Instantiate the keyring input object
var createKeyringInput = new CreateAwsKmsRsaKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = AWS KMS RSA private key ARN,
    PublicKey = publicKey,
    EncryptionAlgorithm = EncryptionAlgorithmSpec.RSAES_OAEP_SHA_256
};

// Create the keyring
var kmsRsaKeyring = mpl.CreateAwsKmsRsaKeyring(createKeyringInput);
```

Erforderliche Verschlüsselungskontexte in Version 4.x

Mit Version 4. x of the AWS Encryption SDK für .NET können Sie den erforderlichen Verschlüsselungskontext CMM verwenden, um [Verschlüsselungskontexte](#) für Ihre kryptografischen Operationen vorzuschreiben. Ein Verschlüsselungskontext ist ein Satz nicht geheimer Schlüssel-Wert-Paare. Der Verschlüsselungskontext ist kryptografisch an die verschlüsselten Daten gebunden, sodass derselbe Verschlüsselungskontext erforderlich ist, um das Feld zu entschlüsseln. Wenn Sie den erforderlichen Verschlüsselungskontext CMM verwenden, können Sie einen oder mehrere erforderliche Verschlüsselungskontextschlüssel (erforderliche Schlüssel) angeben, die in allen Verschlüsselungs- und Entschlüsselungsaufrufen enthalten sein müssen.

Note

Der erforderliche Verschlüsselungskontext CMM ist nur mit Version 3 interoperabel. x von AWS-Verschlüsselungs-SDK for Java Es ist mit keiner anderen Programmiersprachenimplementierung interoperabel. Wenn Sie Daten mit dem erforderlichen Verschlüsselungskontext CMM verschlüsseln, können Sie sie nur mit Version 3 entschlüsseln. x der AWS-Verschlüsselungs-SDK for Java oder Version 4. x von AWS Encryption SDK für .NET.

Beim Verschlüsseln AWS Encryption SDK wird überprüft, ob alle erforderlichen Verschlüsselungskontextschlüssel in dem von Ihnen angegebenen Verschlüsselungskontext enthalten sind. Das AWS Encryption SDK signiert die von Ihnen angegebenen

Verschlüsselungskontexte. Nur die Schlüssel-Wert-Paare, bei denen es sich nicht um erforderliche Schlüssel handelt, werden serialisiert und im Klartext-Format im Header der verschlüsselten Nachricht gespeichert, die der Verschlüsselungsvorgang zurückgibt.

Beim Entschlüsseln müssen Sie einen Verschlüsselungskontext angeben, der alle Schlüssel-Wert-Paare enthält, die die erforderlichen Schlüssel darstellen. Der AWS Encryption SDK verwendet diesen Verschlüsselungskontext und die im Header der verschlüsselten Nachricht gespeicherten Schlüssel-Wert-Paare, um den ursprünglichen Verschlüsselungskontext zu rekonstruieren, den Sie beim Verschlüsselungsvorgang angegeben haben. Wenn der ursprüngliche Verschlüsselungskontext AWS Encryption SDK nicht rekonstruiert werden kann, schlägt der Entschlüsselungsvorgang fehl. Wenn Sie ein Schlüssel-Wert-Paar angeben, das den erforderlichen Schlüssel mit einem falschen Wert enthält, kann die verschlüsselte Nachricht nicht entschlüsselt werden. Sie müssen dasselbe Schlüssel-Wert-Paar angeben, das bei Encrypt angegeben wurde.

Important

Überlegen Sie sich sorgfältig, welche Werte Sie für die erforderlichen Schlüssel in Ihrem Verschlüsselungskontext wählen. Sie müssen in der Lage sein, dieselben Schlüssel und die entsprechenden Werte beim Entschlüsseln erneut anzugeben. Wenn Sie die erforderlichen Schlüssel nicht reproduzieren können, kann die verschlüsselte Nachricht nicht entschlüsselt werden.

Im folgenden Beispiel wird ein AWS KMS Schlüsselbund mit dem erforderlichen Verschlüsselungskontext CMM initialisiert.

```
var encryptionContext = new Dictionary<string, string>()
{
    {"encryption", "context"},
    {"is not", "secret"},
    {"but adds", "useful metadata"},
    {"that can help you", "be confident that"},
    {"the data you are handling", "is what you think it is"}
};

// Instantiate the AWS Encryption SDK and material providers
var esdk = new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

// Instantiate the keyring input object
```

```
var createKeyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = kmsKey
};

// Create the keyring
var kmsKeyring = mpl.CreateAwsKmsKeyring(createKeyringInput);

var createCMMInput = new CreateRequiredEncryptionContextCMMInput
{
    UnderlyingCMM = mpl.CreateDefaultCryptographicMaterialsManager(new
    CreateDefaultCryptographicMaterialsManagerInput{Keyring = kmsKeyring}),
    // If you pass in a keyring but no underlying cmm, it will result in a failure
    because only cmm is supported.
    RequiredEncryptionContextKeys = new List<string>(encryptionContext.Keys)
};

// Create the required encryption context CMM
var requiredEcCMM = mpl.CreateRequiredEncryptionContextCMM(createCMMInput);
```

Wenn Sie einen AWS KMS Schlüsselbund verwenden, verwendet AWS Encryption SDK for .NET auch den Verschlüsselungskontext, um zusätzliche authentifizierte Daten (AAD) in den Aufrufen des Schlüsselbundes bereitzustellen. AWS KMS

AWS Encryption SDK für .NET-Beispiele

Die folgenden Beispiele zeigen die grundlegenden Codierungsmuster, die Sie beim Programmieren mit dem AWS Encryption SDK für .NET verwenden. Insbesondere instanziiieren Sie die Bibliothek AWS Encryption SDK und die Material Provider-Bibliothek. Bevor Sie dann jede Methode aufrufen, instanziiieren Sie ein Objekt, das die Eingabe für die Methode definiert. Dies ist dem Codierungsmuster sehr ähnlich, das Sie in der verwenden. AWS SDK for .NET

Beispiele für die Konfiguration von Optionen in der AWS Encryption SDK, z. B. die Angabe einer alternativen Algorithmussuite, die Beschränkung verschlüsselter Datenschlüssel und die Verwendung von Schlüsseln für AWS KMS mehrere Regionen, finden Sie unter [Konfiguration der AWS Encryption SDK](#).

Weitere Beispiele AWS Encryption SDK für die Programmierung mit für.NET finden Sie in den [Beispielen](#) im `aws-encryption-sdk-net` Verzeichnis des `aws-encryption-sdk-dafny` Repositorys unter GitHub.

Verschlüsseln von Daten im AWS Encryption SDK für .NET

Dieses Beispiel zeigt das grundlegende Muster für die Verschlüsselung von Daten. Es verschlüsselt eine kleine Datei mit Datenschlüsseln, die durch einen AWS KMS Umschließungsschlüssel geschützt sind.

Schritt 1: Instanzieren Sie die Bibliothek AWS Encryption SDK und die Materiallieferantenbibliothek.

Beginnen Sie mit der Instanziierung der Bibliothek AWS Encryption SDK und der Materiallieferantenbibliothek. Sie verwenden die Methoden in, AWS Encryption SDK um Daten zu verschlüsseln und zu entschlüsseln. Sie verwenden die Methoden in der Materialanbieter-Bibliothek, um die Schlüsselbunde zu erstellen, die angeben, welche Schlüssel Ihre Daten schützen.

Die Art AWS Encryption SDK und Weise, wie Sie die Material Provider-Bibliothek instanzieren, unterscheidet sich zwischen den Versionen 3. x und 4. x von AWS Encryption SDK für .NET. Alle folgenden Schritte sind für beide Versionen 3 identisch. x und 4. x von AWS Encryption SDK für .NET.

Version 3.x

```
// Instantiate the AWS Encryption SDK and material providers
var encryptionSdk = AwsEncryptionSdkFactory.CreateDefaultAwsEncryptionSdk();
var materialProviders =

    AwsCryptographicMaterialProvidersFactory.CreateDefaultAwsCryptographicMaterialProviders()
```

Version 4.x

```
// Instantiate the AWS Encryption SDK and material providers
var esdk = new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());
```

Schritt 2: Erstellen Sie ein Eingabeobjekt für den Schlüsselbund.

Jede Methode, die einen Schlüsselbund erstellt, hat eine entsprechende Eingabeobjektklasse. Um beispielsweise das Eingabeobjekt für die `CreateAwsKmsKeyring()` Methode zu erstellen, erstellen Sie eine Instanz der `CreateAwsKmsKeyringInput` Klasse.

Obwohl die Eingabe für diesen Schlüsselbund keinen [Generatorschlüssel spezifiziert, ist der einzelne KMS-Schlüssel](#), der durch den `KmsKeyId` Parameter angegeben wird, der

Generatorschlüssel. Er generiert und verschlüsselt den Datenschlüssel, der die Daten verschlüsselt.

Dieses Eingabeobjekt benötigt einen AWS KMS Client für den AWS-Region KMS-Schlüssel. Um einen AWS KMS Client zu erstellen, instanziiieren Sie die `AmazonKeyManagementServiceClient` Klasse in der AWS SDK for .NET. Wenn Sie den `AmazonKeyManagementServiceClient()` Konstruktor ohne Parameter aufrufen, wird ein Client mit den Standardwerten erstellt.

In einem AWS KMS Schlüsselbund, der AWS Encryption SDK für die Verschlüsselung mit für.NET verwendet wird, können Sie [die KMS-Schlüssel anhand der Schlüssel-ID, des Schlüssel-ARN, des Aliasnamens oder des Alias-ARN identifizieren](#). In einem AWS KMS Schlüsselbund, der zum Entschlüsseln verwendet wird, müssen Sie einen Schlüssel-ARN verwenden, um jeden KMS-Schlüssel zu identifizieren. Wenn Sie Ihren Verschlüsselungsschlüsselbund für die Entschlüsselung wiederverwenden möchten, verwenden Sie eine ARN-Schlüssel-ID für alle KMS-Schlüssel.

```
string keyArn = "arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";  
  
// Instantiate the keyring input object  
var kmsKeyringInput = new CreateAwsKmsKeyringInput  
{  
    KmsClient = new AmazonKeyManagementServiceClient(),  
    KmsKeyId = keyArn  
};
```

Schritt 3: Erstellen Sie den Schlüsselbund.

Um den Schlüsselbund zu erstellen, rufen Sie die Schlüsselbundmethode mit dem Schlüsselbund-Eingabeobjekt auf. In diesem Beispiel wird die `CreateAwsKmsKeyring()` Methode verwendet, die nur einen KMS-Schlüssel benötigt.

```
var keyring = materialProviders.CreateAwsKmsKeyring(kmsKeyringInput);
```

Schritt 4: Definieren Sie einen Verschlüsselungskontext.

Ein [Verschlüsselungskontext](#) ist ein optionales, aber dringend empfohlenes Element kryptografischer Operationen in der AWS Encryption SDK. Sie können ein oder mehrere nicht geheime Schlüssel-Wert-Paare definieren.

Note

Mit Version 4. x of the AWS Encryption SDK für .NET, Sie können in allen Verschlüsselungsanforderungen mit dem [erforderlichen Verschlüsselungskontext CMM einen Verschlüsselungskontext angeben](#).

```
// Define the encryption context
var encryptionContext = new Dictionary<string, string>()
{
    {"purpose", "test"}
};
```

Schritt 5: Erstellen Sie das Eingabeobjekt für die Verschlüsselung.

Bevor Sie die `Encrypt()` Methode aufrufen, erstellen Sie eine Instanz der `EncryptInput` Klasse.

```
string plaintext = File.ReadAllText("C:\\Documents\\CryptoTest\\TestFile.txt");

// Define the encrypt input
var encryptInput = new EncryptInput
{
    Plaintext = plaintext,
    Keyring = keyring,
    EncryptionContext = encryptionContext
};
```

Schritt 6: Verschlüsseln Sie den Klartext.

Verwenden Sie die `Encrypt()` Methode von AWS Encryption SDK, um den Klartext mit dem von Ihnen definierten Schlüsselbund zu verschlüsseln.

Die `EncryptOutput` `Encrypt()` Methode gibt Methoden zum Abrufen der verschlüsselten Nachricht (Ciphertext), den Verschlüsselungskontext und die Algorithmus-Suite an.

```
var encryptOutput = encryptionSdk.Encrypt(encryptInput);
```

Schritt 7: Holen Sie sich die verschlüsselte Nachricht.

Die `Decrypt()` Methode in AWS Encryption SDK for .NET verwendet das `Ciphertext` Mitglied der `EncryptOutput` Instanz.

Das `Ciphertext` Mitglied des `EncryptOutput` Objekts ist die [verschlüsselte Nachricht](#), ein portables Objekt, das die verschlüsselten Daten, verschlüsselten Datenschlüssel und Metadaten einschließlich des Verschlüsselungskontextes enthält. Sie können die verschlüsselte Nachricht sicher für einen längeren Zeitraum speichern oder sie an die `Decrypt()` Methode zur Wiederherstellung des Klartextes weiterleiten.

```
var encryptedMessage = encryptOutput.Ciphertext;
```

Entschlüsselung im strikten Modus in für.NET AWS Encryption SDK

Bewährte Methoden empfehlen, dass Sie die Schlüssel angeben, die Sie zum Entschlüsseln von Daten verwenden. Diese Option wird als strikter Modus bezeichnet. Der AWS Encryption SDK verwendet nur die KMS-Schlüssel, die Sie in Ihrem Schlüsselbund angeben, um den Chiffretext zu entschlüsseln. Die Schlüssel in Ihrem Entschlüsselungsschlüsselbund müssen mindestens einen der Schlüssel enthalten, mit denen die Daten verschlüsselt wurden.

Dieses Beispiel zeigt das grundlegende Muster der Entschlüsselung im strikten Modus mit dem AWS Encryption SDK für.NET.

Schritt 1: Instanzieren Sie die Bibliothek AWS Encryption SDK und die Material Provider-Bibliothek.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk = new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());
```

Schritt 2: Erstellen Sie das Eingabeobjekt für Ihren Schlüsselbund.

Um die Parameter für die Schlüsselbundmethode anzugeben, erstellen Sie ein Eingabeobjekt. Jede Schlüsselbundmethode in der AWS Encryption SDK für.NET hat ein entsprechendes Eingabeobjekt. Da in diesem Beispiel die `CreateAwsKmsKeyring()` Methode verwendet wird, um den Schlüsselbund zu erstellen, wird die `CreateAwsKmsKeyringInput` Klasse für die Eingabe instanziiert.

In einem Schlüsselbund für die Entschlüsselung müssen Sie einen Schlüssel-ARN verwenden, um KMS-Schlüssel zu identifizieren.

```
string keyArn = "arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";  
  
// Instantiate the keyring input object  
var kmsKeyringInput = new CreateAwsKmsKeyringInput  
{  
    KmsClient = new AmazonKeyManagementServiceClient(),  
    KmsKeyId = keyArn  
};
```

Schritt 3: Erstellen Sie den Schlüsselbund.

Um den Schlüsselbund für die Entschlüsselung zu erstellen, werden in diesem Beispiel die `CreateAwsKmsKeyring()` Methode und das Schlüsselbund-Eingabeobjekt verwendet.

```
var keyring = materialProviders.CreateAwsKmsKeyring(kmsKeyringInput);
```

Schritt 4: Erstellen Sie das Eingabeobjekt für die Entschlüsselung.

Um das Eingabeobjekt für die `Decrypt()` Methode zu erstellen, instanziiieren Sie die Klasse `DecryptInput`.

Der `Ciphertext` Parameter des `DecryptInput()` Konstruktors verwendet das `Ciphertext` Mitglied des `EncryptOutput` Objekts, das die `Encrypt()` Methode zurückgegeben hat. Die `Ciphertext` Eigenschaft stellt die [verschlüsselte Nachricht](#) dar, die die verschlüsselten Daten, verschlüsselten Datenschlüssel und Metadaten enthält, die zum Entschlüsseln der Nachricht AWS Encryption SDK benötigt werden.

Mit Version 4. x von AWS Encryption SDK für .NET können Sie den optionalen `EncryptionContext` Parameter verwenden, um Ihren Verschlüsselungskontext in der `Decrypt()` Methode anzugeben.

Verwenden Sie den `EncryptionContext` Parameter, um zu überprüfen, ob der beim Verschlüsseln verwendete Verschlüsselungskontext in dem Verschlüsselungskontext enthalten ist, der zum Entschlüsseln des Chiffretextes verwendet wird. Dadurch werden Paare zum Verschlüsselungskontext AWS Encryption SDK hinzugefügt, einschließlich der digitalen Signatur, wenn Sie eine Algorithmussuite mit Signierung verwenden, z. B. die Standard-Algorithmussuite.

```
var encryptedMessage = encryptOutput.Ciphertext;  
  
var decryptInput = new DecryptInput
```

```
{
    Ciphertext = encryptedMessage,
    Keyring = keyring,
    EncryptionContext = encryptionContext // OPTIONAL
};
```

Schritt 5: Entschlüsseln Sie den Chiffretext.

```
var decryptOutput = encryptionSdk.Decrypt(decryptInput);
```

Schritt 6: Überprüfen Sie den Verschlüsselungskontext — Version 3. x

Die `Decrypt()` Methode der Version 3. x von AWS Encryption SDK for .NET benötigt keinen Verschlüsselungskontext. Es ruft die Verschlüsselungskontextwerte aus den Metadaten in der verschlüsselten Nachricht ab. Bevor Sie den Klartext zurückgeben oder verwenden, sollten Sie jedoch überprüfen, ob der Verschlüsselungskontext, der zum Entschlüsseln des Chiffrextes verwendet wurde, den Verschlüsselungskontext enthält, den Sie bei der Verschlüsselung angegeben haben.

Stellen Sie sicher, dass der beim Verschlüsseln verwendete Verschlüsselungskontext in dem Verschlüsselungskontext enthalten ist, der zum Entschlüsseln des Chiffrextes verwendet wurde. Das AWS Encryption SDK fügt dem Verschlüsselungskontext Paare hinzu, einschließlich der digitalen Signatur, wenn Sie eine Algorithmussuite mit Signierung verwenden, z. B. die Standard-Algorithmussuite.

```
// Verify the encryption context
string contextKey = "purpose";
string contextValue = "test";

if (!decryptOutput.EncryptionContext.TryGetValue(contextKey, out var
    decryptContextValue)
    || !decryptContextValue.Equals(contextValue))
{
    throw new Exception("Encryption context does not match expected values");
}
```

Entschlüsseln mit einem Discovery-Schlüsselbund im für.NET AWS Encryption SDK

Anstatt die KMS-Schlüssel für die Entschlüsselung anzugeben, können Sie einen AWS KMS Discovery-Schlüsselbund angeben. Dabei handelt es sich um einen Schlüsselbund, der keine KMS-

Schlüssel angibt. Mit einem Discovery-Schlüsselbund können die Daten mit dem KMS-Schlüssel, der sie verschlüsselt hat, AWS Encryption SDK entschlüsselt werden, sofern der Anrufer über die Entschlüsselungsberechtigung für den Schlüssel verfügt. Für bewährte Methoden sollten Sie einen Erkennungsfiler hinzufügen, der die KMS-Schlüssel, die verwendet werden können, auf diejenigen beschränkt, die insbesondere AWS-Konten für eine bestimmte Partition verwendet werden können.

Der AWS Encryption SDK für.NET bietet einen einfachen Discovery-Schlüsselbund, für den ein AWS KMS Client erforderlich ist, und einen Discovery-Mehrfachschlüsselbund, für den Sie einen oder mehrere Schlüssel angeben müssen. AWS-Regionen Sowohl der Client als auch die Regionen beschränken die Anzahl der KMS-Schlüssel, die zum Entschlüsseln der verschlüsselten Nachricht verwendet werden können. Die Eingabeobjekte für beide Schlüsselbunde verwenden den empfohlenen Erkennungsfiler.

Das folgende Beispiel zeigt das Muster für die Entschlüsselung von Daten mit einem AWS KMS Discovery-Schlüsselbund und einem Discovery-Filter.

Schritt 1: Instanzieren Sie die Bibliothek AWS Encryption SDK und die Materiallieferantenbibliothek.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk = new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());
```

Schritt 2: Erstellen Sie das Eingabeobjekt für den Schlüsselbund.

Um die Parameter für die Schlüsselbundmethode anzugeben, erstellen Sie ein Eingabeobjekt. Jede Schlüsselbundmethode in der AWS Encryption SDK für.NET hat ein entsprechendes Eingabeobjekt. Da in diesem Beispiel die `CreateAwsKmsDiscoveryKeyring()` Methode verwendet wird, um den Schlüsselbund zu erstellen, wird die `CreateAwsKmsDiscoveryKeyringInput` Klasse für die Eingabe instanziiert.

```
List<string> accounts = new List<string> { "111122223333" };

var discoveryKeyringInput = new CreateAwsKmsDiscoveryKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    DiscoveryFilter = new DiscoveryFilter()
    {
        AccountIds = accounts,
        Partition = "aws"
    }
}
```

```
};
```

Schritt 3: Erstellen Sie den Schlüsselbund.

Um den Schlüsselbund für die Entschlüsselung zu erstellen, werden in diesem Beispiel die `CreateAwsKmsDiscoveryKeyring()` Methode und das Schlüsselbund-Eingabeobjekt verwendet.

```
var discoveryKeyring =  
    materialProviders.CreateAwsKmsDiscoveryKeyring(discoveryKeyringInput);
```

Schritt 4: Erstellen Sie das Eingabeobjekt für die Entschlüsselung.

Um das Eingabeobjekt für die `Decrypt()` Methode zu erstellen, instanziiieren Sie die Klasse `DecryptInput`. Der Wert des `Ciphertext` Parameters ist das `Ciphertext` Mitglied des `EncryptOutput` Objekts, das die `Encrypt()` Methode zurückgibt.

Mit Version 4. x von AWS Encryption SDK für .NET können Sie den optionalen `EncryptionContext` Parameter verwenden, um Ihren Verschlüsselungskontext in der `Decrypt()` Methode anzugeben.

Verwenden Sie den `EncryptionContext` Parameter, um zu überprüfen, ob der beim Verschlüsseln verwendete Verschlüsselungskontext in dem Verschlüsselungskontext enthalten ist, der zum Entschlüsseln des Chiffretextes verwendet wird. Dadurch werden Paare zum Verschlüsselungskontext AWS Encryption SDK hinzugefügt, einschließlich der digitalen Signatur, wenn Sie eine Algorithmussuite mit Signierung verwenden, z. B. die Standard-Algorithmussuite.

```
var ciphertext = encryptOutput.Ciphertext;  
  
var decryptInput = new DecryptInput  
{  
    Ciphertext = ciphertext,  
    Keyring = discoveryKeyring,  
    EncryptionContext = encryptionContext // OPTIONAL  
};  
var decryptOutput = encryptionSdk.Decrypt(decryptInput);
```

Schritt 5: Überprüfen Sie den Verschlüsselungskontext — Version 3. x

Die `Decrypt()` Methode der Version 3. x von AWS Encryption SDK for .NET nimmt keinen Verschlüsselungskontext an `Decrypt()`. Es ruft die Verschlüsselungskontextwerte aus den Metadaten in der verschlüsselten Nachricht ab. Bevor Sie den Klartext zurückgeben oder verwenden, sollten Sie jedoch überprüfen, ob der Verschlüsselungskontext, der zum Entschlüsseln des Chiffretextes verwendet wurde, den Verschlüsselungskontext enthält, den Sie bei der Verschlüsselung angegeben haben.

Stellen Sie sicher, dass der beim Verschlüsseln verwendete Verschlüsselungskontext in dem Verschlüsselungskontext enthalten ist, der zum Entschlüsseln des Chiffretextes verwendet wurde. Das AWS Encryption SDK fügt dem Verschlüsselungskontext Paare hinzu, einschließlich der digitalen Signatur, wenn Sie eine Algorithmussuite mit Signierung verwenden, z. B. die Standard-Algorithmussuite.

```
// Verify the encryption context
string contextKey = "purpose";
string contextValue = "test";

if (!decryptOutput.EncryptionContext.TryGetValue(contextKey, out var
decryptContextValue)
    || !decryptContextValue.Equals(contextValue))
{
    throw new Exception("Encryption context does not match expected values");
}
```

AWS-Verschlüsselungs-SDK for Java

In diesem Thema wird erklärt, wie das AWS-Verschlüsselungs-SDK for Java installiert und verwendet wird. Einzelheiten zur Programmierung mit dem AWS-Verschlüsselungs-SDK for Java finden Sie im [aws-encryption-sdk-java](#) Repository unter GitHub. Die API Dokumentation finden Sie im [Javadoc](#) für die AWS-Verschlüsselungs-SDK for Java

Themen

- [Voraussetzungen](#)
- [Installation](#)
- [AWS KMS Schlüsselanhänger in der AWS-Verschlüsselungs-SDK for Java](#)
- [Erforderliche Verschlüsselungskontexte in Version 3.x](#)

- [AWS-Verschlüsselungs-SDK for Java Beispiele](#)

Voraussetzungen

Stellen Sie vor der Installation von sicher AWS-Verschlüsselungs-SDK for Java, dass Sie die folgenden Voraussetzungen erfüllen.

Eine Java-Entwicklungsumgebung

Sie benötigen Java 8 oder höher. Gehen Sie auf der Oracle-Website zu [Java SE Downloads](#) und laden Sie dann das Java SE Development Kit herunter und installieren Sie es (JDK).

Wenn Sie Oracle verwendenJDK, müssen Sie auch die [Java Cryptography Extension \(JCE\) Unlimited Strength Jurisdiction Policy Files](#) herunterladen und installieren.

Bouncy Castle

Das AWS-Verschlüsselungs-SDK for Java erfordert [Bouncy Castle](#).

- AWS-Verschlüsselungs-SDK for Java Versionen 1.6.1 und höher verwenden Bouncy Castle, um kryptografische Objekte zu serialisieren und zu deserialisieren. [Sie können Bouncy Castle oder Bouncy Castle verwenden, um diese Anforderung zu erfüllen.](#) FIPS Hilfe zur Installation und Konfiguration von Bouncy Castle FIPS finden Sie in der [FIPSBC-Dokumentation](#), insbesondere in den Benutzerhandbüchern und den Sicherheitsrichtlinien. PDFs
- Frühere Versionen von AWS-Verschlüsselungs-SDK for Java verwenden die Kryptografie von Bouncy Castle für JavaAPI. Diese Anforderung wird nur von anderen Anbietern erfüllt, die nicht zu FIPS Bouncy Castle gehören.

Wenn Sie nicht über Bouncy Castle verfügen, gehen Sie zu den [neuesten Versionen von Bouncy Castle](#), um die entsprechende Anbieterdatei herunterzuladen. JDK [Sie können auch Apache Maven verwenden, um das Artefakt für den Standard-Bouncy Castle-Anbieter \(bcprov-ext-jdk15on\) oder das Artefakt für Bouncy Castle \(bc-fips\) zu erhalten.](#) FIPS

AWS SDK for Java

Version 3. x der AWS-Verschlüsselungs-SDK for Java erfordert das AWS SDK for Java 2.x, auch wenn Sie keine AWS KMS Schlüsselringe verwenden.

Ausführung 2. x oder früher von benötigt AWS-Verschlüsselungs-SDK for Java das nicht AWS SDK for Java. Die AWS SDK for Java ist jedoch erforderlich, um [AWS Key Management Service](#)(AWS KMS) als Hauptschlüsselanbieter zu verwenden. Ab AWS-Verschlüsselungs-SDK for Java Version 2.4.0 AWS-Verschlüsselungs-SDK for Java unterstützt der sowohl Version 1.x

als auch 2.x von. AWS SDK for Java AWS Encryption SDK Der Code für AWS SDK for Java 1.x und 2.x ist interoperabel. Sie können beispielsweise Daten mit AWS Encryption SDK Code verschlüsseln, der 1.x unterstützt, und sie mit Code entschlüsseln, der AWS SDK for Java 1.x unterstützt AWS SDK for Java 2.x (oder umgekehrt). Versionen vor 2.4.0 AWS-Verschlüsselungs-SDK for Java unterstützen nur 1.x. AWS SDK for Java Hinweise zur Aktualisierung Ihrer Version von finden Sie unter AWS Encryption SDK. [Migrieren Sie Ihre AWS Encryption SDK](#)

Wenn Sie Ihren AWS-Verschlüsselungs-SDK for Java Code von AWS SDK for Java 1.x auf aktualisieren AWS SDK for Java 2.x, ersetzen Sie Verweise auf die [AWSKMSchnittstelle](#) in AWS SDK for Java 1.x durch Verweise auf die [KmsClientSchnittstelle](#) in. AWS SDK for Java 2.x [Das AWS-Verschlüsselungs-SDK for Java unterstützt die Schnittstelle nicht. KmsAsyncClient](#) Aktualisieren Sie außerdem Ihren Code, sodass die AWS KMS zugehörigen Objekte im `kmssdkv2` Namespace statt im `kms` Namespace verwendet werden.

Verwenden Sie Apache Maven AWS SDK for Java, um das zu installieren.

- Um [das gesamte AWS SDK for Java](#) als Abhängigkeit zu importieren, deklarieren Sie es in Ihrer `pom.xml`-Datei.
- Um eine Abhängigkeit nur für das AWS KMS Modul in AWS SDK for Java 1.x zu erstellen, folgen Sie den Anweisungen zur [Angabe bestimmter Module](#) und setzen Sie den `artifactId` Wert auf `aws-java-sdk-kms`
- Um eine Abhängigkeit nur für das AWS KMS Modul in AWS SDK for Java 2.x zu erstellen, folgen Sie den Anweisungen zur [Angabe bestimmter Module](#). Stellen Sie „`groupId`“ auf `software.amazon.awssdk` und „`artifactId`“ auf `kms` ein.

Weitere Änderungen finden Sie unter [Was ist der Unterschied zwischen AWS SDK for Java 1.x und 2.x](#) im AWS SDK for Java 2.x Entwicklerhandbuch.

In den Java-Beispielen im AWS Encryption SDK Developer Guide wird der verwendet. AWS SDK for Java 2.x

Installation

Installieren Sie die neueste Version von AWS-Verschlüsselungs-SDK for Java.

Note

Alle AWS-Verschlüsselungs-SDK for Java Versionen vor 2.0.0 befinden sich in der [end-of-supportPhase](#).

Sie können sicher von Version 2.0 aus aktualisieren. x und höher auf die neueste Version von AWS-Verschlüsselungs-SDK for Java ohne Code- oder Datenänderungen. In Version 2.0 wurden jedoch [neue Sicherheitsfunktionen](#) eingeführt. x sind nicht abwärtskompatibel. Um von Versionen vor 1.7 zu aktualisieren. x auf Version 2.0. x und höher, Sie müssen zuerst auf die neueste Version 1 aktualisieren. x-Version von AWS Encryption SDK. Details hierzu finden Sie unter [Migrieren Sie Ihre AWS Encryption SDK](#).

Sie können das AWS-Verschlüsselungs-SDK for Java auf folgende Weise installieren.

manuell

Um das [aws-encryption-sdk-java](#) GitHubRepository zu installieren AWS-Verschlüsselungs-SDK for Java, klonen oder laden Sie es herunter.

Verwenden von Apache Maven

Das AWS-Verschlüsselungs-SDK for Java ist über [Apache Maven](#) mit der folgenden Abhängigkeitsdefinition verfügbar.

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-encryption-sdk-java</artifactId>
  <version>3.0.0</version>
</dependency>
```

Schauen Sie sich nach der SDK Installation von zunächst den [Java-Beispielcode](#) in diesem Handbuch und den [Javadoc](#) an. GitHub

AWS KMS Schlüsselanhänger in der AWS-Verschlüsselungs-SDK for Java

Ausführung 3. x von AWS-Verschlüsselungs-SDK for Java verwendet [Schlüsselringe](#), um die [Umschlagverschlüsselung](#) durchzuführen. Für die grundlegenden AWS KMS Schlüsselanhänger ist nur ein AWS-Verschlüsselungs-SDK for Java KMS Schlüssel erforderlich. Sie benötigen außerdem einen AWS KMS Client, der Ihnen die Möglichkeit gibt, den Client für den AWS-Region KMS Schlüssel zu konfigurieren.

Um einen AWS KMS Schlüsselbund mit einem oder mehreren Schlüsseln zu erstellen, verwenden Sie einen Schlüsselbund mit mehreren Schlüsseln. Der AWS-Verschlüsselungs-SDK for Java hat einen speziellen Mehrfachschlüsselbund, der einen oder mehrere AWS KMS Schlüssel akzeptiert,

und einen Standard-Mehrfachschlüsselbund, der einen oder mehrere Schlüsselringe eines beliebigen unterstützten Typs akzeptiert. Manche Programmierer ziehen es vor, eine Methode mit mehreren Schlüsselbunden zu verwenden, um all ihre Schlüsselbunde zu erstellen, und die Methode unterstützt diese Strategie. AWS-Verschlüsselungs-SDK for Java

[Das AWS-Verschlüsselungs-SDK for Java bietet grundlegende Einzelschlüsselringe und Schlüsselanhänger mit mehreren Schlüsseln für alle typischen Anwendungsfälle, einschließlich Schlüssel mit mehreren Regionen. AWS KMS](#)

Um beispielsweise einen AWS KMS Schlüsselbund mit einem Schlüssel zu erstellen, können Sie die `CreateAwsKmsKeyring()`-Methode AWS KMS verwenden.

```
// Instantiate the AWS Encryption SDK and material providers
final AwsCrypto crypto = AwsCrypto.builder().build();
final MaterialProviders materialProviders = MaterialProviders.builder()
    .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
    .build();

// Create the keyring
CreateAwsKmsKeyringInput kmsKeyringInput = CreateAwsKmsKeyringInput.builder()
    .kmsKeyId(keyArn)
    .kmsClient(KmsClient.create())
    .build();
IKeyring kmsKeyring = materialProviders.CreateAwsKmsKeyring(kmsKeyringInput);
```

Verwenden Sie die Methode, um einen Schlüsselbund mit einem oder mehreren AWS KMS Schlüsseln zu erstellen. `CreateAwsKmsMultiKeyring()` In diesem Beispiel werden zwei KMS Schlüssel verwendet. Um einen KMS Schlüssel anzugeben, verwenden Sie nur den `generator` Parameter. Der `msKeyIds` Parameter, der zusätzliche KMS Schlüssel angibt, ist optional.

Die Eingabe für diesen Schlüsselbund benötigt keinen AWS KMS Client. Stattdessen AWS Encryption SDK verwendet der den AWS KMS Standard-Client für jede Region, die durch einen KMS Schlüssel im Schlüsselbund repräsentiert wird. Wenn sich der durch den Wert des Generator Parameters identifizierte KMS Schlüssel beispielsweise in der Region USA West (Oregon) (`us-west-2`) befindet, AWS Encryption SDK wird ein AWS KMS Standard-Client für die `us-west-2` Region erstellt. Wenn Sie den AWS KMS Client anpassen müssen, verwenden Sie die `CreateAwsKmsKeyring()` Methode.

```
// Instantiate the AWS Encryption SDK and material providers
final AwsCrypto crypto = AwsCrypto.builder().build();
final MaterialProviders materialProviders = MaterialProviders.builder()
```

```

        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();

String generatorKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
List<String> additionalKey = Collections.singletonList("arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321");
// Create the keyring
final CreateAwsKmsMultiKeyringInput keyringInput =
    CreateAwsKmsMultiKeyringInput.builder()
        .generator(generatorKey)
        .kmsKeyIds(additionalKey)
        .build();
final IKeyring kmsKeyring = materialProviders.CreateAwsKmsMultiKeyring(keyringInput);

```

AWS-Verschlüsselungs-SDK for Java unterstützt AWS KMS Schlüsselringe, die symmetrische Verschlüsselung (SYMMETRIC_DEFAULT) oder RSA KMS asymmetrische Schlüssel verwenden. AWS KMS Schlüsselringe, die mit asymmetrischen RSA KMS Schlüsseln erstellt wurden, können nur ein key pair enthalten.

Um mit einem asymmetrischen RSA AWS KMS Schlüsselbund zu verschlüsseln, benötigen Sie [kms:GenerateDataKey](#) oder [kms:Encrypt](#) nicht, da Sie bei der Erstellung des Schlüsselbunds das Material der öffentlichen Schlüssel angeben müssen, das Sie für die Verschlüsselung verwenden möchten. Bei der Verschlüsselung mit diesem Schlüsselbund werden keine AWS KMS Anrufe getätigt. [Um mit einem asymmetrischen RSA AWS KMS Schlüsselbund zu entschlüsseln, benötigen Sie die kms:Decrypt-Berechtigung.](#)

Um einen asymmetrischen RSA AWS KMS Schlüsselbund zu erstellen, müssen Sie den öffentlichen Schlüssel und den privaten Schlüssel aus Ihrem asymmetrischen Schlüssel angeben. ARN RSA KMS Der öffentliche Schlüssel muss codiert sein. PEM Im folgenden Beispiel wird ein AWS KMS Schlüsselbund mit einem asymmetrischen RSA key pair erstellt.

```

// Instantiate the AWS Encryption SDK and material providers
final AwsCrypto crypto = AwsCrypto.builder()
    // Specify algorithmSuite without asymmetric signing here
    //
    // ALG_AES_128_GCM_IV12_TAG16_NO_KDF("0x0014"),
    // ALG_AES_192_GCM_IV12_TAG16_NO_KDF("0x0046"),
    // ALG_AES_256_GCM_IV12_TAG16_NO_KDF("0x0078"),
    // ALG_AES_128_GCM_IV12_TAG16_HKDF_SHA256("0x0114"),
    // ALG_AES_192_GCM_IV12_TAG16_HKDF_SHA256("0x0146"),

```

```

        // ALG_AES_256_GCM_IV12_TAG16_HKDF_SHA256("0x0178")

        .withEncryptionAlgorithm(CryptoAlgorithm.ALG_AES_256_GCM_IV12_TAG16_HKDF_SHA256)
        .build();

final MaterialProviders matProv = MaterialProviders.builder()
    .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
    .build();

// Create a KMS RSA keyring.
// This keyring takes in:
// - kmsClient
// - kmsKeyId: Must be an ARN representing an asymmetric RSA KMS key
// - publicKey: A ByteBuffer of a UTF-8 encoded PEM file representing the public
// key for the key passed into kmsKeyId
// - encryptionAlgorithm: Must be either RSAES_OAEP_SHA_256 or RSAES_OAEP_SHA_1
final CreateAwsKmsRsaKeyringInput createAwsKmsRsaKeyringInput =
    CreateAwsKmsRsaKeyringInput.builder()
        .kmsClient(KmsClient.create())
        .kmsKeyId(rsaKeyArn)
        .publicKey(publicKey)
        .encryptionAlgorithm(EncryptionAlgorithmSpec.RSAES_OAEP_SHA_256)
        .build();
IKeyring awsKmsRsaKeyring =
    matProv.CreateAwsKmsRsaKeyring(createAwsKmsRsaKeyringInput);

```

Erforderliche Verschlüsselungskontexte in Version 3.x


Mit Version 3. x von AWS-Verschlüsselungs-SDK for Java, Sie können den erforderlichen Verschlüsselungskontext verwenden, CMM um [Verschlüsselungskontexte](#) für Ihre kryptografischen Operationen erforderlich zu machen. Ein Verschlüsselungskontext ist eine Gruppe von nicht geheimen Schlüssel-Wert-Paaren. Der Verschlüsselungskontext ist kryptografisch an die verschlüsselten Daten gebunden, sodass derselbe Verschlüsselungskontext erforderlich ist, um das Feld zu entschlüsseln. Wenn Sie den erforderlichen Verschlüsselungskontext verwenden CMM, können Sie einen oder mehrere erforderliche Verschlüsselungskontextschlüssel (erforderliche Schlüssel) angeben, die in allen Verschlüsselungs- und Entschlüsselungsaufrufen enthalten sein müssen.

 Note

Der erforderliche Verschlüsselungskontext CMM ist nur mit Version 4 kompatibel. x des Forums AWS Encryption SDK . NET. Es ist mit keiner anderen Programmiersprachenimplementierung interoperabel. Wenn Sie Daten mit dem erforderlichen Verschlüsselungskontext verschlüsselnCMM, können Sie sie nur mit Version 3 entschlüsseln. x der AWS-Verschlüsselungs-SDK for Java oder Version 4. x der AWS Encryption SDK Form. NET.

Beim Verschlüsseln AWS Encryption SDK wird überprüft, ob alle erforderlichen Verschlüsselungskontextschlüssel in dem von Ihnen angegebenen Verschlüsselungskontext enthalten sind. Das AWS Encryption SDK signiert die von Ihnen angegebenen Verschlüsselungskontexte. Nur die Schlüssel-Wert-Paare, bei denen es sich nicht um erforderliche Schlüssel handelt, werden serialisiert und im Klartext-Format im Header der verschlüsselten Nachricht gespeichert, die der Verschlüsselungsvorgang zurückgibt.

Beim Entschlüsseln müssen Sie einen Verschlüsselungskontext angeben, der alle Schlüssel-Wert-Paare enthält, die die erforderlichen Schlüssel darstellen. Der AWS Encryption SDK verwendet diesen Verschlüsselungskontext und die im Header der verschlüsselten Nachricht gespeicherten Schlüssel-Wert-Paare, um den ursprünglichen Verschlüsselungskontext zu rekonstruieren, den Sie beim Verschlüsselungsvorgang angegeben haben. Wenn der ursprüngliche Verschlüsselungskontext AWS Encryption SDK nicht rekonstruiert werden kann, schlägt der Entschlüsselungsvorgang fehl. Wenn Sie ein Schlüssel-Wert-Paar angeben, das den erforderlichen Schlüssel mit einem falschen Wert enthält, kann die verschlüsselte Nachricht nicht entschlüsselt werden. Sie müssen dasselbe Schlüssel-Wert-Paar angeben, das bei Encrypt angegeben wurde.

 Important

Überlegen Sie sich sorgfältig, welche Werte Sie für die erforderlichen Schlüssel in Ihrem Verschlüsselungskontext wählen. Sie müssen in der Lage sein, dieselben Schlüssel und die entsprechenden Werte beim Entschlüsseln erneut anzugeben. Wenn Sie die erforderlichen Schlüssel nicht reproduzieren können, kann die verschlüsselte Nachricht nicht entschlüsselt werden.

Im folgenden Beispiel wird ein AWS KMS Schlüsselbund mit dem erforderlichen Verschlüsselungskontext initialisiert. CMM

```
// Instantiate the AWS Encryption SDK
final AwsCrypto crypto = AwsCrypto.builder()
    .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
    .build();

// Create your encryption context
final Map<String, String> encryptionContext = new HashMap<>();
encryptionContext.put("encryption", "context");
encryptionContext.put("is not", "secret");
encryptionContext.put("but adds", "useful metadata");
encryptionContext.put("that can help you", "be confident that");
encryptionContext.put("the data you are handling", "is what you think it is");

// Create a list of required encryption contexts
final List<String> requiredEncryptionContextKeys = Arrays.asList("encryption",
    "context");

// Create the keyring
final MaterialProviders materialProviders = MaterialProviders.builder()
    .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
    .build();
final CreateAwsKmsKeyringInput keyringInput = CreateAwsKmsKeyringInput.builder()
    .kmsKeyId(keyArn)
    .kmsClient(KmsClient.create())
    .build();
IKeyring kmsKeyring = materialProviders.CreateAwsKmsKeyring(keyringInput);

// Create the required encryption context CMM
ICryptographicMaterialsManager cmm =
    materialProviders.CreateDefaultCryptographicMaterialsManager(
        CreateDefaultCryptographicMaterialsManagerInput.builder()
            .keyring(kmsKeyring)
            .build()
    );
ICryptographicMaterialsManager requiredCMM =
    materialProviders.CreateRequiredEncryptionContextCMM(
        CreateRequiredEncryptionContextCMMInput.builder()
            .requiredEncryptionContextKeys(requiredEncryptionContextKeys)
            .underlyingCMM(cmm)
            .build()
    );
```

AWS-Verschlüsselungs-SDK for Java Beispiele

Die folgenden Beispiele zeigen Ihnen, wie Sie mit AWS-Verschlüsselungs-SDK for Java dem Daten ver- und entschlüsseln können. Diese Beispiele zeigen, wie Version 3 verwendet wird. x und später von AWS-Verschlüsselungs-SDK for Java. Version 3. x von AWS-Verschlüsselungs-SDK for Java benötigt die AWS SDK for Java 2.x. Version 3. x von AWS-Verschlüsselungs-SDK for Java ersetzt die [Hauptschlüsselanbieter](#) durch [Schlüsselringe](#). Beispiele, die frühere Versionen verwenden, finden Sie in der [Release-Liste](#) des [aws-encryption-sdk-java](#) Repositorys unter GitHub.

Themen

- [Verschlüsseln und Entschlüsseln von Zeichenfolgen](#)
- [Verschlüsseln und Entschlüsseln von Byte-Streams](#)
- [Verschlüsseln und Entschlüsseln von Bytestreams mit einem Multi-Keyring](#)

Verschlüsseln und Entschlüsseln von Zeichenfolgen

Das folgende Beispiel zeigt Ihnen, wie Sie Version 3 verwenden. x der AWS-Verschlüsselungs-SDK for Java zum Verschlüsseln und Entschlüsseln von Zeichenketten. Bevor Sie die Zeichenfolge verwenden, konvertieren Sie sie in ein Byte-Array.

[In diesem Beispiel wird ein AWS KMS Schlüsselbund verwendet.](#) Wenn Sie mit einem AWS KMS Schlüsselbund verschlüsseln, können Sie eine Schlüssel-ID, einen SchlüsselARN, einen Aliasnamen oder einen Alias verwenden, um die Schlüssel ARN zu identifizieren. KMS Beim Entschlüsseln müssen Sie einen Schlüssel verwenden, ARN um Schlüssel zu identifizieren. KMS

Wenn Sie die `encryptData()`-Methode aufrufen, wird eine [verschlüsselte Nachricht](#) (`CryptoResult`) zurückgegeben, die den Verschlüsselungstext, die verschlüsselten Datenschlüssel und den Verschlüsselungskontext enthält. Wenn Sie `CryptoResult` auf dem `getResult`-Objekt aufrufen, gibt es eine Base-64-codierte Zeichenfolgenversion der [verschlüsselten Nachricht](#) zurück, die Sie an die `decryptData()`-Methode übergeben können.

In ähnlicher Weise enthält das `decryptData()` zurückgegebene `CryptoResult` Objekt beim Aufrufen die Klartextnachricht und eine AWS KMS key ID. Bevor Ihre Anwendung den Klartext zurückgibt, stellen Sie sicher, dass die AWS KMS key ID und der Verschlüsselungskontext in der verschlüsselten Nachricht den Erwartungen entsprechen.

```
// Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0
```



```
package com.amazonaws.crypto.keyrings;

import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CommitmentPolicy;
import com.amazonaws.encryptionsdk.CryptoResult;
import software.amazon.cryptography.materialproviders.IKeyring;
import software.amazon.cryptography.materialproviders.MaterialProviders;
import
    software.amazon.cryptography.materialproviders.model.CreateAwsKmsMultiKeyringInput;
import software.amazon.cryptography.materialproviders.model.MaterialProvidersConfig;

import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Collections;
import java.util.Map;

/**
 * Encrypts and then decrypts data using an AWS KMS Keyring.
 *
 * <p>Arguments:
 *
 * <ol>
 *   <li>Key ARN: For help finding the Amazon Resource Name (ARN) of your AWS KMS
customer master
key (CMK), see 'Viewing Keys' at
http://docs.aws.amazon.com/kms/latest/developerguide/viewing-keys.html
</li>
</ol>
 */
public class BasicEncryptionKeyringExample {

    private static final byte[] EXAMPLE_DATA = "Hello
World".getBytes(StandardCharsets.UTF_8);

    public static void main(final String[] args) {
        final String keyArn = args[0];

        encryptAndDecryptWithKeyring(keyArn);
    }

    public static void encryptAndDecryptWithKeyring(final String keyArn) {
        // 1. Instantiate the SDK
        // This builds the AwsCrypto client with the RequireEncryptRequireDecrypt
commitment policy,
```

```
// which means this client only encrypts using committing algorithm suites and
enforces
// that the client will only decrypt encrypted messages that were created with a
committing
// algorithm suite.
// This is the default commitment policy if you build the client with
// `AwsCrypto.builder().build()`
// or `AwsCrypto.standard()`.
final AwsCrypto crypto =
    AwsCrypto.builder()
        .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
        .build();

// 2. Create the AWS KMS keyring.
// This example creates a multi keyring, which automatically creates the KMS
client.
final MaterialProviders materialProviders =
    MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
final CreateAwsKmsMultiKeyringInput keyringInput =
    CreateAwsKmsMultiKeyringInput.builder().generator(keyArn).build();
final IKeyring kmsKeyring =
materialProviders.CreateAwsKmsMultiKeyring(keyringInput);

// 3. Create an encryption context
// We recommend using an encryption context whenever possible
// to protect integrity. This sample uses placeholder values.
// For more information see:
// blogs.aws.amazon.com/security/post/Tx2LZ6WBJJANTNW/How-to-Protect-the-Integrity-
of-Your-Encrypted-Data-by-Using-AWS-Key-Management
final Map<String, String> encryptionContext =
    Collections.singletonMap("ExampleContextKey", "ExampleContextValue");

// 4. Encrypt the data
final CryptoResult<byte[], ?> encryptResult =
    crypto.encryptData(kmsKeyring, EXAMPLE_DATA, encryptionContext);
final byte[] ciphertext = encryptResult.getResult();

// 5. Decrypt the data
final CryptoResult<byte[], ?> decryptResult =
    crypto.decryptData(
        kmsKeyring,
        ciphertext,
```

```
        // Verify that the encryption context in the result contains the
        // encryption context supplied to the encryptData method
        encryptionContext);

    // 6. Verify that the decrypted plaintext matches the original plaintext
    assert Arrays.equals(decryptResult.getResult(), EXAMPLE_DATA);
}
}
```

Verschlüsseln und Entschlüsseln von Byte-Streams

Das folgende Beispiel zeigt Ihnen, wie Sie AWS Encryption SDK Bytestreams verschlüsseln und entschlüsseln können.

In diesem Beispiel wird ein [AESRaw-Schlüsselbund](#) verwendet.

Bei der Verschlüsselung verwendet dieses Beispiel die `AwsCrypto.builder().withEncryptionAlgorithm()` Methode, um eine Algorithmusuite ohne [digitale](#) Signaturen anzugeben. Bei der Entschlüsselung wird in diesem Beispiel die Methode verwendet, um sicherzustellen, dass der Chiffretext nicht signiert ist. `createUnsignedMessageDecryptingStream()` Die `createUnsignedMessageDecryptingStream()` Methode schlägt fehl, wenn sie auf einen Chiffretext mit einer digitalen Signatur trifft.

Wenn Sie mit der Standard-Algorithmus-Suite verschlüsseln, die digitale Signaturen enthält, verwenden Sie stattdessen die `createDecryptingStream()` Methode, wie im nächsten Beispiel gezeigt.

```
// Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

package com.amazonaws.crypto.keyrings;

import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CommitmentPolicy;
import com.amazonaws.encryptionsdk.CryptoAlgorithm;
import com.amazonaws.encryptionsdk.CryptoInputStream;
import com.amazonaws.encryptionsdk.jce.JceMasterKey;
import com.amazonaws.util.IOUtils;
import software.amazon.cryptography.materialproviders.IKeyring;
import software.amazon.cryptography.materialproviders.MaterialProviders;
import software.amazon.cryptography.materialproviders.model.AesWrappingAlg;
```

```
import software.amazon.cryptography.materialproviders.model.CreateRawAesKeyringInput;
import software.amazon.cryptography.materialproviders.model.MaterialProvidersConfig;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.security.SecureRandom;
import java.util.Collections;
import java.util.Map;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;

/**
 * <p>
 * Encrypts and then decrypts a file under a random key.
 *
 * <p>
 * Arguments:
 * <ol>
 * <li>Name of file containing plaintext data to encrypt
 * </ol>
 *
 * <p>
 * This program demonstrates using a standard Java {@link SecretKey} object as a {@link
 * IKeyring} to
 * encrypt and decrypt streaming data.
 */
public class FileStreamingKeyringExample {
    private static String srcFile;

    public static void main(String[] args) throws IOException {
        srcFile = args[0];

        // In this example, we generate a random key. In practice,
        // you would get a key from an existing store
        SecretKey cryptoKey = retrieveEncryptionKey();

        // Create a Raw Aes Keyring using the random key and an AES-GCM encryption
        algorithm
        final MaterialProviders materialProviders = MaterialProviders.builder()
            .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
            .build();
```

```
    final CreateRawAesKeyringInput keyringInput =
CreateRawAesKeyringInput.builder()
    .wrappingKey(ByteBuffer.wrap(cryptoKey.getEncoded()))
    .keyNamespace("Example")
    .keyName("RandomKey")
    .wrappingAlg(AesWrappingAlg.ALG_AES128_GCM_IV12_TAG16)
    .build();
    IKeyring keyring = materialProviders.CreateRawAesKeyring(keyringInput);

    // Instantiate the SDK.
    // This builds the AwsCrypto client with the RequireEncryptRequireDecrypt
commitment policy,
    // which means this client only encrypts using committing algorithm suites and
enforces
    // that the client will only decrypt encrypted messages that were created with
a committing
    // algorithm suite.
    // This is the default commitment policy if you build the client with
    // `AwsCrypto.builder().build()`
    // or `AwsCrypto.standard()`.
    // This example encrypts with an algorithm suite that doesn't include signing
for faster decryption,
    // since this use case assumes that the contexts that encrypt and decrypt are
equally trusted.
    final AwsCrypto crypto = AwsCrypto.builder()
        .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)

.withEncryptionAlgorithm(CryptoAlgorithm.ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY)
        .build();

    // Create an encryption context to identify the ciphertext
    Map<String, String> context = Collections.singletonMap("Example",
"FileStreaming");

    // Because the file might be too large to load into memory, we stream the data,
instead of
    //loading it all at once.
    FileInputStream in = new FileInputStream(srcFile);
    CryptoInputStream<JceMasterKey> encryptingStream =
crypto.createEncryptingStream(keyring, in, context);

    FileOutputStream out = new FileOutputStream(srcFile + ".encrypted");
    IOUtils.copy(encryptingStream, out);
    encryptingStream.close();
```

```

        out.close();

        // Decrypt the file. Verify the encryption context before returning the
plaintext.
        // Since the data was encrypted using an unsigned algorithm suite, use the
recommended
        // createUnsignedMessageDecryptingStream method, which only accepts unsigned
messages.
        in = new FileInputStream(srcFile + ".encrypted");
        CryptoInputStream<JceMasterKey> decryptingStream =
crypto.createUnsignedMessageDecryptingStream(keyring, in);
        // Does it contain the expected encryption context?
        if
(!"FileStreaming".equals(decryptingStream.getCryptoResult().getEncryptionContext().get("Exempl
{
            throw new IllegalStateException("Bad encryption context");
        }

        // Write the plaintext data to disk.
        out = new FileOutputStream(srcFile + ".decrypted");
        IOUtils.copy(decryptingStream, out);
        decryptingStream.close();
        out.close();
    }

/**
 * In practice, this key would be saved in a secure location.
 * For this demo, we generate a new random key for each operation.
 */
private static SecretKey retrieveEncryptionKey() {
    SecureRandom rnd = new SecureRandom();
    byte[] rawKey = new byte[16]; // 128 bits
    rnd.nextBytes(rawKey);
    return new SecretKeySpec(rawKey, "AES");
}
}

```

Verschlüsseln und Entschlüsseln von Bytestreams mit einem Multi-Keyring

Das folgende Beispiel zeigt Ihnen, wie Sie den AWS Encryption SDK mit einem [Mehrfachschlüsselbund](#) verwenden. Wenn Sie einen Multi-Schlüsselbund verwenden, um Daten zu verschlüsseln, können alle Umhüllungsschlüssel in einem seiner Schlüsselbunde diese Daten

entschlüsseln. In diesem Beispiel werden ein [AWS KMS Schlüsselbund und ein Raw-Schlüsselbund als RSA untergeordnete Schlüsselanhänger](#) verwendet.

[In diesem Beispiel wird mit der standardmäßigen Algorithmsgsuite verschlüsselt, die eine digitale Signatur enthält.](#) Beim Streaming AWS Encryption SDK gibt der Klartext nach Integritätsprüfungen, aber bevor die digitale Signatur verifiziert wurde, frei. Um zu vermeiden, dass der Klartext verwendet wird, bis die Signatur verifiziert ist, puffert dieses Beispiel den Klartext und schreibt ihn erst auf die Festplatte, wenn die Entschlüsselung und Überprüfung abgeschlossen sind.

```
// Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

package com.amazonaws.crypto.keyrings;

import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CommitmentPolicy;
import com.amazonaws.encryptionsdk.CryptoOutputStream;
import com.amazonaws.util.IOUtils;
import software.amazon.cryptography.materialproviders.IKeyring;
import software.amazon.cryptography.materialproviders.MaterialProviders;
import
    software.amazon.cryptography.materialproviders.model.CreateAwsKmsMultiKeyringInput;
import software.amazon.cryptography.materialproviders.model.CreateMultiKeyringInput;
import software.amazon.cryptography.materialproviders.model.CreateRawRsaKeyringInput;
import software.amazon.cryptography.materialproviders.model.MaterialProvidersConfig;
import software.amazon.cryptography.materialproviders.model.PaddingScheme;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.nio.ByteBuffer;
import java.security.GeneralSecurityException;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.util.Collections;

/**
 * <p>
 * Encrypts a file using both AWS KMS Key and an asymmetric key pair.
 *
 * <p>
 * Arguments:
```

```

* <ol>
* <li>Key ARN: For help finding the Amazon Resource Name (ARN) of your AWS KMS key,
*   see 'Viewing Keys' at http://docs.aws.amazon.com/kms/latest/developerguide/viewing-keys.html
*
* <li>Name of file containing plaintext data to encrypt
* </ol>
* <p>
* You might use AWS Key Management Service (AWS KMS) for most encryption and
* decryption operations, but
* still want the option of decrypting your data offline independently of AWS KMS. This
* sample
* demonstrates one way to do this.
* <p>
* The sample encrypts data under both an AWS KMS key and an "escrowed" RSA key pair
* so that either key alone can decrypt it. You might commonly use the AWS KMS key for
* decryption. However,
* at any time, you can use the private RSA key to decrypt the ciphertext independent
* of AWS KMS.
* <p>
* This sample uses the RawRsaKeyring to generate a RSA public-private key pair
* and saves the key pair in memory. In practice, you would store the private key in a
* secure offline
* location, such as an offline HSM, and distribute the public key to your development
* team.
*/
public class EscrowedEncryptKeyringExample {
    private static ByteBuffer publicEscrowKey;
    private static ByteBuffer privateEscrowKey;

    public static void main(final String[] args) throws Exception {
        // This sample generates a new random key for each operation.
        // In practice, you would distribute the public key and save the private key in
        secure
        // storage.
        generateEscrowKeyPair();

        final String kmsArn = args[0];
        final String fileName = args[1];

        standardEncrypt(kmsArn, fileName);
        standardDecrypt(kmsArn, fileName);

        escrowDecrypt(fileName);
    }
}

```



```
}

    private static void standardEncrypt(final String kmsArn, final String fileName)
throws Exception {
    // Encrypt with the KMS key and the escrowed public key
    // 1. Instantiate the SDK
    // This builds the AwsCrypto client with the RequireEncryptRequireDecrypt
commitment policy,
    // which means this client only encrypts using committing algorithm suites and
enforces
    // that the client will only decrypt encrypted messages that were created with
a committing
    // algorithm suite.
    // This is the default commitment policy if you build the client with
    // `AwsCrypto.builder().build()`
    // or `AwsCrypto.standard()`.
    final AwsCrypto crypto = AwsCrypto.builder()
        .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
        .build();

    // 2. Create the AWS KMS keyring.
    // This example creates a multi keyring, which automatically creates the KMS
client.
    final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
    final CreateAwsKmsMultiKeyringInput keyringInput =
CreateAwsKmsMultiKeyringInput.builder()
        .generator(kmsArn)
        .build();
    IKeyring kmsKeyring = matProv.CreateAwsKmsMultiKeyring(keyringInput);

    // 3. Create the Raw Rsa Keyring with Public Key.
    final CreateRawRsaKeyringInput encryptingKeyringInput =
CreateRawRsaKeyringInput.builder()
        .keyName("Escrow")
        .keyNamespace("Escrow")
        .paddingScheme(PaddingScheme.OAEP_SHA512_MGF1)
        .publicKey(publicEscrowKey)
        .build();
    IKeyring rsaPublicKeyring =
matProv.CreateRawRsaKeyring(encryptingKeyringInput);

    // 4. Create the multi-keyring.
```

```
    final CreateMultiKeyringInput createMultiKeyringInput =
CreateMultiKeyringInput.builder()
    .generator(kmsKeyring)
    .childKeyrings(Collections.singletonList(rsaPublicKeyring))
    .build();
    IKeyring multiKeyring = matProv.CreateMultiKeyring(createMultiKeyringInput);

    // 5. Encrypt the file
    // To simplify this code example, we omit the encryption context. Production
code should always
    // use an encryption context.
    final FileInputStream in = new FileInputStream(fileName);
    final FileOutputStream out = new FileOutputStream(fileName + ".encrypted");
    final CryptoOutputStream<?> encryptingStream =
crypto.createEncryptingStream(multiKeyring, out);

    IOUtils.copy(in, encryptingStream);
    in.close();
    encryptingStream.close();
}

private static void standardDecrypt(final String kmsArn, final String fileName)
throws Exception {
    // Decrypt with the AWS KMS key and the escrow public key.

    // 1. Instantiate the SDK.
    // This builds the AwsCrypto client with the RequireEncryptRequireDecrypt
commitment policy,
    // which means this client only encrypts using committing algorithm suites and
enforces
    // that the client will only decrypt encrypted messages that were created with
a committing
    // algorithm suite.
    // This is the default commitment policy if you build the client with
    // `AwsCrypto.builder().build()`
    // or `AwsCrypto.standard()`.
    final AwsCrypto crypto = AwsCrypto.builder()
        .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
        .build();

    // 2. Create the AWS KMS keyring.
    // This example creates a multi keyring, which automatically creates the KMS
client.
    final MaterialProviders matProv = MaterialProviders.builder()
```

```
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
    final CreateAwsKmsMultiKeyringInput keyringInput =
CreateAwsKmsMultiKeyringInput.builder()
        .generator(kmsArn)
        .build();
    IKeyring kmsKeyring = matProv.CreateAwsKmsMultiKeyring(keyringInput);

    // 3. Create the Raw Rsa Keyring with Public Key.
    final CreateRawRsaKeyringInput encryptingKeyringInput =
CreateRawRsaKeyringInput.builder()
        .keyName("Escrow")
        .keyNamespace("Escrow")
        .paddingScheme(PaddingScheme.OAEP_SHA512_MGF1)
        .publicKey(publicEscrowKey)
        .build();
    IKeyring rsaPublicKeyring =
matProv.CreateRawRsaKeyring(encryptingKeyringInput);

    // 4. Create the multi-keyring.
    final CreateMultiKeyringInput createMultiKeyringInput =
CreateMultiKeyringInput.builder()
        .generator(kmsKeyring)
        .childKeyrings(Collections.singletonList(rsaPublicKeyring))
        .build();
    IKeyring multiKeyring = matProv.CreateMultiKeyring(createMultiKeyringInput);

    // 5. Decrypt the file
    // To simplify this code example, we omit the encryption context. Production
code should always
    // use an encryption context.
    final FileInputStream in = new FileInputStream(fileName + ".encrypted");
    final FileOutputStream out = new FileOutputStream(fileName + ".decrypted");
    // Since we are using a signing algorithm suite, we avoid streaming decryption
directly to the output file,
    // to ensure that the trailing signature is verified before writing any
untrusted plaintext to disk.
    final ByteArrayOutputStream plaintextBuffer = new ByteArrayOutputStream();
    final CryptoOutputStream<?> decryptingStream =
crypto.createDecryptingStream(multiKeyring, plaintextBuffer);
    IOUtils.copy(in, decryptingStream);
    in.close();
    decryptingStream.close();
```

```
        final ByteArrayInputStream plaintextReader = new
ByteArrayInputStream(plaintextBuffer.toByteArray());
        IOUtils.copy(plaintextReader, out);
        out.close();
    }

private static void escrowDecrypt(final String fileName) throws Exception {
    // You can decrypt the stream using only the private key.
    // This method does not call AWS KMS.

    // 1. Instantiate the SDK
    final AwsCrypto crypto = AwsCrypto.standard();

    // 2. Create the Raw Rsa Keyring with Private Key.
    final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
    final CreateRawRsaKeyringInput encryptingKeyringInput =
CreateRawRsaKeyringInput.builder()
        .keyName("Escrow")
        .keyNamespace("Escrow")
        .paddingScheme(PaddingScheme.OAEP_SHA512_MGF1)
        .publicKey(publicEscrowKey)
        .privateKey(privateEscrowKey)
        .build();
    IKeyring escrowPrivateKeyring =
matProv.CreateRawRsaKeyring(encryptingKeyringInput);

    // 3. Decrypt the file
    // To simplify this code example, we omit the encryption context. Production
code should always
    // use an encryption context.
    final FileInputStream in = new FileInputStream(fileName + ".encrypted");
    final FileOutputStream out = new FileOutputStream(fileName + ".deescrowed");
    final CryptoOutputStream<?> decryptingStream =
crypto.createDecryptingStream(escrowPrivateKeyring, out);
    IOUtils.copy(in, decryptingStream);
    in.close();
    decryptingStream.close();
}

private static void generateEscrowKeyPair() throws GeneralSecurityException {
```

```
final KeyPairGenerator kg = KeyPairGenerator.getInstance("RSA");
kg.initialize(4096); // Escrow keys should be very strong
final KeyPair keyPair = kg.generateKeyPair();
publicEscrowKey = RawRsaKeyringExample.getPEMPublicKey(keyPair.getPublic());
privateEscrowKey = RawRsaKeyringExample.getPEMPrivateKey(keyPair.getPrivate());

}
}
```

AWS-Verschlüsselungs-SDK for JavaScript

Die AWS-Verschlüsselungs-SDK for JavaScript wurde entwickelt, um eine clientseitige Verschlüsselungsbibliothek für Entwickler bereitzustellen, die Webbrowseranwendungen in JavaScript oder Webserveranwendungen in Node.js schreiben.

Wie alle Implementierungen des AWS Encryption SDK bietet auch die AWS-Verschlüsselungs-SDK for JavaScript erweiterte Datenschutzfunktionen. Dazu gehören die [Envelope-Verschlüsselung](#), [zusätzliche authentifizierte Daten](#) (AAD) und [Algorithmen-Pakete](#) mit sicherem, authentifiziertem, symmetrischem Schlüssel, wie z. B. 256-Bit-AES-GCM mit Schlüsselableitung und Signatur.

Alle sprachspezifischen Implementierungen des AWS Encryption SDK sind so konzipiert, dass sie interoperabel sind, vorbehaltlich der Einschränkungen der Sprache. Weitere Informationen zu Spracheinschränkungen für JavaScript finden Sie unter [the section called "Kompatibilität"](#).

Weitere Informationen

- Für Details über die Programmierung mit der AWS-Verschlüsselungs-SDK for JavaScript, finden Sie unter [aws-encryption-sdk-JavaScript-Repository](#) auf GitHub aus.
- Beispiele für Programmierung finden Sie unter [the section called "Beispiele"](#) und die [example-browser](#) und [example-node](#) Module im [aws-encryption-sdk-JavaScript-Repository](#).
- Für ein reales Beispiel für die Verwendung des AWS-Verschlüsselungs-SDK for JavaScript Informationen zum Verschlüsseln von Daten in einer Webanwendung finden Sie unter [So aktivieren Sie die Verschlüsselung in einem Browser mit dem AWS-Verschlüsselungs-SDK for JavaScript und Node.js im AWS Blog-Sicherheit](#).

Themen

- [Kompatibilität des AWS-Verschlüsselungs-SDK for JavaScript](#)
- [Installieren des AWS-Verschlüsselungs-SDK for JavaScript](#)

- [Module im AWS-Verschlüsselungs-SDK for JavaScript](#)
- [AWS-Verschlüsselungs-SDK for JavaScript Beispiele für](#)

Kompatibilität des AWS-Verschlüsselungs-SDK for JavaScript

Das AWS-Verschlüsselungs-SDK for JavaScript ist so konzipiert, dass es mit anderen Sprachenimplementierungen des AWS Encryption SDK kompatibel ist. In den meisten Fällen können Sie Daten mit dem AWS-Verschlüsselungs-SDK for JavaScript verschlüsseln und mit jeder anderen Sprachenimplementierung, einschließlich der [AWS Encryption SDK-Befehlszeilenschnittstelle](#) entschlüsseln. Und Sie können das AWS-Verschlüsselungs-SDK for JavaScript verwenden, um [verschlüsselte Nachrichten](#) zu entschlüsseln, die von anderen Sprachenimplementierungen des AWS Encryption SDK erzeugt werden.

Wenn Sie jedoch das AWS-Verschlüsselungs-SDK for JavaScript verwenden, müssen Sie einige Kompatibilitätsprobleme in der JavaScript-Sprachenimplementierung und in Webbrowsern beachten.

Wenn Sie verschiedene Sprachenimplementierungen verwenden, achten Sie außerdem darauf, kompatible Hauptschlüsselanbieter, Hauptschlüssel und Schlüsselbunde zu konfigurieren. Details hierzu finden Sie unter [Schlüsselbund-Kompatibilität](#).

AWS-Verschlüsselungs-SDK for JavaScript-Kompatibilität

Die JavaScript-Implementierung des AWS Encryption SDK unterscheidet sich von anderen Sprachenimplementierungen auf folgende Weise:

- Die Verschlüsselungsoperation des AWS-Verschlüsselungs-SDK for JavaScript gibt keinen Verschlüsselungstext ohne Frame zurück. Das AWS-Verschlüsselungs-SDK for JavaScript entschlüsselt jedoch Verschlüsselungstext mit und ohne Frame, der von anderen Sprachenimplementierungen des AWS Encryption SDK zurückgegeben wird.
- Ab Node.js-Version 12.9.0 unterstützt Node.js die folgenden RSA-Schlüsselumhüllungsoptionen:
 - OAEP mit SHA1, SHA256, SHA384 oder SHA512
 - OAEP mit SHA1 und MGF1 mit SHA1
 - PKCS1v15
- Vor Version 12.9.0 unterstützt Node.js nur die folgenden RSA-Schlüsselumhüllungsoptionen:
 - OAEP mit SHA1 und MGF1 mit SHA1
 - PKCS1v15

Browserkompatibilität

Einige Webbrowser unterstützen keine grundlegenden kryptografischen Operationen, die für das AWS-Verschlüsselungs-SDK for JavaScript erforderlich sind. Sie können einige der fehlenden Operationen kompensieren, indem Sie einen Fallback für die WebCryptoAPI, die der Browser implementiert.

Webbrowser-Einschränkungen

Die folgenden Einschränkungen gelten für alle Webbrowser:

- Die WebCryptoAPI unterstützt keine Schlüsselumhüllung von PKCS1v15.
- Browser unterstützen keine 192-Bit-Schlüssel.

Erforderliche kryptografische Operationen

Das AWS-Verschlüsselungs-SDK for JavaScript erfordert die folgenden Operationen in Webbrowsern. Wenn ein Browser diese Operationen nicht unterstützt, ist er nicht mit dem AWS-Verschlüsselungs-SDK for JavaScript kompatibel.

- Der Browser muss `crypto.getRandomValues()` enthalten, was eine Methode zum Generieren kryptografisch zufälliger Werte ist. Weitere Informationen zu den Webbrowser-Versionen, die `crypto.getRandomValues()` unterstützen, finden Sie unter [Kann ich Krypto verwenden.getRandomValues\(\)?aus](#).

Erforderlicher Fallback

Das AWS-Verschlüsselungs-SDK for JavaScript erfordert die folgenden Bibliotheken und Operationen in Webbrowsern. Wenn Sie einen Webbrowser unterstützen, der diese Anforderungen nicht erfüllt, müssen Sie einen Fallback konfigurieren. Andernfalls schlagen Versuche, das AWS-Verschlüsselungs-SDK for JavaScript mit dem Browser zu verwenden, fehl.

- Die WebCryptoAPI, die grundlegende kryptografische Operationen in Webanwendungen durchführt, ist nicht für alle Browser verfügbar. Weitere Informationen zu den Webbrowser-Versionen, die Web-Kryptografie unterstützen, finden Sie unter [Kann ich Web-Kryptografie verwenden?](#).
- Moderne Versionen des Safari-Webrowsers unterstützen keine AES-GCM-Verschlüsselung von Null-Bytes, was das AWS Encryption SDK erfordert. Wenn der Browser das WebCryptoAPI, aber

AES-GCM nicht zum Verschlüsseln von Null-Bytes verwenden kann. Das AWS-Verschlüsselungs-SDK for JavaScript verwendet die Fallback-Bibliothek nur für die Nullbyte-Verschlüsselung. Es benutzt das WebCryptoAPI für alle anderen Operationen.

Um einen Fallback für eine der Einschränkungen zu konfigurieren, fügen Sie die folgenden Anweisungen zu Ihrem Code hinzu. Geben Sie in der Funktion [configureFallback](#) eine Bibliothek an, die die fehlenden Funktionen unterstützt. Im folgenden Beispiel wird Microsoft Research verwendet. JavaScript-Kryptografie-Bibliothek (`msrcrypto`), aber Sie können es durch eine kompatible Bibliothek ersetzen. Ein vollständiges Beispiel finden Sie unter [fallback.ts](#).

```
import { configureFallback } from '@aws-crypto/client-browser'  
configureFallback(msrCrypto)
```

Installieren des AWS-Verschlüsselungs-SDK for JavaScript

Das AWS-Verschlüsselungs-SDK for JavaScript besteht aus einer Sammlung von voneinander abhängigen Modulen. Einige der Module sind nur Sammlungen von Modulen, die für die Zusammenarbeit konzipiert sind. Einige Module sind so konzipiert, dass sie unabhängig voneinander arbeiten. Für alle Implementierungen sind einige Module erforderlich, einige andere werden nur für spezielle Fälle benötigt. Informationen zu den Modulen finden Sie im AWS Encryption SDK zum JavaScript, siehe [Module im AWS-Verschlüsselungs-SDK for JavaScript](#) und der `README.md`-Datei in jedem der Module in der [aws-encryption-sdk-javascript](#) Repository auf GitHub.

Note

Alle Versionen des AWS-Verschlüsselungs-SDK for JavaScript früher als 2.0.0 sind in [end-of-supportPhase](#).

Sie können sicher von Version 2.0 aktualisieren und später zur neuesten Version des AWS-Verschlüsselungs-SDK for JavaScript ohne Code- oder Datenänderungen. Allerdings [neue Sicherheitsfunktionen](#) eingeführt in Version 2.0.x sind nicht abwärtskompatibel. Um von Versionen vor 1.7 zu aktualisieren auf Version 2.0.x und später müssen Sie zuerst auf die neueste Version 1 aktualisieren. Version des AWS-Verschlüsselungs-SDK for JavaScript. Details hierzu finden Sie unter [Migrieren Sie Ihre AWS Encryption SDK](#).

Um die Module zu installieren, verwenden Sie den [npm-Paketmanager](#).

Verwenden Sie zum Beispiel den folgenden Befehl, um das `client-node`-Modul zu installieren, das alle Module enthält, die Sie mit dem AWS-Verschlüsselungs-SDK for JavaScript in Node.js programmieren müssen.

```
npm install @aws-crypto/client-node
```

Verwenden Sie den folgenden Befehl, um das `client-browser`-Modul zu installieren, das alle Module enthält, die Sie mit dem AWS-Verschlüsselungs-SDK for JavaScript im Browser programmieren müssen.

```
npm install @aws-crypto/client-browser
```

Für Arbeitsbeispiele zur Verwendung des AWS-Verschlüsselungs-SDK for JavaScript, siehe die Beispiele in `example-node` und `example-browser` Module in der [aws-encryption-sdk-javascript](#) Repository auf GitHub.

Module im AWS-Verschlüsselungs-SDK for JavaScript

Die Module im AWS-Verschlüsselungs-SDK for JavaScript machen es einfach, den Code zu installieren, den Sie für Ihre Projekte benötigen.

Module für JavaScript Node.js

[client-node](#)

Enthält alle Module, die Sie benötigen, um mit dem AWS-Verschlüsselungs-SDK for JavaScript in Node.js zu programmieren.

[caching-materials-manager-Knoten](#)

Exportiert Funktionen, die die [Datenschlüssel-Caching](#)-Funktion im AWS-Verschlüsselungs-SDK for JavaScript in Node.js unterstützen.

[decrypt-node](#)

Exportiert Funktionen, die verschlüsselte Nachrichten entschlüsseln und verifizieren, die Daten und Datenströme darstellen. Im `client-node`-Modul enthalten.

[encrypt-node](#)

Exportiert Funktionen, die verschiedene Datentypen verschlüsseln und signieren. Im `client-node`-Modul enthalten.

[example-node](#)

Exportiert Arbeitsbeispiele der Programmierung mit dem AWS-Verschlüsselungs-SDK for JavaScript in Node.js. Enthält ein Beispiel für verschiedene Arten von Schlüsselbunden und verschiedene Arten von Daten.

[hkdf-node](#)

Exportiert eine [HMAC-basierte Schlüsselableitungsfunktion](#) (HKDF), die das AWS-Verschlüsselungs-SDK for JavaScript in Node.js in bestimmten Algorithmen-Paketen verwendet. Die AWS-Verschlüsselungs-SDK for JavaScript verwenden Sie im Browser die native HKDF-Funktion im WebCryptoAPI.

[integration-node](#)

Definiert Tests, die überprüfen, ob das AWS-Verschlüsselungs-SDK for JavaScript in Node.js mit anderen Sprachenimplementierungen des AWS Encryption SDK kompatibel ist.

[kms-keyring-node](#)

Exportiert Funktionen, die unterstützen AWS KMSSchlüsselbunde in Node.js.

[raw-aes-keyring-Knoten](#)

Exportiert Funktionen, die [Unformatierte AES-Schlüsselbunde](#) in Node.js unterstützen.

[raw-rsa-keyring-Knoten](#)

Exportiert Funktionen, die [Unformatierte RSA-Schlüsselbunde](#) in Node.js unterstützen.

Module für JavaScript Browser

[client-browser](#)

Enthält alle Module, die Sie benötigen, um mit dem AWS-Verschlüsselungs-SDK for JavaScript im Browser zu programmieren.

[caching-materials-manager-Browser](#)

Exportiert Funktionen, die die [Datenschlüssel-Caching](#)-Funktion für JavaScript im Browser unterstützen.

[decrypt-browser](#)

Exportiert Funktionen, die verschlüsselte Nachrichten entschlüsseln und verifizieren, die Daten und Datenströme darstellen.

[encrypt-browser](#)

Exportiert Funktionen, die verschiedene Datentypen verschlüsseln und signieren.

[example-browser](#)

Arbeitsbeispiele für die Programmierung mit dem AWS-Verschlüsselungs-SDK for JavaScript im Browser. Enthält Beispiele für verschiedene Arten von Schlüsselbunden und verschiedene Arten von Daten.

[integration-browser](#)

Definiert Tests, die überprüfen, ob das AWS-Verschlüsselungs-SDK for Java-Skript im Browser mit anderen Sprachenimplementierungen des AWS Encryption SDK kompatibel ist.

[kms-keyring-browser](#)

Exportiert Funktionen, die unterstützen [AWS KMS-Schlüsselbünde](#) Im Browser.

[raw-aes-keyring-Browser](#)

Exportiert Funktionen, die [Unformatierte AES-Schlüsselbunde](#) im Browser unterstützen.

[raw-rsa-keyring-Browser](#)

Exportiert Funktionen, die [Unformatierte RSA-Schlüsselbunde](#) im Browser unterstützen.

Module für alle Implementierungen

[cache-material](#)

Unterstützt die [Datenschlüssel-Caching](#)-Funktion. Stellt Code für die Zusammenstellung der kryptografischen Materialien bereit, die mit jedem Datenschlüssel zwischengespeichert werden.

[kms-keyring](#)

Exportiert Funktionen, die [KMS-Schlüsselbunde](#) unterstützen.

[material-management](#)

Implementiert den [Manager von kryptographischen Materialien](#) (CMM).

[raw-keyring](#)

Exportiert Funktionen, die für unformatierte AES- und RSA-Schlüsselbunde erforderlich sind.

[serialize](#)

Exportiert Funktionen, die das SDK verwendet, um seine Ausgabe zu serialisieren.

[web-crypto-backend](#)

Exportiert Funktionen, die die WebCryptoAPI im AWS-Verschlüsselungs-SDK for JavaScript im Browser.

AWS-Verschlüsselungs-SDK for JavaScript Beispiele für

Die folgenden Beispiele zeigen Ihnen, wie Sie mit dem AWS-Verschlüsselungs-SDK for JavaScript Daten verschlüsseln und entschlüsseln.

Weitere Beispiele zur Verwendung des AWS-Verschlüsselungs-SDK for JavaScript im [example-node](#) und [example-browser](#)-Module im [aws-encryption-sdk-javascript](#)-Repository auf GitHub aus. Diese Beispielmuster werden nicht installiert, wenn Sie die Module `client-browser` oder `client-node` installieren.

Die vollständigen Codebeispiele finden Sie unter: Knoten: [kms_simple.ts](#), Browser: [kms_simple.ts](#)

Themen

- [Verschlüsseln von Daten mit einem AWS KMSSchlüsselring](#)
- [Entschlüsseln von Daten mit einem AWS KMSSchlüsselring](#)

Verschlüsseln von Daten mit einem AWS KMSSchlüsselring

Das folgende Beispiel zeigt, wie Sie das AWS-Verschlüsselungs-SDK for JavaScript zum Verschlüsseln und Entschlüsseln eines kurzen Zeichenfolgen- oder Byte-Arrays verwenden.

In diesem Beispiel ist ein [AWS KMSSchlüsselring](#), eine Art Schlüsselbund, der ein AWS KMS key um Datenschlüssel zu generieren und zu verschlüsseln. Hilfe beim Erstellen eines AWS KMS key, finden Sie unter [Erstellen von Schlüsseln](#) im AWS Key Management Service Entwicklerhandbuch aus. Hilfe bei der Identifizierung des AWS KMS keys in einem AWS KMSSchlüsselbund finden Sie unter [Identifizierung AWS KMS keys in einem AWS KMS Schlüsselbund](#)


Schritt 1: Erstellen Sie den Schlüsselbund.

Erstellen eines AWS KMSSchlüsselbund für die Verschlüsselung.

Beim Verschlüsseln mit einem AWS KMSSchlüsselbund, müssen Sie ein Generatorschlüssel, das heißt ein AWS KMS key, der verwendet wird, um den Klartext-Datenschlüssel zu generieren und zu verschlüsseln. Sie können auch null oder mehr zusätzliche Schlüssel angeben, die denselben

Klartext-Datenschlüssel verschlüsseln. Der Schlüsselbund gibt den Klartext-Datenschlüssel und eine verschlüsselte Kopie dieses Datenschlüssels für jeden AWS KMS key im Schlüsselbund, einschließlich des Generatorschlüssels, zurück. Um die Daten zu entschlüsseln, müssen Sie einen der verschlüsselten Datenschlüssel entschlüsseln.

Um die AWS KMS keys für einen Verschlüsselungsschlüsselbund im AWS-Verschlüsselungs-SDK for JavaScript anzugeben, können Sie einen [beliebigen unterstützten AWS KMS-Schlüsselbezeichner](#) verwenden. In diesem Beispiel wird ein Generatorschlüssel verwendet, der durch seinen [Alias-ARN](#) identifiziert wird, und ein zusätzlicher Schlüssel, der durch einen [Schlüssel-ARN](#) identifiziert wird.

 Note

Wenn Sie vorhaben, Ihre wiederzuverwenden AWS KMS Schlüsselbund zum Entschlüsseln müssen Sie Schlüssel-ARNs verwenden, um die AWS KMS keys im Schlüsselbund.

Bevor Sie diesen Code ausführen, ersetzen Sie die AWS KMS key-Beispielbezeichner durch gültige Bezeichner. Sie müssen über die [erforderlichen Berechtigungen verfügen, um die AWS KMS keys im Schlüsselbund zu verwenden](#).

JavaScript Browser

Geben Sie zunächst Ihre Anmeldeinformationen im Browser ein. Die AWS-Verschlüsselungs-SDK for JavaScript-Beispiele verwenden die [webpack.DefinePlugin](#), das die Anmeldeinformationskonstanten durch Ihre tatsächlichen Anmeldeinformationen ersetzt. Sie können jedoch jede Methode verwenden, um Ihre Anmeldeinformationen anzugeben. Verwenden Sie dann die Anmeldeinformationen, um einen AWS KMS-Client zu erstellen.

```
declare const credentials: {accessKeyId: string, secretAccessKey:string,
  sessionToken:string }

const clientProvider = getClient(KMS, {
  credentials: {
    accessKeyId,
    secretAccessKey,
    sessionToken
  }
})
```

Geben Sie als Nächstes das AWS KMS keys für den Generatorschlüssel und den zusätzlichen Schlüssel. Erstellen Sie dann ein AWS KMS Schlüsselbund unter Verwendung des AWS KMS Kunden und der AWS KMS keys aus.

```
const generatorKeyId = 'arn:aws:kms:us-west-2:111122223333:alias/EncryptDecrypt'  
const keyIds = ['arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab']  
  
const keyring = new KmsKeyringBrowser({ clientProvider, generatorKeyId, keyIds })
```

JavaScript Node.js

```
const generatorKeyId = 'arn:aws:kms:us-west-2:111122223333:alias/EncryptDecrypt'  
const keyIds = ['arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab']  
  
const keyring = new KmsKeyringNode({ generatorKeyId, keyIds })
```

Schritt 2: Legen Sie den Verschlüsselungskontext fest.

Ein [Verschlüsselungskontext](#) ist eine Art zufälliger, nicht geheimer, zusätzlich authentifizierter Daten. Wenn Sie beim Verschlüsseln einen Verschlüsselungskontext angeben, bindet das AWS Encryption SDK den Verschlüsselungskontext kryptografisch an den Verschlüsselungstext. Daher ist zum Entschlüsseln der Daten derselbe Verschlüsselungskontext erforderlich. Die Verwendung eines Verschlüsselungskontexts ist optional, aber wir empfehlen dies als eine bewährte Methode.

Erstellen Sie ein einfaches Objekt, das die Verschlüsselungskontextpaare enthält. Der Schlüssel und der Wert in jedem Paar müssen eine Zeichenfolge sein.

JavaScript Browser

```
const context = {  
  stage: 'demo',  
  purpose: 'simple demonstration app',  
  origin: 'us-west-2'  
}
```

JavaScript Node.js

```
const context = {  
  stage: 'demo',
```

```
purpose: 'simple demonstration app',
origin: 'us-west-2'
}
```

Schritt 3: Verschlüsseln Sie die Daten.

Um die Klartextdaten zu verschlüsseln, rufen Sie die `encrypt`-Funktion auf. Übergeben Sie den AWS KMSSchlüsselbund, Klartextdaten und den Verschlüsselungskontext.

Die `encrypt`-Funktion gibt eine [verschlüsselte Nachricht](#) (`result`) zurück, die die verschlüsselten Daten, den verschlüsselten Datenschlüssel und wichtige Metadaten enthält, einschließlich des Verschlüsselungskontexts und der Signatur.

Sie können [diese verschlüsselte Nachricht entschlüsseln](#), indem Sie das AWS Encryption SDK für jede unterstützte Programmiersprache verwenden.

JavaScript Browser

```
const plaintext = new Uint8Array([1, 2, 3, 4, 5])

const { result } = await encrypt(keyring, plaintext, { encryptionContext:
  context })
```

JavaScript Node.js

```
const plaintext = 'asdf'

const { result } = await encrypt(keyring, plaintext, { encryptionContext:
  context })
```

Entschlüsseln von Daten mit einem AWS KMSSchlüsselring

Sie können das AWS-Verschlüsselungs-SDK for JavaScript verwenden, um die verschlüsselte Nachricht zu entschlüsseln und die ursprünglichen Daten wiederherzustellen.

In diesem Beispiel entschlüsseln wir die Daten, die wir im [the section called “Verschlüsseln von Daten mit einem AWS KMSSchlüsselring”](#)-Beispiel verschlüsselt haben.


Schritt 1: Erstellen Sie den Schlüsselbund.

Um die Daten zu entschlüsseln, übergeben Sie die [verschlüsselte Nachricht](#) (`result`), die die `encrypt`-Funktion zurückgegeben hat. Die verschlüsselte Nachricht enthält die verschlüsselten

Daten, die verschlüsselten Datenschlüssel und wichtige Metadaten, einschließlich des Verschlüsselungskontexts und der Signatur.

Sie müssen auch einen angeben [AWS KMSSchlüsselring](#) beim Entschlüsseln. Sie können denselben Schlüsselbund verwenden, der zum Verschlüsseln der Daten verwendet wurde, oder einen anderen Schlüsselbund. Um erfolgreich zu sein, muss mindestens ein AWS KMS key im Entschlüsselungsschlüsselbund in der Lage sein, einen der verschlüsselten Datenschlüssel in der verschlüsselten Nachricht zu entschlüsseln. Da keine Datenschlüssel generiert werden, müssen Sie keinen Generatorschlüssel in einem Entschlüsselungsschlüsselbund angeben. Wenn Sie dies tun, werden der Generatorschlüssel und zusätzliche Schlüssel auf die gleiche Weise behandelt.

So geben Sie ein [AWS KMS key](#) für einen Entschlüsselungsschlüsselring im [AWS-Verschlüsselungs-SDK for JavaScript](#) verwenden Sie die [Schlüssel-ARN](#) aus. Andernfalls wird der AWS KMS key nicht erkannt. Hilfe bei der Identifizierung des [AWS KMS keys](#) in einem [AWS KMSSchlüsselbund](#) finden Sie unter [Identifizierung AWS KMS keys in einem AWS KMS Schlüsselbund](#)

 Note

Wenn Sie denselben Schlüsselbund zum Verschlüsseln und Entschlüsseln verwenden, verwenden Sie Schlüssel-ARNs, um die AWS KMS keys im Schlüsselbund zu identifizieren.

In diesem Beispiel erstellen wir einen Schlüsselbund, der nur einen der AWS KMS keys im Verschlüsselungsschlüsselbund enthält. Ersetzen Sie vor Ausführung dieses Codes den ARN des Beispiel-Schlüssels durch einen gültigen. Sie müssen über die `kms:Decrypt`-Berechtigung für den AWS KMS key verfügen.

JavaScript Browser

Geben Sie zunächst Ihre Anmeldeinformationen im Browser ein. Die [AWS-Verschlüsselungs-SDK for JavaScript](#)-Beispiele verwenden die [webpack.DefinePlugin](#), das die Anmeldeinformationskonstanten durch Ihre tatsächlichen Anmeldeinformationen ersetzt. Sie können jedoch jede Methode verwenden, um Ihre Anmeldeinformationen anzugeben. Verwenden Sie dann die Anmeldeinformationen, um einen AWS KMS-Client zu erstellen.

```
declare const credentials: {accessKeyId: string, secretAccessKey:string,  
  sessionToken:string }
```



```
const clientProvider = getClient(KMS, {
  credentials: {
    accessKeyId,
    secretAccessKey,
    sessionToken
  }
})
```

Erstellen Sie als Nächstes eine AWS KMSSchlüsselbund unter Verwendung des AWS KMS-Client. In diesem Beispiel wird nur einer der AWS KMS keys aus dem Verschlüsselungsschlüsselbund verwendet.

```
const keyIds = ['arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab']

const keyring = new KmsKeyringBrowser({ clientProvider, keyIds })
```

JavaScript Node.js

```
const keyIds = ['arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab']

const keyring = new KmsKeyringNode({ keyIds })
```

Schritt 2: Entschlüsseln Sie die Daten.

Rufen Sie als Nächstes die `decrypt`-Funktion auf. Übergeben Sie den gerade erstellten Entschlüsselungsschlüsselbund (`keyring`) und die [verschlüsselte Nachricht](#), die die `encrypt`-Funktion zurückgegeben hat (`result`). Das AWS Encryption SDK verwendet den Schlüsselbund, um einen der verschlüsselten Datenschlüssel zu entschlüsseln. Dann verwendet es den Klartext-Datenschlüssel, um die Daten zu entschlüsseln.

Wenn der Aufruf erfolgreich ist, enthält das `plaintext`-Feld die Klartextdaten (entschlüsselt). Das `messageHeader`-Feld enthält Metadaten über den Entschlüsselungsprozess, einschließlich des Verschlüsselungskontexts, der zum Entschlüsseln der Daten verwendet wurde.

JavaScript Browser

```
const { plaintext, messageHeader } = await decrypt(keyring, result)
```

JavaScript Node.js

```
const { plaintext, messageHeader } = await decrypt(keyring, result)
```

Schritt 3: Überprüfen Sie den Verschlüsselungskontext.

Der [Verschlüsselungskontext](#), der zum Entschlüsseln der Daten verwendet wurde, ist im Nachrichten-Header (`messageHeader`) enthalten, den die `decrypt`-Funktion zurückgibt. Bevor Ihre Anwendung die Klartextdaten zurückgibt, stellen Sie sicher, dass der Verschlüsselungskontext, den Sie beim Verschlüsseln angegeben haben, im Verschlüsselungskontext enthalten ist, der bei der Entschlüsselung verwendet wurde. Eine Nichtübereinstimmung kann darauf hindeuten, dass die Daten manipuliert wurden oder dass Sie nicht den richtigen Verschlüsselungstext entschlüsselt haben.

Bei der Überprüfung des Verschlüsselungskontexts ist keine genaue Übereinstimmung erforderlich. Wenn Sie einen Verschlüsselungsalgorithmus mit Signatur verwenden, fügen Sie der [Manager von kryptographischen Materialien](#) (CMM) den öffentlichen Signaturschlüssel zum Verschlüsselungskontext hinzu, bevor die Nachricht verschlüsselt wird. Aber alle Verschlüsselungskontextpaare, die Sie übermittelt haben, sollten in den zurückgegebenen Verschlüsselungskontext aufgenommen werden.

Holen Sie sich zuerst den Verschlüsselungskontext aus dem Nachrichten-Header. Stellen Sie dann sicher, dass jedes Schlüssel-Wert-Paar im ursprünglichen Verschlüsselungskontext (`context`) mit einem Schlüssel-Wert-Paar im zurückgegebenen Verschlüsselungskontext (`encryptionContext`) übereinstimmt.

JavaScript Browser

```
const { encryptionContext } = messageHeader

Object
  .entries(context)
  .forEach(([key, value]) => {
    if (encryptionContext[key] !== value) throw new Error('Encryption Context
    does not match expected values')
  })
```

JavaScript Node.js

```
const { encryptionContext } = messageHeader
```

```
Object
  .entries(context)
  .forEach(([key, value]) => {
    if (encryptionContext[key] !== value) throw new Error('Encryption Context
    does not match expected values')
  })
```

Wenn die Verschlüsselungskontext-Prüfung erfolgreich ist, können Sie die Klartextdaten zurückgeben.

AWS-Verschlüsselungs-SDK for Python

In diesem Thema wird erklärt, wie das AWS-Verschlüsselungs-SDK for Python installiert und verwendet wird. Einzelheiten zur Programmierung mit dem AWS-Verschlüsselungs-SDK for Python finden Sie im [aws-encryption-sdk-python](#) Repository unter GitHub. Eine API-Dokumentation finden Sie in [Read the Docs](#).

Themen

- [Voraussetzungen](#)
- [Installation](#)
- [Beispiel-Code für AWS-Verschlüsselungs-SDK for Python](#)

Voraussetzungen

Stellen Sie vor der Installation von sicher AWS-Verschlüsselungs-SDK for Python, dass Sie die folgenden Voraussetzungen erfüllen.

Eine unterstützte Version von Python

Python 3.8 oder höher ist für die AWS-Verschlüsselungs-SDK for Python Versionen 3.2.0 und höher erforderlich.

Frühere Versionen von AWS Encryption SDK unterstützen Python 2.7 und Python 3.4 und höher, wir empfehlen jedoch, dass Sie die neueste Version von verwenden AWS Encryption SDK.

Weitere Informationen zum Download von Python finden Sie unter [Python-Downloads](#).

Das pip-Installationstool for Python

`pip` ist in Python 3.6 und späteren Versionen enthalten, obwohl Sie es möglicherweise aktualisieren möchten. Weitere Informationen zum Aktualisieren oder Installieren `pip` finden Sie in der `pip` Dokumentation unter [Installation](#).

Installation

Installieren Sie die neueste Version von AWS-Verschlüsselungs-SDK for Python.

Note

Alle AWS-Verschlüsselungs-SDK for Python Versionen vor 3.0.0 befinden sich in der [end-of-supportPhase](#).

Sie können sicher von Version 2.0 aus aktualisieren. x und höher auf die neueste Version von AWS Encryption SDK ohne Code- oder Datenänderungen. In Version 2.0 wurden jedoch [neue Sicherheitsfunktionen](#) eingeführt. x sind nicht abwärtskompatibel. Um von Versionen vor 1.7 zu aktualisieren. x auf Version 2.0. x und höher, Sie müssen zuerst auf die neueste Version 1 aktualisieren. x-Version von AWS Encryption SDK. Details hierzu finden Sie unter [Migrieren Sie Ihre AWS Encryption SDK](#).

Verwenden Sie, `pip` um die zu installieren AWS-Verschlüsselungs-SDK for Python, wie in den folgenden Beispielen gezeigt.

Installieren der neuesten Version

```
pip install aws-encryption-sdk
```

Weitere Informationen zur Verwendung von `pip` für die Installation und die Aktualisierung von Paketen finden Sie unter [Pakete installieren](#).

Das AWS-Verschlüsselungs-SDK for Python erfordert die [Kryptografie-Bibliothek](#) (`pyca/cryptography`) auf allen Plattformen. Alle Versionen von installieren und erstellen die Bibliothek `pip` automatisch unter Windows. `cryptography pip8.1` und höher werden automatisch `cryptography` auf Linux installiert und erstellt. Wenn Sie eine frühere Version von verwenden `pip` und Ihre Linux-Umgebung nicht über die Tools verfügt, die zum Erstellen der `cryptography` Bibliothek erforderlich sind,

müssen Sie sie installieren. Weitere Informationen finden Sie unter [Erstellen von Kryptographie unter Linux](#).

In den Versionen 1.10.0 und 2.5.0 liegt die AWS-Verschlüsselungs-SDK for Python [Kryptografieabhängigkeit](#) zwischen 2.5.0 und 3.3.2. Andere Versionen von AWS-Verschlüsselungs-SDK for Python installieren die neueste Version der Kryptografie. Wenn Sie eine neuere Version der Kryptografie als 3.3.2 benötigen, empfehlen wir Ihnen, die neueste Hauptversion von zu verwenden. AWS-Verschlüsselungs-SDK for Python

Die neueste Entwicklungsversion von finden Sie im AWS-Verschlüsselungs-SDK for Python [aws-encryption-sdk-python](#) Repository unter. GitHub

Schauen Sie sich nach der AWS-Verschlüsselungs-SDK for Python Installation von zunächst den [Python-Beispielcode](#) in diesem Handbuch an.

Beispiel-Code für AWS-Verschlüsselungs-SDK for Python

Die folgenden Beispiele zeigen Ihnen, wie Sie mit dem AWS-Verschlüsselungs-SDK for Python Daten verschlüsseln und entschlüsseln.

Die Beispiele in diesem Abschnitt zeigen, wie Sie verwenden können [Version 2.0.x](#) und später von der AWS-Verschlüsselungs-SDK for Python aus. Für Beispiele, die frühere Versionen verwenden, finden Sie Ihre Version im [Versionen](#)-Liste der [aws-encryption-sdk-python](#) Repository auf GitHub aus.

Themen

- [Verschlüsseln und Entschlüsseln von Zeichenfolgen](#)
- [Verschlüsseln und Entschlüsseln von Byte-Streams](#)
- [Verschlüsseln und Entschlüsseln von Byte-Streams mit mehreren Masterschlüsselanbietern](#)
- [Verwenden von Datenschlüssel-Caching zum Verschlüsseln von Nachrichten](#)

Verschlüsseln und Entschlüsseln von Zeichenfolgen

Das folgende Beispiel zeigt Ihnen, wie Sie mit dem AWS Encryption SDK Zeichenfolgen verschlüsseln und entschlüsseln. In diesem Beispiel verwenden wir einen AWS KMS key in [AWS Key Management Service \(AWS KMS\)](#) als Masterschlüssel.

Beim Verschlüsseln wird der `StrictAwsKmsMasterKeyProvider`-Konstruktor verwendet eine Schlüssel-ID, einen Schlüssel-ARN, einen Aliasnamen oder einen Alias-ARN. Beim Entschlüsseln

ist [esbenötigt einen Schlüssel-ARN](#) aus. In diesem Fall, weil der `keyArn`-Parameter zum Verschlüsseln und Entschlüsseln verwendet, sein Wert muss ein Schlüssel-ARN sein. Weitere Informationen über IDs für AWS KMS Schlüssel, siehe [Schlüsselkennungen](#) im AWS Key Management Service Entwicklerhandbuch.

```
# Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You
# may not use this file except in compliance with the License. A copy of
# the License is located at
#
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is
# distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF
# ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.
"""Example showing basic encryption and decryption of a value already in memory."""
import aws_encryption_sdk
from aws_encryption_sdk import CommitmentPolicy

def cycle_string(key_arn, source_plaintext, botocore_session=None):
    """Encrypts and then decrypts a string under an &KMS; key.

    :param str key_arn: Amazon Resource Name (ARN) of the &KMS; key
    :param bytes source_plaintext: Data to encrypt
    :param botocore_session: existing botocore session instance
    :type botocore_session: botocore.session.Session
    """
    # Set up an encryption client with an explicit commitment policy. If you do not
    explicitly choose a
    # commitment policy, REQUIRE_ENCRYPT_REQUIRE_DECRYPT is used by default.
    client =
aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT)

    # Create an AWS KMS master key provider
    kms_kwargs = dict(key_ids=[key_arn])
    if botocore_session is not None:
        kms_kwargs["botocore_session"] = botocore_session
    master_key_provider =
aws_encryption_sdk.StrictAwsKmsMasterKeyProvider(**kms_kwargs)
```

```

# Encrypt the plaintext source data
ciphertext, encryptor_header = client.encrypt(source=source_plaintext,
key_provider=master_key_provider)

# Decrypt the ciphertext
cycled_plaintext, decrypted_header = client.decrypt(source=ciphertext,
key_provider=master_key_provider)

# Verify that the "cycled" (encrypted, then decrypted) plaintext is identical to
the source plaintext
assert cycled_plaintext == source_plaintext

# Verify that the encryption context used in the decrypt operation includes all key
pairs from
# the encrypt operation. (The SDK can add pairs, so don't require an exact match.)
#
# In production, always use a meaningful encryption context. In this sample, we
omit the
# encryption context (no key pairs).
assert all(
    pair in decrypted_header.encryption_context.items() for pair in
encryptor_header.encryption_context.items()
)

```

Verschlüsseln und Entschlüsseln von Byte-Streams

Das folgende Beispiel zeigt Ihnen, wie Sie mit dem AWS Encryption SDK Byte-Streams verschlüsseln und entschlüsseln. Dieses Beispiel verwendet nicht AWS. Es verwendet einen statischen, flüchtigen Masterschlüsselanbieter.

Bei der Verschlüsselung verwendet dieses Beispiel eine alternative Algorithmus-Suite ohne [digitale Signaturen](#) (AES_256_GCM_HKDF_SHA512_COMMIT_KEY) enthalten. Diese Algorithmus-Suite ist geeignet, wenn Benutzern, die Daten verschlüsseln und entschlüsseln, gleichermaßen vertraut wird. Beim Entschlüsseln verwendet das Beispiel die `decrypt-unsignedStreaming`-Modus, der fehlschlägt, wenn er auf signierten Chiffretext stößt. Die `decrypt-unsignedStreaming`-Modus wird in eingeführt AWS Encryption SDK Versionen 1.9.x und 2.2.x aus.

```

# Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You
# may not use this file except in compliance with the License. A copy of
# the License is located at

```

```

#
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is
# distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF
# ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.
"""Example showing creation and use of a RawMasterKeyProvider."""
import filecmp
import os

import aws_encryption_sdk
from aws_encryption_sdk.identifiers import Algorithm, CommitmentPolicy,
    EncryptionKeyType, WrappingAlgorithm
from aws_encryption_sdk.internal.crypto.wrapping_keys import WrappingKey
from aws_encryption_sdk.key_providers.raw import RawMasterKeyProvider

class StaticRandomMasterKeyProvider(RawMasterKeyProvider):
    """Randomly generates 256-bit keys for each unique key ID."""

    provider_id = "static-random"

    def __init__(self, **kwargs): # pylint: disable=unused-argument
        """Initialize empty map of keys."""
        self._static_keys = {}

    def _get_raw_key(self, key_id):
        """Returns a static, randomly-generated symmetric key for the specified key
ID.

:param str key_id: Key ID
:returns: Wrapping key that contains the specified static key
:rtype: :class:`aws_encryption_sdk.internal.crypto.WrappingKey`
"""
        try:
            static_key = self._static_keys[key_id]
        except KeyError:
            static_key = os.urandom(32)
            self._static_keys[key_id] = static_key
        return WrappingKey(
            wrapping_algorithm=WrappingAlgorithm.AES_256_GCM_IV12_TAG16_NO_PADDING,
            wrapping_key=static_key,
            wrapping_key_type=EncryptionKeyType.SYMMETRIC,
        )

```



```
def cycle_file(source_plaintext_filename):
    """Encrypts and then decrypts a file under a custom static master key provider.
    :param str source_plaintext_filename: Filename of file to encrypt
    """
    # Set up an encryption client with an explicit commitment policy. Note that if you
    do not explicitly choose a
    # commitment policy, REQUIRE_ENCRYPT_REQUIRE_DECRYPT is used by default.
    client =
aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQU

    # Create a static random master key provider
    key_id = os.urandom(8)
    master_key_provider = StaticRandomMasterKeyProvider()
    master_key_provider.add_master_key(key_id)

    ciphertext_filename = source_plaintext_filename + ".encrypted"
    cycled_plaintext_filename = source_plaintext_filename + ".decrypted"

    # Encrypt the plaintext source data
    # We can use an unsigned algorithm suite here under the assumption that the
    contexts that encrypt
    # and decrypt are equally trusted.
    with open(source_plaintext_filename, "rb") as plaintext, open(ciphertext_filename,
"wb") as ciphertext:
        with client.stream(
            algorithm=Algorithm.AES_256_GCM_HKDF_SHA512_COMMIT_KEY,
            mode="e",
            source=plaintext,
            key_provider=master_key_provider,
        ) as encryptor:
            for chunk in encryptor:
                ciphertext.write(chunk)

    # Decrypt the ciphertext
    # We can use the recommended "decrypt-unsigned" streaming mode since we encrypted
    with an unsigned algorithm suite.
    with open(ciphertext_filename, "rb") as ciphertext, open(cycled_plaintext_filename,
"wb") as plaintext:
        with client.stream(mode="decrypt-unsigned", source=ciphertext,
key_provider=master_key_provider) as decryptor:
            for chunk in decryptor:
                plaintext.write(chunk)
```

```
# Verify that the "cycled" (encrypted, then decrypted) plaintext is identical to
the source
# plaintext
assert filecmp.cmp(source_plaintext_filename, cycled_plaintext_filename)

# Verify that the encryption context used in the decrypt operation includes all key
pairs from
# the encrypt operation
#
# In production, always use a meaningful encryption context. In this sample, we
omit the
# encryption context (no key pairs).
assert all(
    pair in decryptor.header.encryption_context.items() for pair in
encryptor.header.encryption_context.items()
)
return ciphertext_filename, cycled_plaintext_filename
```

Verschlüsseln und Entschlüsseln von Byte-Streams mit mehreren Masterschlüsselanbietern

Das folgende Beispiel zeigt, wie Sie das AWS Encryption SDK mit mehr als einem Masterschlüsselanbieter verwenden. Die Verwendung mehrerer Masterschlüsselanbieter schafft Redundanz für den Fall, dass ein Masterschlüsselanbieter für die Entschlüsselung nicht verfügbar ist. In diesem Beispiel verwenden wir einen AWS KMS Key und ein RSA-Schlüsselpaar als Masterschlüssel.

Dieses Beispiel wird mit dem [Standard-Algorithmen-Paket](#) umfasst einen [digitale Signatur](#) aus. Beim Streamen wird der AWS Encryption SDK gibt Klartext nach Integritätsprüfungen frei, aber bevor es die digitale Signatur überprüft hat. Um die Verwendung des Klartextes zu vermeiden, bis die Signatur überprüft wird, puffert dieses Beispiel den Klartext und schreibt ihn nur auf die Festplatte, wenn die Entschlüsselung und Überprüfung abgeschlossen ist.

```
# Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You
# may not use this file except in compliance with the License. A copy of
# the License is located at
#
# http://aws.amazon.com/apache2.0/
#
```

```
# or in the "license" file accompanying this file. This file is
# distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF
# ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.
"""Example showing creation of a RawMasterKeyProvider, how to use multiple
master key providers to encrypt, and demonstrating that each master key
provider can then be used independently to decrypt the same encrypted message.
"""
import filecmp
import os

from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa

import aws_encryption_sdk
from aws_encryption_sdk.identifiers import CommitmentPolicy, EncryptionKeyType,
    WrappingAlgorithm
from aws_encryption_sdk.internal.crypto.wrapping_keys import WrappingKey
from aws_encryption_sdk.key_providers.raw import RawMasterKeyProvider

class StaticRandomMasterKeyProvider(RawMasterKeyProvider):
    """Randomly generates and provides 4096-bit RSA keys consistently per unique key
    id."""

    provider_id = "static-random"

    def __init__(self, **kwargs): # pylint: disable=unused-argument
        """Initialize empty map of keys."""
        self._static_keys = {}

    def _get_raw_key(self, key_id):
        """Retrieves a static, randomly generated, RSA key for the specified key id.

        :param str key_id: User-defined ID for the static key
        :returns: Wrapping key that contains the specified static key
        :rtype: :class:`aws_encryption_sdk.internal.crypto.WrappingKey`
        """
        try:
            static_key = self._static_keys[key_id]
        except KeyError:
            private_key = rsa.generate_private_key(public_exponent=65537,
            key_size=4096, backend=default_backend())
```

```

        static_key = private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.PKCS8,
            encryption_algorithm=serialization.NoEncryption(),
        )
        self._static_keys[key_id] = static_key
    return WrappingKey(
        wrapping_algorithm=WrappingAlgorithm.RSA_OAEP_SHA1_MGF1,
        wrapping_key=static_key,
        wrapping_key_type=EncryptionKeyType.PRIVATE,
    )

```

```

def cycle_file(key_arn, source_plaintext_filename, botocore_session=None):
    """Encrypts and then decrypts a file using an AWS KMS master key provider and a
    custom static master
    key provider. Both master key providers are used to encrypt the plaintext file, so
    either one alone
    can decrypt it.

    :param str key_arn: Amazon Resource Name (ARN) of the &KMS; key
    (http://docs.aws.amazon.com/kms/latest/developerguide/viewing-keys.html)
    :param str source_plaintext_filename: Filename of file to encrypt
    :param botocore_session: existing botocore session instance
    :type botocore_session: botocore.session.Session
    """
    # "Cycled" means encrypted and then decrypted
    ciphertext_filename = source_plaintext_filename + ".encrypted"
    cycled_kms_plaintext_filename = source_plaintext_filename + ".kms.decrypted"
    cycled_static_plaintext_filename = source_plaintext_filename + ".static.decrypted"

    # Set up an encryption client with an explicit commitment policy. Note that if you
    do not explicitly choose a
    # commitment policy, REQUIRE_ENCRYPT_REQUIRE_DECRYPT is used by default.
    client =
aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQU

    # Create an AWS KMS master key provider
    kms_kwargs = dict(key_ids=[key_arn])
    if botocore_session is not None:
        kms_kwargs["botocore_session"] = botocore_session
    kms_master_key_provider =
aws_encryption_sdk.StrictAwsKmsMasterKeyProvider(**kms_kwargs)

```

```
# Create a static master key provider and add a master key to it
static_key_id = os.urandom(8)
static_master_key_provider = StaticRandomMasterKeyProvider()
static_master_key_provider.add_master_key(static_key_id)

# Add the static master key provider to the AWS KMS master key provider
# The resulting master key provider uses AWS KMS master keys to generate (and
encrypt)
# data keys and static master keys to create an additional encrypted copy of each
data key.
kms_master_key_provider.add_master_key_provider(static_master_key_provider)

# Encrypt plaintext with both AWS KMS and static master keys
with open(source_plaintext_filename, "rb") as plaintext, open(ciphertext_filename,
"wb") as ciphertext:
    with client.stream(source=plaintext, mode="e",
key_provider=kms_master_key_provider) as encryptor:
        for chunk in encryptor:
            ciphertext.write(chunk)

# Decrypt the ciphertext with only the AWS KMS master key
# Buffer the data in memory before writing to disk. This ensures verification of the
digital signature before returning plaintext.
with open(ciphertext_filename, "rb") as ciphertext,
open(cycled_kms_plaintext_filename, "wb") as plaintext:
    with client.stream(
        source=ciphertext, mode="d",
key_provider=aws_encryption_sdk.StrictAwsKmsMasterKeyProvider(**kms_kwargs)
    ) as kms_decryptor:
        plaintext.write(kms_decryptor.read())

# Decrypt the ciphertext with only the static master key
# Buffer the data in memory before writing to disk to ensure verification of the
signature before returning plaintext.
with open(ciphertext_filename, "rb") as ciphertext,
open(cycled_static_plaintext_filename, "wb") as plaintext:
    with client.stream(source=ciphertext, mode="d",
key_provider=static_master_key_provider) as static_decryptor:
        plaintext.write(static_decryptor.read())

# Verify that the "cycled" (encrypted, then decrypted) plaintext is identical to
the source plaintext
assert filecmp.cmp(source_plaintext_filename, cycled_kms_plaintext_filename)
assert filecmp.cmp(source_plaintext_filename, cycled_static_plaintext_filename)
```

```
# Verify that the encryption context in the decrypt operation includes all key
pairs from the
# encrypt operation.
#
# In production, always use a meaningful encryption context. In this sample, we
omit the
# encryption context (no key pairs).
assert all(
    pair in kms_decryptor.header.encryption_context.items() for pair in
encryptor.header.encryption_context.items()
)
assert all(
    pair in static_decryptor.header.encryption_context.items()
    for pair in encryptor.header.encryption_context.items()
)
return (ciphertext_filename, cycled_kms_plaintext_filename,
        cycled_static_plaintext_filename)
```

Verwenden von Datenschlüssel-Caching zum Verschlüsseln von Nachrichten

Das folgende Beispiel zeigt, wie Sie das [Datenschlüssel-Caching](#) im AWS-Verschlüsselungs-SDK für Python verwenden. Es wurde entwickelt, um Ihnen zu zeigen, wie Sie eine Instanz von [Lokaler Cache](#) (LocalCryptoMaterialsCache) mit dem erforderlichen Kapazitätswert und einer Instanz des [Caching von Managern von krypto](#) (CMM zwischenspeichern) mit [Sicherheitsschwellenwerte Cache](#) aus.

In diesem sehr einfachen Beispiel wird eine Funktion erstellt, die eine feste Zeichenfolge verschlüsselt. Sie können damit ein AWS KMS key, die erforderliche Cache-Größe (Kapazität) und ein Höchstalter. Ein komplexeres, reales Beispiel des Datenschlüssel-Cachings finden Sie unter [Beispielcode für das Zwischenspeichern von Datenschlüsseln](#).

In diesem Beispiel wird auch ein [Verschlüsselungskontext](#) als zusätzlich authentifizierte Daten verwendet, auch wenn dies optional ist. Wenn Sie Daten entschlüsseln, die mit einem Verschlüsselungskontext verschlüsselt wurden, stellen Sie sicher, dass Ihre Anwendung überprüft, dass der Verschlüsselungskontext der erwartete ist, bevor Sie die Klartextdaten an den Aufrufer zurückgeben. Ein Verschlüsselungskontext ist eine bewährte Methode jeder Verschlüsselungs- oder Entschlüsselungsoperation, er spielt aber eine spezielle Rolle im Datenschlüssel-Caching. Details dazu finden Sie unter [.Verschlüsselungskontext: Wie Cache-Einträge ausgewählt werden](#) aus.

```
# Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
#
# Licensed under the Apache License, Version 2.0 (the "License"). You
# may not use this file except in compliance with the License. A copy of
# the License is located at
#
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is
# distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF
# ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.
"""Example of encryption with data key caching."""
import aws_encryption_sdk
from aws_encryption_sdk import CommitmentPolicy

def encrypt_with_caching(kms_key_arn, max_age_in_cache, cache_capacity):
    """Encrypts a string using an &KMS; key and data key caching.

    :param str kms_key_arn: Amazon Resource Name (ARN) of the &KMS; key
    :param float max_age_in_cache: Maximum time in seconds that a cached entry can be
    used
    :param int cache_capacity: Maximum number of entries to retain in cache at once
    """
    # Data to be encrypted
    my_data = "My plaintext data"

    # Security thresholds
    # Max messages (or max bytes per) data key are optional
    MAX_ENTRY_MESSAGES = 100

    # Create an encryption context
    encryption_context = {"purpose": "test"}

    # Set up an encryption client with an explicit commitment policy. Note that if you
    do not explicitly choose a
    # commitment policy, REQUIRE_ENCRYPT_REQUIRE_DECRYPT is used by default.
    client =
aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT)

    # Create a master key provider for the &KMS; key
    key_provider =
aws_encryption_sdk.StrictAwsKmsMasterKeyProvider(key_ids=[kms_key_arn])
```

```
# Create a local cache
cache = aws_encryption_sdk.LocalCryptoMaterialsCache(cache_capacity)

# Create a caching CMM
caching_cmm = aws_encryption_sdk.CachingCryptoMaterialsManager(
    master_key_provider=key_provider,
    cache=cache,
    max_age=max_age_in_cache,
    max_messages_encrypted=MAX_ENTRY_MESSAGES,
)

# When the call to encrypt data specifies a caching CMM,
# the encryption operation uses the data key cache specified
# in the caching CMM
encrypted_message, _header = client.encrypt(
    source=my_data, materials_manager=caching_cmm,
    encryption_context=encryption_context
)

return encrypted_message
```

AWS Encryption SDK-Befehlszeilenschnittstelle

Die AWS Encryption SDK-Befehlszeilenschnittstelle (AWS Encryption CLI) ermöglicht die Verwendung der AWS Encryption SDK um Daten interaktiv in der Befehlszeile und in Skripten zu verschlüsseln und entschlüsseln. Sie benötigen keine Kryptographie- oder Programmierkenntnisse.

Note

Versionen des AWS Verschlüsselungs-CLIs vor 4.0.0 befinden sich im [end-of-supportPhase](#). Sie können sicher von Version 2.1 aktualisieren und später zur neuesten Version des AWS Verschlüsselungs-CLI ohne Code- oder Datenänderungen. Allerdings [neue Sicherheitsfunktionen](#) eingeführt in Version 2.1 sind nicht abwärtskompatibel. Um von Version 1.7 zu aktualisieren oder früher müssen Sie zuerst auf die neueste Version 1 aktualisieren. Version des AWS Verschlüsselung CLI. Details hierzu finden Sie unter [Migrieren Sie Ihre AWS Encryption SDK](#).

Neue Sicherheitsfunktionen wurden ursprünglich in veröffentlicht AWS Encryption CLI Version 1.7 und 2.0.x. Allerdings AWS Verschlüsselung CLI Version 1.8 ersetzt

Version 1.7.x und AWS Verschlüsselung CLI 2.1.x ersetzt 2.0.x. Details dazu finden Sie im entsprechenden [Sicherheitsberatung](#) in der [aws-encryption-sdk-cli](#) Repository auf GitHub.

Wie alle Implementierungen der AWS Encryption SDK, das AWS Encryption CLI bietet erweiterte Datenschutzfunktionen. Dazu zählen [Envelope](#)-, zusätzliche authentifizierte Daten (AAD) und sicherer, authentifizierter, symmetrischer Schlüssel [Algorithmus-Suiten](#), wie 256-Bit-AES-GCM mit Schlüsselableitung [entscheidendes Engagement](#) und unterschreiben.

Die AWS Encryption CLI basiert auf [AWS-Verschlüsselungs-SDK for Python](#) und wird unter Linux, macOS und Windows unterstützt. Sie können Befehle und Skripts zum Ver- und Entschlüsseln Ihrer Daten in Ihrer bevorzugten Shell unter Linux oder macOS, in einem Eingabeaufforderungsfenster (cmd.exe) unter Windows und in einem PowerShell Konsole auf jedem System.

Alle sprachspezifischen Implementierungen des AWS Encryption SDK, einschließlich der AWS Verschlüsselungs-CLI sind interoperabel. Beispielsweise können Sie Daten verschlüsseln mit dem [AWS-Verschlüsselungs-SDK for Java](#) und entschlüssele es mit dem AWS Verschlüsselung CLI.

In diesem Thema wird die AWS Encryption CLI erklärt, wie es installiert und verwendet wird, und bietet mehrere Beispiele, die Ihnen den Einstieg erleichtern. Für einen schnellen Start siehe [So verschlüsseln und entschlüsseln Sie Ihre Daten mit dem AWS Verschlüsselung CLI](#) in der AWS Sicherheits-Blog. Weitere Informationen finden Sie unter [Lesen Sie die Dokumente](#), und entwickeln Sie gemeinsam mit uns die AWS Verschlüsselungs-CLI im [aws-encryption-sdk-cli](#) Repository auf GitHub.

Leistung

Die AWS Encryption CLI basiert auf [AWS-Verschlüsselungs-SDK for Python](#). Jedes Mal, wenn Sie die CLI ausführen, starten Sie eine neue Instance der Python-Laufzeitumgebung. Um die Leistung zu verbessern, verwenden Sie nach Möglichkeit einen einzigen Befehl anstelle einer Reihe unabhängiger Befehle. Führen Sie beispielsweise einen Befehl aus, der die Dateien in einem Verzeichnis rekursiv verarbeitet, anstatt separate Befehle für jede Datei auszuführen.

Themen

- [Installation der AWS Encryption SDK Befehlszeilenschnittstelle](#)
- [So verwenden Sie die AWS Encryption CLI](#)
- [Beispiele für die AWS CLI für Verschlüsselung](#)
- [AWS Encryption SDK CLI – Syntax und Parameterreferenz](#)

- [Versionen des AWS Verschlüsselung CLI](#)

Installation der AWS Encryption SDK Befehlszeilenschnittstelle

In diesem Thema wird erklärt, wie die AWS Encryption CLI installiert wird. Ausführliche Informationen finden Sie im [aws-encryption-sdk-cli](#) Repository unter GitHub und in [der Dokumentation](#).

Themen

- [Installieren der Voraussetzungen](#)
- [Installation und Aktualisierung der AWS Encryption CLI](#)

Installieren der Voraussetzungen

Die AWS Encryption CLI basiert auf dem AWS-Verschlüsselungs-SDK for Python. Um die AWS Encryption CLI zu installieren, benötigen Sie Python und pip das Python-Paketverwaltungstool. Python und pip stehen auf allen unterstützten Plattformen zur Verfügung.

Installieren Sie die folgenden Voraussetzungen, bevor Sie die AWS Encryption CLI installieren:

Python

Python 3.8 oder höher ist für die AWS Encryption CLI Versionen 4.2.0 und höher erforderlich.

Frühere Versionen der AWS Encryption CLI unterstützen Python 2.7 und 3.4 und höher, wir empfehlen jedoch, die neueste Version der AWS Encryption CLI zu verwenden.

Python ist in den meisten Linux- und macOS-Installationen enthalten, Sie müssen jedoch auf Python 3.6 oder höher aktualisieren. Wir empfehlen Ihnen, die neueste Version von Python zu verwenden. Unter Windows müssen Sie Python installieren; es ist standardmäßig nicht installiert. Informationen zum Herunterladen und Installieren von Python finden Sie unter [Python-Downloads](#).

Um festzustellen, ob Python installiert ist, geben Sie in der Befehlszeile Folgendes ein.

```
python
```

Um die Python-Version zu überprüfen, verwenden Sie den Parameter `-V` (Großbuchstabe V).

```
python -V
```

Unter Windows fügen Sie nach der Installation von Python den Pfad zur Python .exe Datei zum Wert der Umgebungsvariablen Path hinzu.

Standardmäßig ist Python im Verzeichnis All Users oder in einem Benutzerprofilverzeichnis (\$home oder %userprofile%) im Unterverzeichnis AppData\Local\Programs\Python installiert. Um den Speicherort der Datei Python .exe auf Ihrem System zu finden, überprüfen Sie einen der folgenden Registrierungsschlüssel. Sie können es verwenden PowerShell , um die Registrierung zu durchsuchen.

```
PS C:\> dir HKLM:\Software\Python\PythonCore\version\InstallPath
# -or-
PS C:\> dir HKCU:\Software\Python\PythonCore\version\InstallPath
```

pip

pip ist der Python-Paketmanager. Um die AWS Encryption CLI und ihre Abhängigkeiten zu installieren, benötigen Sie pip 8.1 oder höher. Informationen zum Installieren und Aktualisieren von pip finden Sie unter [Installation](#) in der pip-Dokumentation.

Bei Linux-Installationen können pip Versionen vor 8.1 die Kryptografiebibliothek, die die AWS Encryption CLI benötigt, nicht erstellen. Wenn Sie Ihre pip Version nicht aktualisieren möchten, können Sie die Build-Tools separat installieren. Weitere Informationen finden Sie unter [Kryptographie unter Linux](#).

AWS Command Line Interface

Das AWS Command Line Interface (AWS CLI) ist nur erforderlich, wenn Sie AWS KMS keys in AWS Key Management Service (AWS KMS) mit der AWS Encryption CLI verwenden. Wenn Sie einen anderen [Hauptschlüsselanbieter](#) verwenden, AWS CLI ist nicht erforderlich.

Für die Verwendung AWS KMS keys mit der AWS Encryption CLI müssen Sie den [installieren](#) und [konfigurieren](#) AWS CLI. Die Konfiguration stellt die Anmeldeinformationen, die Sie zur Authentifizierung verwenden, für die AWS Encryption CLI zur AWS KMS Verfügung.

Installation und Aktualisierung der AWS Encryption CLI

Installieren Sie die neueste Version der AWS Encryption CLI. Wenn Sie die AWS Encryption CLI pip zur Installation verwenden, werden automatisch die Bibliotheken installiert, die die CLI benötigt, einschließlich der [AWS-Verschlüsselungs-SDK for PythonPython-Kryptografiebibliothek](#) und der [AWS SDK for Python \(Boto3\)](#).

Note

Versionen der AWS Encryption CLI vor 4.0.0 befinden sich in der [end-of-supportPhase](#). Sie können problemlos von Version 2.1 aus aktualisieren. x und höher auf die neueste Version der AWS Encryption CLI ohne Code- oder Datenänderungen. In Version 2.1 wurden jedoch [neue Sicherheitsfunktionen](#) eingeführt. x sind nicht abwärtskompatibel. Um von Version 1.7 zu aktualisieren. x oder früher, Sie müssen zuerst auf die neueste Version 1 aktualisieren. x-Version der AWS Encryption CLI. Details hierzu finden Sie unter [Migrieren Sie IhreAWS Encryption SDK](#).

Neue Sicherheitsfunktionen wurden ursprünglich in den AWS Encryption CLI Versionen 1.7 veröffentlicht. x und 2.0. x. Allerdings AWS Encryption CLI Version 1.8. x ersetzt Version 1.7. x und AWS Encryption CLI 2.1. x ersetzt 2.0. x. Einzelheiten finden Sie in der entsprechenden [Sicherheitsempfehlung](#) im [aws-encryption-sdk-cli](#)Repository unter GitHub.

Um die neueste Version der AWS Encryption CLI zu installieren

```
pip install aws-encryption-sdk-cli
```

Um auf die neueste Version der AWS Encryption CLI zu aktualisieren

```
pip install --upgrade aws-encryption-sdk-cli
```

Um die Versionsnummern Ihrer AWS Encryption CLI zu finden und AWS Encryption SDK

```
aws-encryption-cli --version
```

In der Ausgabe werden die Versionsnummern beider Bibliotheken aufgeführt.

```
aws-encryption-sdk-cli/2.1.0 aws-encryption-sdk/2.0.0
```

Um auf die neueste Version der AWS Encryption CLI zu aktualisieren

```
pip install --upgrade aws-encryption-sdk-cli
```

Durch die Installation der AWS Encryption CLI wird auch die neueste Version von installiert AWS SDK for Python (Boto3), sofern sie nicht bereits installiert ist. Wenn Boto3 installiert ist, überprüft das Installationsprogramm die Boto3-Version und aktualisiert sie bei Bedarf.

Um Ihre installierte Version von Boto3 zu finden

```
pip show boto3
```

Um auf die neueste Version von Boto3 zu aktualisieren

```
pip install --upgrade boto3
```

Informationen zur Installation der Version der AWS Encryption CLI, die sich derzeit in der Entwicklung befindet, finden Sie im [aws-encryption-sdk-cli](#) Repository unter GitHub.

Weitere Informationen zur Verwendung von `pip` für die Installation und die Aktualisierung von Python-Paketen finden Sie in der [pip-Dokumentation](#).

So verwenden Sie die AWS Encryption CLI

In diesem Thema wird erklärt, wie die Parameter in der AWS Encryption CLI verwendet werden. Beispiele finden Sie unter [Beispiele für die AWS CLI für Verschlüsselung](#). Eine vollständige Dokumentation finden Sie in [Read the Docs](#). Die in diesen Beispielen gezeigte Syntax bezieht sich auf AWS Encryption CLI Version 2.1. x und später.

Note

[Versionen der AWS Encryption CLI vor 4.0.0 befinden sich in der E-Phase. nd-of-support](#)

Sie können problemlos von Version 2.1 aus aktualisieren. x und höher auf die neueste Version der AWS Encryption CLI ohne Code- oder Datenänderungen. In Version 2.1 wurden jedoch [neue Sicherheitsfunktionen](#) eingeführt. x sind nicht abwärtskompatibel. Um von Version 1.7 zu aktualisieren. x oder früher, Sie müssen zuerst auf die neueste Version 1 aktualisieren. x-Version der AWS Verschlüsselungs-CLI. Details hierzu finden Sie unter [Migrieren Sie Ihre AWS Encryption SDK](#).

Neue Sicherheitsfunktionen wurden ursprünglich in den Versionen 1.7 der AWS Encryption CLI veröffentlicht. x und 2.0. x. Allerdings AWS Encryption CLI Version 1.8. x ersetzt Version

1.7. x und AWS Verschlüsselungs-CLI 2.1. x ersetzt 2.0. x. Einzelheiten finden Sie in der entsprechenden [Sicherheitsempfehlung](#) im [aws-encryption-sdk-cli](#) Repository unter GitHub.

Ein Beispiel, das zeigt, wie die Sicherheitsfunktion verwendet wird, die verschlüsselte Datenschlüssel einschränkt, finden Sie unter [Beschränkung verschlüsselter Datenschlüssel](#).

Ein Beispiel für die Verwendung von Schlüsseln AWS KMS mit mehreren Regionen finden Sie unter [Verwenden Sie mehrere Regionen AWS KMS keys](#).

Themen

- [Daten verschlüsseln und entschlüsseln](#)
- [So spezifizieren Sie Wrapping-Schlüssel](#)
- [Eingaben bereitstellen](#)
- [Den Ausgabespeicherort festlegen](#)
- [Einen Verschlüsselungskontext verwenden](#)
- [So spezifizieren Sie eine Verpflichtungspolitik](#)
- [Parameter in einer Konfigurationsdatei speichern](#)

Daten verschlüsseln und entschlüsseln

Die AWS Verschlüsselungs-CLI verwendet die Funktionen von, AWS Encryption SDK um das sichere Verschlüsseln und Entschlüsseln von Daten zu vereinfachen.

Note

Der `--master-keys` Parameter ist in Version 1.8 veraltet. x der AWS Encryption CLI und in Version 2.1 entfernt. x. Verwenden Sie stattdessen den `--wrapping-keys` Parameter. Ab Version 2.1. x, der `--wrapping-keys` Parameter ist beim Verschlüsseln und Entschlüsseln erforderlich. Details hierzu finden Sie unter [AWS Encryption SDK CLI – Syntax und Parameterreferenz](#).

- Wenn Sie Daten in der AWS Encryption CLI verschlüsseln, geben Sie Ihre Klartextdaten und einen [Wrapping-Schlüssel](#) (oder Masterschlüssel) an AWS KMS key, z. B. in (). AWS Key Management Service AWS KMS Wenn Sie einen benutzerdefinierten Hauptschlüsselanbieter

verwenden, müssen Sie auch den Anbieter angeben. Außerdem geben Sie Ausgabespeicherorte für die [verschlüsselte Nachricht](#) und für Metadaten über die Verschlüsselungsoperation an. Ein [Verschlüsselungskontext](#) ist optional, wird aber empfohlen.

In Version 1.8. x, der `--commitment-policy` Parameter ist erforderlich, wenn Sie den `--wrapping-keys` Parameter verwenden; andernfalls ist er nicht gültig. Ab Version 2.1. x, der `--commitment-policy` Parameter ist optional, wird aber empfohlen.

```
aws-encryption-cli --encrypt --input myPlaintextData \  
    --wrapping-keys key=1234abcd-12ab-34cd-56ef-1234567890ab \  
    --output myEncryptedMessage \  
    --metadata-output ~/metadata \  
    --encryption-context purpose=test \  
    --commitment-policy require-encrypt-require-decrypt
```

Die AWS Encryption CLI verschlüsselt Ihre Daten unter einem eindeutigen Datenschlüssel. Anschließend verschlüsselt es den Datenschlüssel unter den von Ihnen angegebenen Wrapping-Schlüsseln. Sie gibt eine [verschlüsselte Nachricht](#) und Metadaten über die Operation zurück. Die verschlüsselte Nachricht enthält Ihre verschlüsselten Daten (verschlüsselter Text) und eine verschlüsselte Kopie des Datenschlüssels. Sie müssen sich nicht um die Speicherung, Verwaltung oder den Verlust des Datenschlüssels kümmern.

- Wenn Sie Daten entschlüsseln, übergeben Sie Ihre verschlüsselte Nachricht, den optionalen Verschlüsselungskontext und den Speicherort für die Klartextausgabe und die Metadaten. Sie geben auch die Wickelschlüssel an, die die AWS Encryption CLI zum Entschlüsseln der Nachricht verwenden kann, oder teilen der AWS Encryption CLI mit, dass sie alle Wrapping-Schlüssel verwenden kann, mit denen die Nachricht verschlüsselt wurde.

Ab Version 1.8. x, der `--wrapping-keys` Parameter ist beim Entschlüsseln optional, wird aber empfohlen. Ab Version 2.1. x, der `--wrapping-keys` Parameter ist beim Verschlüsseln und Entschlüsseln erforderlich.

Beim Entschlüsseln können Sie das Schlüsselattribut des `--wrapping-keys` Parameters verwenden, um die Wrapping-Schlüssel anzugeben, mit denen Ihre Daten entschlüsselt werden. Die Angabe eines AWS KMS Wrapping-Schlüssels beim Entschlüsseln ist optional, aber es ist eine [bewährte Methode](#), die verhindert, dass Sie einen Schlüssel verwenden, den Sie nicht verwenden

wollten. Wenn Sie einen benutzerdefinierten Hauptschlüsselanbieter verwenden, müssen Sie den Anbieter und den Wrapping-Schlüssel angeben.

Wenn Sie das Schlüsselattribut nicht verwenden, müssen Sie das [Discovery-Attribut](#) des `--wrapping-keys` Parameters auf `true` festlegen, sodass die AWS Encryption-CLI die Entschlüsselung mithilfe eines beliebigen Wrapping-Schlüssels, der die Nachricht verschlüsselt hat, ermöglicht.

Es hat sich bewährt, den `--max-encrypted-data-keys` Parameter zu verwenden, um zu vermeiden, dass eine falsch formatierte Nachricht mit einer übermäßigen Anzahl von verschlüsselten Datenschlüsseln entschlüsselt wird. Geben Sie die erwartete Anzahl verschlüsselter Datenschlüssel (einer für jeden bei der Verschlüsselung verwendeten Wrapping-Schlüssel) oder ein angemessenes Maximum (z. B. 5) an. Details hierzu finden Sie unter [Beschränkung verschlüsselter Datenschlüssel](#).

Der `--buffer` Parameter gibt Klartext erst zurück, nachdem alle Eingaben verarbeitet wurden, einschließlich der Überprüfung der digitalen Signatur, falls eine vorhanden ist.

Der `--decrypt-unsigned` Parameter entschlüsselt Chiffretext und stellt sicher, dass Nachrichten vor der Entschlüsselung nicht signiert sind. Verwenden Sie diesen Parameter, wenn Sie den `--algorithm` Parameter verwendet und eine Algorithmussuite ohne digitale Signatur zum Verschlüsseln von Daten ausgewählt haben. Wenn der Chiffretext signiert ist, schlägt die Entschlüsselung fehl.

Sie können `--decrypt` oder `--decrypt-unsigned` für die Entschlüsselung verwenden, aber nicht beide.

```
aws-encryption-cli --decrypt --input myEncryptedMessage \  
  --wrapping-keys key=1234abcd-12ab-34cd-56ef-1234567890ab \  
  --output myPlaintextData \  
  --metadata-output ~/metadata \  
  --max-encrypted-data-keys 1 \  
  --buffer \  
  --encryption-context purpose=test \  
  --commitment-policy require-encrypt-require-decrypt
```

Die AWS Verschlüsselungs-CLI verwendet den Wrapping-Schlüssel, um den Datenschlüssel in der verschlüsselten Nachricht zu entschlüsseln. Anschließend verwendet sie den Datenschlüssel zum Entschlüsseln Ihrer Daten. Sie gibt Ihre Klartextdaten und Metadaten über die Operation zurück.

So spezifizieren Sie Wrapping-Schlüssel

Wenn Sie Daten in der AWS Encryption CLI verschlüsseln, müssen Sie mindestens einen [Wrapping-Schlüssel](#) (oder Masterschlüssel) angeben. Sie können AWS KMS keys in AWS Key Management Service (AWS KMS), Wrapping-Schlüssel von einem benutzerdefinierten [Hauptschlüsselanbieter](#) oder beides verwenden. Der Kundenmasterschlüssel-Anbieter kann ein beliebiger kompatibler Python-Masterschlüssel-Anbieter sein.

Um Wrapping-Schlüssel in den Versionen 1.8 zu spezifizieren. x und später verwenden Sie den `--wrapping-keys` Parameter (`-w`). Der Wert dieses Parameters ist eine Sammlung von [Attributen](#) mit dem `attribute=value` Format. Welche Attribute Sie verwenden, hängt von dem Masterschlüssel-Anbieter und dem Befehl ab.

- AWS KMS. In Verschlüsselungsbefehlen müssen Sie einen `--wrapping-keys` Parameter mit einem Schlüsselattribut angeben. Ab Version 2.1. x, der `--wrapping-keys` Parameter ist auch in Entschlüsselungsbefehlen erforderlich. Beim Entschlüsseln muss der `--wrapping-keys` Parameter ein Schlüsselattribut oder ein Discovery-Attribut mit einem Wert von `true` (aber nicht beides) haben. Andere Attribute sind optional.
- Kundenmasterschlüssel-Anbieter. Sie müssen in jedem Befehl einen `--wrapping-keys` Parameter angeben. Der Parameterwert muss `key-` und `provider-`Attribute besitzen.

Sie können [mehrere `--wrapping-keys` Parameter](#) und mehrere Schlüsselattribute in denselben Befehl einschließen.

Umschließen wichtiger Parameterattribute

Der Wert des `--wrapping-keys`-Parameters besteht aus den folgenden Attributen und ihren Werten. In allen Verschlüsselungsbefehlen ist ein `--wrapping-keys --master-keys` Parameter (oder Parameter) erforderlich. Ab Version 2.1. x, der `--wrapping-keys` Parameter ist auch beim Entschlüsseln erforderlich.

Wenn ein Attributname oder Wert Leerzeichen oder Sonderzeichen enthält, schließen Sie den Namen und den Wert in Anführungszeichen ein. Zum Beispiel `--wrapping-keys key=12345 "provider=my cool provider"`.

Schlüssel: Geben Sie einen Umhüllungsschlüssel an

Verwenden Sie das Schlüsselattribut, um einen Verpackungsschlüssel zu identifizieren. Bei der Verschlüsselung kann der Wert ein beliebiger Schlüsselbezeichner sein, den der Hauptschlüsselanbieter erkennt.

```
--wrapping-keys key=1234abcd-12ab-34cd-56ef-1234567890ab
```

In einem Verschlüsselungsbefehl müssen Sie mindestens ein Schlüsselattribut und einen Wert angeben. Verwenden Sie mehrere [Schlüsselattribute](#), um Ihren Datenschlüssel unter mehreren Umhüllungsschlüsseln zu verschlüsseln.

```
aws-encryption-cli --encrypt --wrapping-keys  
key=1234abcd-12ab-34cd-56ef-1234567890ab key=1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d
```

In Verschlüsselungsbefehlen, die verwendet werdenAWS KMS keys, kann der Wert des Schlüssels die Schlüssel-ID, sein Schlüssel-ARN, ein Aliasname oder ein Alias-ARN sein. Dieser Verschlüsselungsbefehl beispielsweise verwendet einen Alias-ARN im Wert des key-Attributs. Einzelheiten zu den Schlüsselkennungen für eine AWS KMS key finden Sie unter [Schlüsselkennungen](#) im AWS Key Management ServiceEntwicklerhandbuch.

```
aws-encryption-cli --encrypt --wrapping-keys key=arn:aws:kms:us-  
west-2:111122223333:alias/ExampleAlias
```

In Entschlüsselungsbefehlen, die einen Kundenmasterschlüssel-Anbieter verwenden, müssen die key- und provider-Attribute angegeben werden.

```
\\ Custom master key provider  
aws-encryption-cli --decrypt --wrapping-keys provider='myProvider' key='100101'
```

In Entschlüsselungsbefehlen, die verwendet werdenAWS KMS, können Sie das Schlüsselattribut verwenden, um das für die Entschlüsselung AWS KMS keys zu verwendende Attribut anzugeben, oder das [Discovery-Attribut](#) mit einem Wert von `true`, sodass die Encryption-CLI jedes Attribut verwenden kann, AWS KMS key das zum AWS Verschlüsseln der Nachricht verwendet wurde. Wenn Sie einen angebenAWS KMS key, muss es sich um einen der Verpackungsschlüssel handeln, die zum Verschlüsseln der Nachricht verwendet wurden.

Die Angabe des Wrapping-Schlüssels hat sich [AWS Encryption SDKbewährt](#). Es stellt sicher, dass Sie das verwenden, AWS KMS key was Sie verwenden möchten.

In einem Entschlüsselungsbefehl muss der Wert des Schlüsselattributs ein [Schlüssel-ARN](#) sein.

```
\\ AWS KMS key
aws-encryption-cli --decrypt --wrapping-keys key=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

Entdeckung: Verwenden Sie AWS KMS key beim Entschlüsseln eine beliebige

Wenn Sie die AWS KMS keys Verwendung beim Entschlüsseln nicht einschränken müssen, können Sie das Discovery-Attribut mit einem Wert von `true` verwenden. Ein Wert von `true` ermöglicht es der AWS Encryption CLI, sie mit einem beliebigen Code zu entschlüsseln, der AWS KMS key die Nachricht verschlüsselt hat. Wenn Sie kein Discovery-Attribut angeben, ist Discovery `false` (Standard). Das Discovery-Attribut ist nur in Entschlüsselungsbefehlen gültig und nur, wenn die Nachricht mit AWS KMS keys verschlüsselt wurde.

Das Discovery-Attribut mit einem Wert von `true` ist eine Alternative zur Verwendung des Schlüsselattributs zur Angabe AWS KMS keys. Beim Entschlüsseln einer mit AWS KMS keys verschlüsselten Nachricht muss jeder `--wrapping-keys` Parameter ein Schlüsselattribut oder ein Discovery-Attribut mit einem Wert von `haben true`, aber nicht beide.

Wenn die Erkennung wahr ist, empfiehlt es sich, die Attribute `discovery-partition` und `discovery-account` zu verwenden, um die AWS KMS keys verwendeten Attribute auf die von Ihnen angegebenen Attribute zu beschränken. AWS-Konten Im folgenden Beispiel ermöglichen die Discovery-Attribute der AWS Encryption CLI, jedes AWS KMS key der angegebenen AWS-Konten Attribute zu verwenden.

```
aws-encryption-cli --decrypt --wrapping-keys \
  discovery=true \
  discovery-partition=aws \
  discovery-account=111122223333 \
  discovery-account=444455556666
```

Anbieter: Geben Sie den Hauptschlüsselanbieter an

Das `provider`-Attribut identifiziert den [Masterschlüssel-Anbieter](#). Der Standardwert ist `aws-kms` und steht für AWS KMS. Wenn Sie einen anderen Masterschlüsselanbieter verwenden, ist das `provider`-Attribut nicht erforderlich.

```
--wrapping-keys key=12345 provider=my_custom_provider
```

Weitere Informationen zur Verwendung von benutzerdefinierten (nicht-AWS KMS) Hauptschlüsselanbietern finden Sie im Thema Erweiterte Konfiguration in der [README-Datei](#) für das [AWSEncryption CLI-Repository](#).

Region: Geben Sie eine an AWS-Region

Verwenden Sie das `regions`-Attribut, um das AWS-Region von einem anzugeben AWS KMS key. Dieses Attribut ist nur in Verschlüsselungsbefehlen und nur dann gültig, wenn der Masterschlüssel-Anbieter AWS KMS ist.

```
--encrypt --wrapping-keys key=alias/primary-key region=us-east-2
```

AWS CLI-Befehle für Verschlüsselung verwenden den Wert AWS-Region, der im Schlüsselattributwert angegeben ist, wenn er eine Region enthält, z. B. einen ARN. Wenn der Schlüsselwert eine AWS-Region angibt, wird das Regionsattribut ignoriert.

Das `region`-Attribut hat Vorrang vor allen anderen Regionsangaben. Wenn Sie kein Regionsattribut verwenden, verwenden AWS Encryption CLI-Befehle das in Ihrem AWS CLI [benannten Profil](#), falls vorhanden, AWS-Region angegebene oder Ihr Standardprofil.

Profil: Angabe eines benannten Profils

Verwenden Sie das `profile`-Attribut, um ein AWS CLI [benanntes Profil](#) anzugeben. Benannte Profile können Anmeldeinformationen und eine enthaltene AWS-Region. Dieses Attribut ist nur gültig, wenn der Masterschlüssel-Anbieter AWS KMS ist.

```
--wrapping-keys key=alias/primary-key profile=admin-1
```

Sie können das `profile`-Attribut verwenden, um alternative Anmeldeinformationen in Ver- und Entschlüsselungsbefehlen anzugeben. In einem Verschlüsselungsbefehl verwendet die AWS Verschlüsselungs-CLI das AWS-Region im benannten Profil nur, wenn der Schlüsselwert keine Region enthält und es kein Regionsattribut gibt. In einem Entschlüsselungsbefehl wird das AWS-Region im Namen angegebene Profil ignoriert.

So geben Sie mehrere Wrapping-Schlüssel an

Sie können in jedem Befehl mehrere Umhüllungsschlüssel (oder Hauptschlüssel) angeben.

Wenn Sie mehr als einen Verpackungsschlüssel angeben, generiert und verschlüsselt der erste Verpackungsschlüssel den Datenschlüssel, der zum Verschlüsseln Ihrer Daten verwendet wird. Die

anderen Wickelschlüssel verschlüsseln denselben Datenschlüssel. Die resultierende [verschlüsselte Nachricht](#) enthält die verschlüsselten Daten („Ciphertext“) und eine Sammlung von verschlüsselten Datenschlüsseln, von denen einer mit jedem Wrapping-Schlüssel verschlüsselt wird. Jedes Wrapping kann einen verschlüsselten Datenschlüssel entschlüsseln und dann die Daten entschlüsseln.

Es gibt zwei Möglichkeiten, mehrere Umhüllungsschlüssel anzugeben:

- Fügen Sie mehrere Schlüsselattribute in den `--wrapping-keys` Parameterwert ein.

```
$key_oregon=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
$key_ohio=arn:aws:kms:us-east-2:111122223333:key/0987ab65-43cd-21ef-09ab-87654321cdef

--wrapping-keys key=$key_oregon key=$key_ohio
```

- Verwendung von mehreren `--wrapping-keys`-Parametern im selben Befehl. Verwenden Sie diese Syntax, wenn die von Ihnen angegebenen Attributwerte nicht für alle Umbruchschlüssel im Befehl gelten.

```
--wrapping-keys region=us-east-2 key=alias/test_key \
--wrapping-keys region=us-west-1 key=alias/test_key
```

Das Discovery-Attribut mit dem Wert von `true` ermöglicht es der AWS Encryption CLI AWS KMS key, jedes zu verwenden, das die Nachricht verschlüsselt hat. Wenn Sie mehrere `--wrapping-keys` Parameter in demselben Befehl verwenden, überschreibt die Verwendung `discovery=true` in einem beliebigen `--wrapping-keys` Parameter effektiv die Grenzen des Schlüsselattributs in anderen `--wrapping-keys` Parametern.

Im folgenden Befehl beschränkt das Schlüsselattribut im ersten `--wrapping-keys` Parameter beispielsweise die AWS Verschlüsselungs-CLI auf den angegebenen AWS KMS key Wert. Das Discovery-Attribut im zweiten `--wrapping-keys` Parameter ermöglicht es der AWS Encryption CLI jedoch, jedes AWS KMS key der angegebenen Konten zu verwenden, um die Nachricht zu entschlüsseln.

```
aws-encryption-cli --decrypt \
  --wrapping-keys key=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab \
  --wrapping-keys discovery=true \
    discovery-partition=aws \
```

```
discovery-account=111122223333 \  
discovery-account=444455556666
```

Eingaben bereitstellen

[Der Verschlüsselungsvorgang in der AWS Encryption CLI verwendet Klartextdaten als Eingabe und gibt eine verschlüsselte Nachricht zurück.](#) Die Entschlüsselungsoperation übernimmt eine verschlüsselte Nachricht als Eingabe und gibt Klartextdaten zurück.

Der `--input` Parameter (`-i`), der der AWS Encryption-CLI mitteilt, wo die Eingabe zu finden ist, ist in allen AWS Encryption-CLI-Befehlen erforderlich.

Sie können Eingaben wie folgt bereitstellen:

- Über eine Datei.

```
--input myData.txt
```

- Unter Verwendung eines Dateinamenmusters.

```
--input testdir/*.xml
```

- Unter Verwendung eines Verzeichnisses oder Verzeichnisnamenmusters. Wenn die Eingabe ein Verzeichnis ist, muss der `--recursive`-Parameter (`-r`, `-R`) angegeben werden.

```
--input testdir --recursive
```

- Eingabe an den Befehl weiterleiten (stdin). Einen Wert von `-` für den `--input`-Parameter verwenden. (Der Parameter `--input` ist immer erforderlich.)

```
echo 'Hello World' | aws-encryption-cli --encrypt --input -
```

Den Ausgabespeicherort festlegen

Der `--output` Parameter teilt der AWS Encryption CLI mit, wo die Ergebnisse des Verschlüsselungs- oder Entschlüsselungsvorgangs geschrieben werden sollen. Es ist in jedem AWS Encryption-CLI-Befehl erforderlich. Die AWS Encryption CLI erstellt für jede Eingabedatei in der Operation eine neue Ausgabedatei.

Wenn bereits eine Ausgabedatei existiert, gibt die AWS Verschlüsselungs-CLI standardmäßig eine Warnung aus und überschreibt dann die Datei. Um ein Überschreiben zu verhindern, verwenden Sie den Parameter `--interactive`, der Sie vor dem Überschreiben zur Bestätigung auffordert, oder `--no-overwrite`, der die Eingabe überspringt, wenn die Ausgabe ein Überschreiben verursachen würde. Um die Überschreibwarnung zu unterdrücken, verwenden Sie `--quiet`. Um Fehler und Warnungen von der AWS Encryption CLI zu erfassen, verwenden Sie den `>&1` Umleitungsoperator, um sie in den Ausgabestream zu schreiben.

Note

Befehle, die Ausgabedateien überschreiben, löschen zunächst die Ausgabedatei. Wenn der Befehl fehlschlägt, ist die Ausgabedatei möglicherweise bereits gelöscht.

Sie können den Ausgabespeicherort auf verschiedene Arten wählen.

- Angabe eines Dateinamens. Wenn Sie einen Pfad zur Datei angeben, müssen alle Verzeichnisse im Pfad vorhanden sein, bevor der Befehl ausgeführt wird.

```
--output myEncryptedData.txt
```

- Angabe eines Verzeichnisses. Die Ausgabeverzeichnis muss vorhanden sein, bevor der Befehl ausgeführt wird.

Wenn die Eingabe Unterverzeichnisse enthält, erzeugt der Befehl die Unterverzeichnisse unter dem angegebenen Verzeichnis.

```
--output Test
```

Wenn das Ausgabeverzeichnis ein Verzeichnis (ohne Dateinamen) ist, erstellt die AWS Verschlüsselungs-CLI Ausgabedateinamen auf der Grundlage der Eingabedateinamen plus einem Suffix. Verschlüsselungsoperationen fügen dem Eingabedateinamen `.encrypted` hinzu, Entschlüsselungsoperationen fügen `.decrypted` hinzu. Mit dem Parameter `--suffix` ändern Sie das Suffix.

Wenn Sie z. B. `file.txt` verschlüsseln, erstellt der Verschlüsselungsbefehl `file.txt.encrypted`. Wenn Sie `file.txt.encrypted` entschlüsseln, erstellt der Entschlüsselungsbefehl `file.txt.encrypted.decrypted`.

- In die Befehlszeile schreiben (stdout). Geben Sie einen Wert von - für den --output-Parameter ein. Sie können --output - verwenden, um die Ausgabe an einen anderen Befehl oder ein Programm weiterzuleiten.

```
--output -
```

Einen Verschlüsselungskontext verwenden

Mit der AWS Encryption CLI können Sie einen Verschlüsselungskontext für Verschlüsselungs- und Entschlüsselungsbefehle bereitstellen. Es ist nicht erforderlich, aber eine bewährte Methode für die Kryptografie, die wir empfehlen.

Ein Verschlüsselungskontext ist eine Art zufälliger, nicht geheimer zusätzlicher authentifizierter Daten. In der AWS Encryption CLI besteht der Verschlüsselungskontext aus einer Sammlung von name=value Paaren. Sie können beliebigen Inhalt in den Paaren verwenden, einschließlich Informationen über die Dateien, Daten, die Ihnen helfen, den Verschlüsselungsvorgang in Protokollen zu finden, oder Daten, die Sie für Ihre Berechtigungen oder Richtlinien benötigen.

In einem Verschlüsselungsbefehl

Der Verschlüsselungskontext, den Sie in einem Verschlüsselungsbefehl angeben, sowie alle weiteren Paare, die vom [CMM](#) hinzugefügt werden, sind kryptographisch an die verschlüsselten Daten gebunden. Er ist außerdem (in Klartext) in der [verschlüsselten Nachricht](#) enthalten, die den Befehl zurückgibt. Wenn Sie eine verwenden AWS KMS key, wird der Verschlüsselungskontext möglicherweise auch im Klartext in Prüfprotokollen und -protokollen angezeigt, z. B. AWS CloudTrail

Das folgende Beispiel zeigt einen Verschlüsselungskontext mit drei name=value-Paaren.

```
--encryption-context purpose=test dept=IT class=confidential
```

In einem Entschlüsselungsbefehl

In einem Entschlüsselungsbefehl hilft Ihnen der Verschlüsselungskontext zu bestätigen, dass Sie die richtige verschlüsselte Nachricht entschlüsseln.

Sie müssen in einem Entschlüsselungsbefehl keinen Verschlüsselungskontext angeben, auch wenn beim Verschlüsseln ein Verschlüsselungskontext verwendet wurde. Wenn Sie dies jedoch

tun, überprüft die AWS Encryption CLI, ob jedes Element im Verschlüsselungskontext des Entschlüsselungsbefehls mit einem Element im Verschlüsselungskontext der verschlüsselten Nachricht übereinstimmt. Wenn ein Element nicht übereinstimmt, schlägt der Entschlüsselungsbefehl fehl.

Beispielsweise entschlüsselt der folgende Befehl die verschlüsselte Nachricht nur, wenn ihr Verschlüsselungskontext `dept=IT` enthält.

```
aws-encryption-cli --decrypt --encryption-context dept=IT ...
```

Ein Verschlüsselungskontext ist ein wichtiger Teil Ihrer Sicherheitsstrategie. Wenn Sie jedoch einen Verschlüsselungskontext wählen, denken Sie jedoch daran, dass seine Werte nicht geheim sind. Nehmen Sie keine vertraulichen Daten in den Verschlüsselungskontext auf.

So geben Sie einen Verschlüsselungskontext an

- Verwenden Sie in einem `encrypt`-Befehl den `--encryption-context`-Parameter mit einem oder mehreren `name=value`-Paaren. Verwenden Sie ein Leerzeichen, um die einzelnen Paare zu trennen.

```
--encryption-context name=value [name=value] ...
```

- In einem `decrypt`-Befehl kann der `--encryption-context`-Parameterwert `name=value`-Paare, `name`-Elemente (ohne Werte) oder eine Kombination aus beidem enthalten.

```
--encryption-context name[=value] [name] [name=value] ...
```

Wenn der `name` oder `value` in einem `name=value`-Paar Leerzeichen oder Sonderzeichen enthält, schließen Sie gesamte Paar in Anführungszeichen ein.

```
--encryption-context "department=software engineering" "AWS-Region=us-west-2"
```

Dieser Entschlüsselungsbefehl enthält beispielsweise einen Verschlüsselungskontext mit zwei Paaren, `purpose=test` und `dept=23`.

```
aws-encryption-cli --decrypt --encryption-context purpose=test dept=23 ...
```

Dieser Entschlüsselungsbefehl würde erfolgreich ausgeführt. Der Verschlüsselungskontext in den einzelnen Befehlen ist eine Teilmenge des ursprünglichen Verschlüsselungskontexts.

```
\\ Any one or both of the encryption context pairs
aws-encryption-cli --decrypt --encryption-context dept=23 ...

\\ Any one or both of the encryption context names
aws-encryption-cli --decrypt --encryption-context purpose ...

\\ Any combination of names and pairs
aws-encryption-cli --decrypt --encryption-context dept purpose=test ...
```

Diese Verschlüsselungsbefehle würden jedoch fehlschlagen. Der Verschlüsselungskontext in der verschlüsselten Nachricht enthält nicht die angegebenen Elemente.

```
aws-encryption-cli --decrypt --encryption-context dept=Finance ...
aws-encryption-cli --decrypt --encryption-context scope ...
```

So spezifizieren Sie eine Verpflichtungspolitik

Verwenden Sie den [--commitment-policyParameter](#), um die [Commitment-Richtlinie](#) für den Befehl festzulegen. Dieser Parameter wurde in Version 1.8 eingeführt. x. Es ist gültig für Befehle zum Verschlüsseln und Entschlüsseln. Die von Ihnen festgelegte Commitment-Richtlinie gilt nur für den Befehl, in dem sie erscheint. Wenn Sie keine Commitment-Richtlinie für einen Befehl festlegen, verwendet die AWS Verschlüsselungs-CLI den Standardwert.

Mit dem folgenden Parameterwert wird beispielsweise die Commitment-Richtlinie auf festgelegt `require-encrypt-allow-decrypt`, die immer mit Schlüsselzusage verschlüsselt, aber einen Chiffretext entschlüsselt, der mit oder ohne Schlüsselzusage verschlüsselt wurde.

```
--commitment-policy require-encrypt-allow-decrypt
```

Parameter in einer Konfigurationsdatei speichern

Sie können Zeit sparen und Tippfehler vermeiden, indem Sie häufig verwendete AWS Verschlüsselungs-CLI-Parameter und -Werte in Konfigurationsdateien speichern.

Eine Konfigurationsdatei ist eine Textdatei, die Parameter und Werte für einen AWS Encryption-CLI-Befehl enthält. Wenn Sie in einem AWS Encryption-CLI-Befehl auf eine Konfigurationsdatei

verweisen, wird die Referenz durch die Parameter und Werte in der Konfigurationsdatei ersetzt. Der Effekt ist der gleiche, als ob Sie den Dateiinhalt in der Befehlszeile eingegeben. Eine Konfigurationsdatei kann einen beliebigen Namen haben und sich in einem beliebigen Verzeichnis befinden, auf das der aktuelle Benutzer zugreifen kann.

Die folgende Beispielkonfigurationsdatei `key.conf`, spezifiziert zwei AWS KMS keys in verschiedenen Regionen.

```
--wrapping-keys key=arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab  
--wrapping-keys key=arn:aws:kms:us-  
east-2:111122223333:key/0987ab65-43cd-21ef-09ab-87654321cdef
```

Um die Konfigurationsdatei in einem Befehl zu verwenden, stellen sie dem Dateinamen ein At-Zeichen (@) voraus. Verwenden Sie in einer PowerShell Konsole ein Backtick-Zeichen, um das At-Zeichen (`@) zu umgehen.

Dieser Beispielbefehl verwendet die Datei `key.conf` in einem Verschlüsselungsbefehl.

Bash

```
$ aws-encryption-cli -e @key.conf -i hello.txt -o testdir
```

PowerShell

```
PS C:\> aws-encryption-cli -e `@key.conf -i .\Hello.txt -o .\TestDir
```

Konfigurationsdateiregeln

Für die Verwendung von Konfigurationsdateien gelten die folgenden Regeln:

- Sie können mehrere Parameter in jede Konfigurationsdatei aufnehmen und in beliebiger Reihenfolge auflisten. Listen Sie jeden Parameter mit seinen Werten (falls vorhanden) in einer separaten Zeile auf.
- Verwenden Sie # zum Hinzufügen eines Kommentars für eine ganze oder einen Teil einer Zeile.
- Sie können Verweise auf andere Konfigurationsdateien aufnehmen. Benutze keinen Backtick, um dem @ Schild zu entkommen, auch nicht in PowerShell.

- Wenn Sie Anführungszeichen in einer Konfigurationsdatei verwenden, kann sich der angegebene Text nicht über mehrere Zeilen erstrecken.

Dies ist beispielsweise der Inhalt einer `encrypt.conf`-Beispieldatei.

```
# Archive Files
--encrypt
--output /archive/logs
--recursive
--interactive
--encryption-context class=unclassified dept=IT
--suffix # No suffix
--metadata-output ~/metadata
@caching.conf # Use limited caching
```

Sie können auch mehrere Konfigurationsdateien in einem Befehl angeben. Dieser Beispielbefehl verwendet die Konfigurationsdateien `encrypt.conf` und `master-keys.conf`.

Bash

```
$ aws-encryption-cli -i /usr/logs @encrypt.conf @master-keys.conf
```

PowerShell

```
PS C:\> aws-encryption-cli -i $home\Test\*.log `@encrypt.conf `@master-keys.conf
```

Als Nächstes: [Probieren Sie die Beispiele für die AWS Verschlüsselungs-CLI](#) aus

Beispiele für die AWS CLI für Verschlüsselung

Verwenden Sie die folgenden Beispiele, um die AWS Verschlüsselungs-CLI auf der Plattform, die Sie bevorzugen. Weitere Informationen zum Master-Schlüssel und anderen Parametern finden Sie unter [So verwenden Sie die AWS Encryption CLI](#). Informationen zum schnellen Einstieg finden Sie unter [AWS Encryption SDK CLI – Syntax und Parameterreferenz](#).

Note

Die folgenden Beispiele verwenden die Syntax für AWS Verschlüsselung CLI Version 2.1.x aus.

Neue Sicherheitsfunktionen wurden ursprünglich in AWS Verschlüsselung CLI Version 1.7.x2.0 und 2.0.x aus. Allerdings AWS Verschlüsselung CLI Version 1.8.x ersetzt Version 1.7.x und AWS Verschlüsselung CLI 2.1.x ersetzt 2.0.x aus. Einzelheiten finden Sie im entsprechenden [Sicherheitsberatung im aws-encryption-sdk-CLI-Repository](#) auf GitHub aus.

Ein Beispiel, das zeigt, wie Sie die Sicherheitsfunktion verwenden, die verschlüsselte Datenschlüssel einschränkt, finden Sie unter [Beschränkung verschlüsselter Datenschlüssel](#) aus.

Für ein Beispiel, das zeigt, wie man benutzt AWS KMS Weitere regionale Schlüssel finden Sie unter [Verwenden Sie mehrere Regionen AWS KMS keys](#) aus.

Themen

- [Verschlüsseln einer Datei](#)
- [Entschlüsseln einer Datei](#)
- [Alle Dateien in einem Verzeichnis verschlüsseln](#)
- [Alle Dateien in einem Verzeichnis entschlüsseln](#)
- [Verschlüsseln und Entschlüsseln in der Befehlszeile](#)
- [Verwenden mehrerer Hauptschlüssel](#)
- [Verschlüsseln und Entschlüsseln in Skripten](#)
- [Verwenden von Datenschlüssel-Caching](#)

Verschlüsseln einer Datei

In diesem Beispiel verwenden wir den AWS Verschlüsselungs-CLI zum Verschlüsseln des `hello.txt`-Datei, die eine Zeichenfolge „Hello World“ enthält.

Wenn Sie einen verschlüsselten Befehl für eine Datei ausführen, wird der AWS Verschlüsselungs-CLI ruft den Inhalt der Datei ab, generiert eine eindeutige [Datenschlüssel](#), verschlüsselt den Dateiinhalt unter dem Datenschlüssel und schreibt dann [Verschlüsselte Nachricht](#) in eine neue Datei.

Der erste Befehl speichert den Schlüssel-ARN eines AWS KMS key in `$keyArnVariable`. Beim Verschlüsseln mit einem AWS KMS key verwenden, können Sie es anhand einer Schlüssel-ID, eines Schlüssel-ARN, eines Aliasnamens oder Alias-ARN identifizieren. Für Details über die Schlüsselkennungen für AWS KMS key finden Sie unter [Key Bezeichner](#) im AWS Key Management Service Entwicklerhandbuch aus.

Der zweite Befehl verschlüsselt den Dateinhalt. Der Befehl verwendet den Parameter `--encrypt`, um die Operation anzugeben, und den Parameter `--input`, um die zu verschlüsselnde Datei anzugeben. Der [--wrapping-keys-Parameter](#) und das dafür erforderliche Schlüsselattribut weisen den Befehl an, den durch den Schlüssel-ARN dargestellten AWS KMS key zu verwenden.

Der Befehl verwendet den `--metadata-output`-Parameter, um eine Textdatei für die Metadaten über die Verschlüsselungsoperation anzugeben. Als bewährte Methode verwendet der Befehl den `--encryption-context`-Parameter, um einen [Verschlüsselungskontext](#) anzugeben.

Dieser Befehl verwendet auch die [--commitment-policyEinstellung](#) um die Verpflichtungspolitik explizit festzulegen. In Version 1.8.x Dieser Parameter ist erforderlich, wenn Sie die `--wrapping-keys`-Parameter. Ab Version 2.1.x, der `--commitment-policy` Der Parameter ist zwar optional, wird aber empfohlen.

Der Wert des `--output`-Parameters, ein Punkt (`.`), weist den Befehl an, die Ausgabedatei in das aktuelle Verzeichnis zu speichern.

Bash

```

\\ To run this example, replace the fictitious key ARN with a valid value.
$ keyArn=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

$ aws-encryption-cli --encrypt \
    --input hello.txt \
    --wrapping-keys key=$keyArn \
    --metadata-output ~/metadata \
    --encryption-context purpose=test \
    --commitment-policy require-encrypt-require-decrypt \
    --output .

```

PowerShell

```

# To run this example, replace the fictitious key ARN with a valid value.
PS C:\> $keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

PS C:\> aws-encryption-cli --encrypt `
    --input Hello.txt `
    --wrapping-keys key=$keyArn `
    --metadata-output $home\Metadata.txt `
    --commitment-policy require-encrypt-require-decrypt `
    --encryption-context purpose=test `

```

```
--output .
```

Wenn der Verschlüsselungsbefehl erfolgreich ausgeführt wird, erfolgt keine Ausgabe. Um festzustellen, ob der Befehl erfolgreich ausgeführt wurde, überprüfen Sie den booleschen Wert in der Variablen `$?`. Wenn der Befehl erfolgreich war, ist der Wert von `$?ist0` (Bash) oder `True` (PowerShell) enthalten. Wenn der Befehl fehlschlägt, ist der Wert von `$?ist` ungleich Null (Bash) oder `False` (PowerShell) enthalten.

Bash

```
$ echo $?  
0
```

PowerShell

```
PS C:\> $?  
True
```

Sie können auch einen Verzeichnisauflistungsbefehl ausführen, um zu überprüfen, ob der Verschlüsselungsbefehl eine neue Datei erstellt hat, `hello.txt.encrypted`. Da der Verschlüsselungsbefehl keinen Dateinamen für die Ausgabe angegeben hat, hat die AWS Verschlüsselungs-CLI die Ausgabe in eine Datei mit dem gleichen Namen wie die Eingabedatei mit der `.encrypted` Suffix. Um ein anderes Suffix zu verwenden oder das Suffix wegzulassen, verwenden Sie den `--suffix`-Parameter.

Die Datei `hello.txt.encrypted` enthält eine [verschlüsselte Nachricht](#), die den Verschlüsselungstext der `hello.txt`-Datei enthält, eine verschlüsselte Kopie des Datenschlüssels und zusätzlichen Metadaten, einschließlich des Verschlüsselungskontexts.

Bash

```
$ ls  
hello.txt hello.txt.encrypted
```

PowerShell

```
PS C:\> dir
```

```
Directory: C:\TestCLI
```

Mode	LastWriteTime	Length	Name
----	-----	-----	----
-a----	9/15/2017 5:57 PM	11	Hello.txt
-a----	9/17/2017 1:06 PM	585	Hello.txt.encrypted

Entschlüsseln einer Datei

In diesem Beispiel verwenden wir den AWS Verschlüsselungs-CLI zum Entschlüsseln des Inhalts der `Hello.txt.encrypted` Datei, die im vorangegangenen Beispiel verschlüsselt wurde.

Der Entschlüsselungsbefehl verwendet den Parameter `--decrypt`, um die Operation anzugeben, und den Parameter `--input`, um die zu entschlüsselnde Datei anzugeben. Der Wert des `--output`-Parameters ist ein Punkt, der das aktuelle Verzeichnis darstellt.

Die `--wrapping-keys` Parameterwert mit einem Schlüsselattribut gibt den Umschlagschlüssel an, der zum Entschlüsseln der verschlüsselten Nachricht verwendet wird. Im Entschlüsseln von Befehlen mit AWS KMS keys muss der Wert des Schlüsselattributs [Schlüssel-ARN](#) aus. Die `--wrapping-keys` Der Parameter muss in einem Entschlüsselungsbefehl angegeben werden. Wenn Sie verwenden AWS KMS keys verwenden können Sie die Schlüsselattribut zum Angabe AWS KMS key zum Entschlüsseln oder `Entdeckung` Attribut mit einem Wert von `true` (aber nicht beides). Wenn Sie einen Kundenmasterschlüssel-Anbieter verwenden, wird die `Schlüssel` und `Anbieter` Attribute sind erforderlich.

Die [--commitment-policy](#) Einstellung ist ab Version 2.1 optional, aber es wird empfohlen. Wenn Sie es verwenden, wird Ihre Absicht explizit klar, auch wenn Sie den Standardwert angeben, `require-encrypt-require-decrypt` aus.

Der Parameter `--encryption-context` ist optional im Entschlüsselungsbefehl, auch dann, wenn im Verschlüsselungsbefehl ein [Verschlüsselungskontext](#) angegeben ist. In diesem Fall verwendet der Entschlüsselungsbefehl denselben Verschlüsselungskontext wie denjenigen, der im Verschlüsselungsbefehl angegeben wurde. Vor dem Entschlüsseln wird der AWS Die Verschlüsselungs-CLI überprüft, ob der Verschlüsselungskontext in der verschlüsselten Nachricht einen `purpose=test` Paar. Wenn dies nicht der Fall ist, schlägt der Entschlüsselungsbefehl fehl.

Der `--metadata-output`-Parameter gibt eine Datei für Metadaten über die Entschlüsselungsoperation an. Der Wert des `--output`-Parameters, ein Punkt (`.`), weist den Befehl an, die Ausgabe datei in das aktuelle Verzeichnis zu schreiben.

Verwenden Sie als bewährte Methode die `--max-encrypted-data-keys`-Parameter, um zu vermeiden, dass eine fehlerhafte Nachricht mit einer übermäßigen Anzahl verschlüsselter Datenschlüssel entschlüsselt wird. Geben Sie die erwartete Anzahl verschlüsselter Datenschlüssel (einen für jeden in der Verschlüsselung verwendeten Umschlüsselschlüssel) oder ein angemessenes Maximum (z. B. 5) an. Details hierzu finden Sie unter [Beschränkung verschlüsselter Datenschlüssel](#).

Die `--buffer` gibt Klartext erst zurück, nachdem alle Eingaben verarbeitet wurden, einschließlich der Überprüfung der digitalen Signatur, falls eine vorhanden ist.

Bash

```

\\ To run this example, replace the fictitious key ARN with a valid value.
$ keyArn=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

$ aws-encryption-cli --decrypt \
    --input hello.txt.encrypted \
    --wrapping-keys key=$keyArn \
    --commitment-policy require-encrypt-require-decrypt \
    --encryption-context purpose=test \
    --metadata-output ~/metadata \
    --max-encrypted-data-keys 1 \
    --buffer \
    --output .

```

PowerShell

```

\\ To run this example, replace the fictitious key ARN with a valid value.
PS C:\> $keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

PS C:\> aws-encryption-cli --decrypt `
    --input Hello.txt.encrypted `
    --wrapping-keys key=$keyArn `
    --commitment-policy require-encrypt-require-decrypt `
    --encryption-context purpose=test `
    --metadata-output $home\Metadata.txt `
    --max-encrypted-data-keys 1 `
    --buffer `
    --output .

```

Wenn der Entschlüsselungsbefehl erfolgreich ausgeführt wird, erfolgt keine Ausgabe. Um festzustellen, ob der Befehl erfolgreich ausgeführt wurde, rufen Sie den Wert der Variablen `$?` ab. Sie können auch einen Verzeichnisauflistungsbefehl ausführen, um zu überprüfen, ob der Verschlüsselungsbefehl eine neue Datei erstellt mit dem Suffix `.decrypted` erstellt hat. Um den Klartextinhalt anzuzeigen, verwenden Sie einen Befehl, um den Dateiinhalt abzurufen, wie beispielsweise `cat` oder [Get-Content](#).

Bash

```
$ ls
hello.txt  hello.txt.encrypted  hello.txt.encrypted.decrypted

$ cat hello.txt.encrypted.decrypted
Hello World
```

PowerShell

```
PS C:\> dir

Directory: C:\TestCLI

Mode                LastWriteTime         Length Name
----                -
-a----             9/17/2017   1:01 PM             11 Hello.txt
-a----             9/17/2017   1:06 PM           585 Hello.txt.encrypted
-a----             9/17/2017   1:08 PM             11 Hello.txt.encrypted.decrypted

PS C:\> Get-Content Hello.txt.encrypted.decrypted
Hello World
```

Alle Dateien in einem Verzeichnis verschlüsseln

In diesem Beispiel verwenden wir den AWS Verschlüsselungs-CLI zum Verschlüsseln des Inhalts aller Dateien in einem Verzeichnis.

Wenn ein Befehl mehrere Dateien betrifft, wird die AWS CLI für die Verschlüsselung verarbeitet jede Datei einzeln. Sie ruft den Dateiinhalt ab, ruft einen eindeutigen [Datenschlüssel](#) für die Datei aus dem Master-Schlüssel ab, verschlüsselt den Inhalt der Datei unter dem Datenschlüssel und

schreibt die Ergebnisse in eine neue Datei im Ausgabeverzeichnis. Aus diesem Grund können Sie die Ausgabedateien unabhängig voneinander entschlüsseln.

Diese Auflistung des TestDir-Verzeichnisses zeigt die Klartext-Dateien, die wir verschlüsseln möchten.

Bash

```
$ ls testdir
cool-new-thing.py  hello.txt  employees.csv
```

PowerShell

```
PS C:\> dir C:\TestDir

Directory: C:\TestDir

Mode                LastWriteTime         Length Name
----                -
-a----            9/12/2017   3:11 PM         2139 cool-new-thing.py
-a----            9/15/2017   5:57 PM           11 Hello.txt
-a----            9/17/2017   1:44 PM           46 Employees.csv
```

Der erste Befehl speichert das [Amazon Resource Name \(ARN, Amazon-Ressourcename\)](#) von einem AWS KMS key in `$keyArnVariable`.

Der zweite Befehl verschlüsselt den Inhalt von Dateien im TestDir-Verzeichnis und schreibt die Dateien mit dem verschlüsselten Inhalt in das TestEnc-Verzeichnis. Wenn das TestEnc-Verzeichnis nicht vorhanden ist, schlägt der Befehl fehl. Da der Eingabespeicherort ein Verzeichnis ist, muss der `--recursive`-Parameter angegeben werden.

Die [--wrapping-keys](#) Einstellung, und es ist erforderlich Schlüssel, geben Sie den zu verwendenden Umbruchschlüssel an. Der Verschlüsselungsbefehl umfasst einen [Verschlüsselungskontext](#), `dept=IT`. Wenn Sie einen Verschlüsselungskontext in einem Befehl angeben, der mehrere Dateien verschlüsselt, wird für alle Dateien derselbe Verschlüsselungskontext verwendet.

Der Befehl hat auch eine `--metadata-output` Parameter, um das zu sagen AWS Verschlüsselungs-CLI, wohin sie die Metadaten über die Verschlüsselungsoperationen schreiben soll. Die AWS Die Verschlüsselungs-CLI schreibt einen Metadaten-Datensatz für jede Datei, die verschlüsselt wurde.

Die `--commitment-policy parameter` ist ab Version 2.1 optional, aber es wird empfohlen. Wenn der Befehl oder das Skript fehlschlägt, weil es einen Chiffretext nicht entschlüsseln kann, hilft Ihnen die Einstellung der expliziten Verpflichtungsrichtlinie möglicherweise, das Problem schnell zu erkennen.

Wenn der Befehl abgeschlossen ist, wird die AWS Encryption CLI die verschlüsselten Dateien in die `TestEnc`-Verzeichnis, gibt aber keine Ausgabe zurück.

Der letzte Befehl listet die Dateien im Verzeichnis `TestEnc` auf. Es gibt eine Ausgabedatei des verschlüsselten Inhalts für jede Eingabedatei mit Klartext-Inhalt. Da der Befehl kein alternatives Suffix angegeben hat, hat der Verschlüsselungsbefehl `.encrypted` an jeden der jede der Eingabedateinamen angefügt.

Bash

```
# To run this example, replace the fictitious key ARN with a valid master key
  identifier.
$ keyArn=arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

$ aws-encryption-cli --encrypt \
    --input testdir --recursive\
    --wrapping-keys key=$keyArn \
    --encryption-context dept=IT \
    --commitment-policy require-encrypt-require-decrypt \
    --metadata-output ~/metadata \
    --output testenc

$ ls testenc
cool-new-thing.py.encrypted  employees.csv.encrypted  hello.txt.encrypted
```

PowerShell

```
# To run this example, replace the fictitious key ARN with a valid master key
  identifier.
PS C:\> $keyArn = arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

PS C:\> aws-encryption-cli --encrypt `
    --input .\TestDir --recursive `
    --wrapping-keys key=$keyArn `
    --encryption-context dept=IT `
```

```
--commitment-policy require-encrypt-require-decrypt `
--metadata-output .\Metadata\Metadata.txt `
--output .\TestEnc
```

```
PS C:\> dir .\TestEnc
```

```
Directory: C:\TestEnc
```

Mode	LastWriteTime	Length	Name
-a----	9/17/2017 2:32 PM	2713	cool-new-thing.py.encrypted
-a----	9/17/2017 2:32 PM	620	Hello.txt.encrypted
-a----	9/17/2017 2:32 PM	585	Employees.csv.encrypted

Alle Dateien in einem Verzeichnis entschlüsseln

In diesem Beispiel werden alle Dateien in einem Verzeichnis entschlüsselt. Es beginnt mit den Dateien im Verzeichnis TestEnc, die im vorherigen Beispiel verschlüsselt wurden.

Bash

```
$ ls testenc
cool-new-thing.py.encrypted hello.txt.encrypted employees.csv.encrypted
```

PowerShell

```
PS C:\> dir C:\TestEnc
```

```
Directory: C:\TestEnc
```

Mode	LastWriteTime	Length	Name
-a----	9/17/2017 2:32 PM	2713	cool-new-thing.py.encrypted
-a----	9/17/2017 2:32 PM	620	Hello.txt.encrypted
-a----	9/17/2017 2:32 PM	585	Employees.csv.encrypted

Dieser Entschlüsselungsbefehl entschlüsselt alle Dateien im TestEnc-Verzeichnis und schreibt die Klartext-Dateien in die TestDec-Verzeichnis. Die --wrapping-keys Parameterwert mit einem SchlüsselAttribut und ein [Schlüssel-ARN](#) value sagt AWS Verschlüsselung CLI welche AWS KMS keysum die Dateien zu entschlüsseln. Der Befehl verwendet die --interactive Parameter, um

das zu sagen AWS Verschlüsselungs-CLI, die Sie auffordert, eine Datei mit demselben Namen zu überschreiben.

Dieser Befehl verwendet auch den Verschlüsselungskontext, die bereitgestellt wurde, als die Dateien verschlüsselt wurden. Beim Entschlüsseln von mehreren Dateien wird die AWS Die Verschlüsselungs-CLI überprüft den Verschlüsselungskontext jeder Datei. Wenn die Überprüfung des Verschlüsselungskontextes für eine Datei fehlschlägt, wird der AWS Die Verschlüsselungs-CLI lehnt die Datei ab, schreibt eine Warnung, zeichnet den Ausfall in den Metadaten auf und fährt dann mit der Überprüfung der verbleibenden Dateien fort. Wenn das Symbol AWS Die Verschlüsselungs-CLI kann eine Datei aus anderen Gründen nicht entschlüsseln, der gesamte Entschlüsselungsbefehl schlägt sofort fehl.

In diesem Beispiel enthalten die verschlüsselten Nachrichten in allen Eingabedateien das Verschlüsselungskontextelement `dept=IT`. Wenn Sie jedoch Nachrichten mit unterschiedlichen Verschlüsselungskontexten entschlüsseln, können Sie möglicherweise immer noch einen Teil des Verschlüsselungskontexts überprüfen. Hatten beispielsweise einige Nachrichten den Verschlüsselungskontext `dept=finance`, andere hatten `dept=IT`, könnten Sie überprüfen, ob der Verschlüsselungskontext immer einen `dept`-Namen enthält, ohne den Wert anzugeben. Wenn Sie spezifischer vorgehen möchten, können Sie die Dateien in separaten Befehlen entschlüsseln.

Die Entschlüsselungsbefehl gibt keine Ausgabe zurück, aber Sie können einen Verzeichnislistenbefehl verwenden, um zu prüfen, ob der Befehl neue Dateien mit dem Suffix `.decrypted` erstellt hat. Um den Klartextinhalt anzuzeigen, verwenden Sie einen Befehl, um den Dateiinhalt abzurufen.

Bash

```
# To run this example, replace the fictitious key ARN with a valid master key
  identifier.
$ keyArn=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

$ aws-encryption-cli --decrypt \
    --input testenc --recursive \
    --wrapping-keys key=$keyArn \
    --encryption-context dept=IT \
    --commitment-policy require-encrypt-require-decrypt \
    --metadata-output ~/metadata \
    --max-encrypted-data-keys 1 \
    --buffer \
    --output testdec --interactive
```

```
$ ls testdec
cool-new-thing.py.encrypted.decrypted  hello.txt.encrypted.decrypted
employees.csv.encrypted.decrypted
```

PowerShell

```
# To run this example, replace the fictitious key ARN with a valid master key
  identifier.
PS C:\> $keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

PS C:\> aws-encryption-cli --decrypt `
        --input C:\TestEnc --recursive `
        --wrapping-keys key=$keyArn `
        --encryption-context dept=IT `
        --commitment-policy require-encrypt-require-decrypt `
        --metadata-output $home\Metadata.txt `
        --max-encrypted-data-keys 1 `
        --buffer `
        --output C:\TestDec --interactive

PS C:\> dir .\TestDec

            Mode                LastWriteTime         Length Name
-----
-a----          10/8/2017   4:57 PM             2139 cool-new-
thing.py.encrypted.decrypted
-a----          10/8/2017   4:57 PM              46 Employees.csv.encrypted.decrypted
-a----          10/8/2017   4:57 PM              11 Hello.txt.encrypted.decrypted
```

Verschlüsseln und Entschlüsseln in der Befehlszeile

Diese Beispiele zeigen Ihnen, wie Sie Eingaben an Befehle übergeben (stdin) und Ausgaben in die Befehlszeile schreiben (stdout). Sie erklären, wie stdin und stdout in einem Befehl dargestellt werden, und wie die eingebauten Base64-Codierungstools verwendet werden, um zu verhindern, dass die Shell Nicht-ASCII-Zeichen falsch interpretiert.

Dieses Beispiel übergibt eine Klartext-Zeichenfolge an einen Verschlüsselungsbefehl und speichert die verschlüsselte Nachricht in einer Variablen. Dann übergibt es die verschlüsselte Nachricht in der Variablen an einen Entschlüsselungsbefehl, der seine Ausgabe in die Pipeline schreibt (stdout).

Das Beispiel besteht aus drei Befehlen:

- Der erste Befehl speichert das [Schlüssel-ARN](#) von einem AWS KMS key in `$keyArn`.

Bash

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

PowerShell

```
PS C:\> $keyArn = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
```

- Der zweite Befehl übergibt die Zeichenfolge `Hello World` an den Verschlüsselungsbefehl weiter und speichert das Ergebnis in der Variablen `$encrypted`.

Die `--input` und `--output`-Parameter sind in allen erforderlich AWS Verschlüsselungs-CLI-Befehle. Um anzugeben, dass eine Eingabe dem Befehl übergeben wird (stdin), verwenden Sie einen Bindestrich (-) für den Wert des `--input`-Parameters. Um die Ausgabe in die Befehlszeile zu senden (stdout), verwenden Sie einen Bindestrich für den Wert des `--output`-Parameters.

Der `--encode`-Parameter bewirkt, dass die Ausgabe mit Base64 codiert wird, bevor sie zurückgegeben wird. Dadurch wird verhindert, dass die Shell interpretiert die Nicht-ASCII-Zeichen in der verschlüsselten Nachricht falsch interpretiert.

Da dieser Befehl nur ein Machbarkeitsnachweis ist, lassen wir den Verschlüsselungskontext weg und unterdrücken die Metadaten (`-S`).

Bash

```
$ encrypted=$(echo 'Hello World' | aws-encryption-cli --encrypt -S \
--input - --output - --
encode \
--wrapping-keys key=
$keyArn )
```

PowerShell

```
PS C:\> $encrypted = 'Hello World' | aws-encryption-cli --encrypt -S `
```



```

encode `
$keyArn
--input - --output - --
--wrapping-keys key=

```

- Der dritte Befehl übergibt die verschlüsselte Nachricht in der Variablen `$encrypted` an den Entschlüsselungsbefehl.

Dieser Entschlüsselungsbefehl verwendet `--input -`, um anzugeben, dass die Eingabe aus der Pipeline (stdin) kommt, und `--output -`, um die Ausgabe in die Pipeline zu senden (stdout). (Der Eingabeparameter verwendet den Speicherort der Eingabe, nicht die tatsächlich eingegebenen Bytes, Sie können also die Variable `$encrypted` nicht als Wert des `--input`-Parameters verwenden.)

In diesem Beispiel verwenden wir den `Entdeckung`Attribut der `--wrapping-keys`-Parameter, um die AWS Verschlüsselungs-CLI zur Verwendung eines beliebigen AWS KMS Key um die Daten zu entschlüsseln. Es gibt kein [Verpflichtungspolitik](#) verwendet daher den Standardwert für Version 2.1.x und höher `require-encrypt-require-decrypt` aus.

Da die Ausgabe verschlüsselt und dann codiert wurde, verwendet der Entschlüsselungsbefehl den `--decode`-Parameter zum Decodieren der mit Base64 codierten Eingabe, bevor sie entschlüsselt wird. Sie können auch den `--decode`-Parameter verwenden, um mit Base64 codierte Eingaben zu decodieren, bevor sie verschlüsselt werden.

Auch hier lässt der Befehl den Verschlüsselungskontext weg und unterdrückt die Metadaten (`-S`).

Bash

```

$ echo $encrypted | aws-encryption-cli --decrypt --wrapping-keys discovery=true
--input - --output - --decode --buffer -S
Hello World

```

PowerShell

```

PS C:\> $encrypted | aws-encryption-cli --decrypt --wrapping-keys discovery=$true
--input - --output - --decode --buffer -S
Hello World

```

Sie können die Ver- und Entschlüsselungsoperationen auch in einem einzigen Befehl ohne die dazwischenliegende Variable durchführen.

Wie im vorherigen Beispiel haben die Parameter `--input` und `--output` einen `--Wert`, und der Befehl verwendet den `--encode`-Parameter, um die Ausgabe zu codieren, und den `--decode` Parameter, um die Eingabe zu decodieren.

Bash

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

$ echo 'Hello World' |
    aws-encryption-cli --encrypt --wrapping-keys key=$keyArn --input - --
output - --encode -S |
    aws-encryption-cli --decrypt --wrapping-keys discovery=true --input - --
output - --decode -S
Hello World
```

PowerShell

```
PS C:\> $keyArn = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

PS C:\> 'Hello World' |
    aws-encryption-cli --encrypt --wrapping-keys key=$keyArn --input - --
output - --encode -S |
    aws-encryption-cli --decrypt --wrapping-keys discovery=$true --input
- --output - --decode -S
Hello World
```

Verwenden mehrerer Hauptschlüssel

Dieses Beispiel zeigt, wie mehrere Master-Schlüssel beim Ver- und Entschlüsseln von Daten im AWS CLI für Verschlüsselung.

Wenn Sie mehrere Master-Schlüssel verwenden, um Daten zu verschlüsseln, kann einer der Master-Schlüssel verwendet werden, um die Daten zu entschlüsseln. Diese Strategie stellt sicher, dass Sie die Daten entschlüsseln können, selbst wenn einer der Master-Schlüssel nicht verfügbar ist. Wenn Sie die verschlüsselten Daten in mehreren speichern AWS-Regionen Mit dieser Strategie können Sie einen Master-Schlüssel in derselben Region verwenden, um die Daten zu entschlüsseln.

Wenn Sie mit mehreren Master-Schlüsseln verschlüsseln, spielt der erste Master-Schlüssel eine spezielle Rolle. Er generiert den Datenschlüssel, der zum Verschlüsseln der Daten verwendet wird. Die verbleibenden Master-Schlüssel verschlüsseln den Klartext-Datenschlüssel. Die resultierende [verschlüsselte Nachricht](#) enthält die verschlüsselten Daten und eine Sammlung von verschlüsselten Datenschlüsseln, einen für jeden Master-Schlüssel. Obwohl der erste Master-Schlüssel den Datenschlüssel generiert hat, kann jeder der Master-Schlüssel einen der Datenschlüssel entschlüsseln, der verwendet werden kann, um die Daten zu entschlüsseln.

Verschlüsseln mit drei Hauptschlüsseln

Dieser Beispielbefehl verwendet drei Wrapping Keys, um die `Finance.log` Datei, jeweils eine von drei AWS-Regionen aus.

Er schreibt die verschlüsselte Nachricht in das Verzeichnis `Archive`. Der Befehl verwendet den `--suffix`-Parameter ohne Wert, um das Suffix zu unterdrücken, sodass die Eingabe- und Ausgabedateinamen gleich sind.

Der Befehl verwendet den `--wrapping-keys`-Parameter mit drei Schlüsselattributen. Sie können auch mehrere `--wrapping-keys`-Parameter im selben Befehl verwenden.

Um die Protokolldatei zu verschlüsseln, wird die AWS Encryption CLI fragt den ersten Umschlagschlüssel in der Liste, `$key1`, um den Datenschlüssel zu generieren, der zum Verschlüsseln der Daten verwendet wird. Anschließend verwendet sie jeden der anderen Umschlagschlüssel zum Verschlüsseln einer Klartext-Kopie desselben Datenschlüssels. Die verschlüsselte Nachricht in der Ausgabedatei enthält alle drei verschlüsselten Datenschlüssel.

Bash

```
$ key1=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
$ key2=arn:aws:kms:us-east-2:111122223333:key/0987ab65-43cd-21ef-09ab-87654321cdef
$ key3=arn:aws:kms:ap-
southeast-1:111122223333:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d

$ aws-encryption-cli --encrypt --input /logs/finance.log \
                    --output /archive --suffix \
                    --encryption-context class=log \
                    --metadata-output ~/metadata \
                    --wrapping-keys key=$key1 key=$key2 key=$key3
```

PowerShell

```

PS C:\> $key1 = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
PS C:\> $key2 = 'arn:aws:kms:us-east-2:111122223333:key/0987ab65-43cd-21ef-09ab-87654321cdef'
PS C:\> $key3 = 'arn:aws:kms:ap-southeast-1:111122223333:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d'

PS C:\> aws-encryption-cli --encrypt --input D:\Logs\Finance.log `
    --output D:\Archive --suffix `
    --encryption-context class=log `
    --metadata-output $home\Metadata.txt `
    --wrapping-keys key=$key1 key=$key2 key=$key3

```

Dieser Befehl entschlüsselt die verschlüsselte Kopie der Datei `Finance.log` und schreibt sie in eine `Finance.log.clear`-Datei im Verzeichnis `Finance`. Um Daten zu entschlüsseln, die unter drei verschlüsselt sind AWS KMS keys können Sie dieselben drei angeben AWS KMS keys oder irgendeine Teilmenge von ihnen. Dieses Beispiel gibt nur einen der AWS KMS keys aus.

Um das zu sagen AWS Verschlüsselung CLI welche AWS KMS keys Um Ihre Daten zu entschlüsseln, verwenden Sie die Schlüsselattribut der `--wrapping-keys`-Parameter. Beim Entschlüsseln mit AWS KMS keys, der Wert des Schlüssel-Attribut muss ein [Schlüssel-ARN](#) aus.

Sie benötigen die Berechtigung zum Aufrufen der [API entschlüsseln](#) auf der AWS KMS keys Sie geben an. Weitere Informationen finden Sie unter [Authentifizierung und -Zugriffskontrolle für AWS KMS](#) aus.

Als bewährte Methode verwenden diese Beispiele die `--max-encrypted-data-keys`-Parameter, um zu vermeiden, dass eine fehlerhafte Nachricht mit einer übermäßigen Anzahl verschlüsselter Datenschlüssel entschlüsselt wird. Obwohl in diesem Beispiel nur ein Umschlagschlüssel für die Entschlüsselung verwendet wird, enthält die verschlüsselte Nachricht drei (3) verschlüsselte Datenschlüssel; einen für jeden der drei beim Verschlüsseln verwendeten Wrapping Keys. Geben Sie die erwartete Anzahl verschlüsselter Datenschlüssel oder einen angemessenen Maximalwert an, z. B. Wenn Sie einen Maximalwert von weniger als 3 angeben, schlägt der Befehl fehl. Details hierzu finden Sie unter [Beschränkung verschlüsselter Datenschlüssel](#).

Bash

```
$ aws-encryption-cli --decrypt --input /archive/finance.log \
```

```
--wrapping-keys key=$key1 \  
--output /finance --suffix '.clear' \  
--metadata-output ~/metadata \  
--max-encrypted-data-keys 3 \  
--buffer \  
--encryption-context class=log
```

PowerShell

```
PS C:\> aws-encryption-cli --decrypt \  
--input D:\Archive\Finance.log \  
--wrapping-keys key=$key1 \  
--output D:\Finance --suffix '.clear' \  
--metadata-output .\Metadata\Metadata.txt \  
--max-encrypted-data-keys 3 \  
--buffer \  
--encryption-context class=log
```

Verschlüsseln und Entschlüsseln in Skripten

Dieses Beispiel veranschaulicht, wie Sie die verwenden AWS Verschlüsselungs-CLI in Skripten. Sie können Skripte schreiben, die nur Daten verschlüsseln und entschlüsseln, oder Skripte, die als Teil eines Datenverwaltungsprozesses verschlüsseln oder entschlüsseln.

In diesem Beispiel ruft das Skript eine Sammlung von Protokolldateien ab, komprimiert sie, verschlüsselt sie und kopiert die verschlüsselten Dateien in einen Amazon S3 S3-Bucket. Dieses Skript verarbeitet jede Datei einzeln, sodass Sie sie unabhängig voneinander entschlüsseln und erweitern können.

Wenn Sie Dateien komprimieren und verschlüsseln, stellen Sie sicher, dass Sie sie komprimieren, bevor Sie sie verschlüsseln. Ordnungsgemäß verschlüsselte Daten können nicht komprimiert werden.

Warning

Seien Sie vorsichtig, wenn Sie Daten komprimieren, die Geheimnisse oder Daten enthalten, die von böswilligen Angreifern kontrolliert werden könnten. Die endgültige Größe der komprimierten Daten verrät möglicherweise versehentlich vertrauliche Informationen über ihren Inhalt.

Bash

```
# Continue running even if an operation fails.
set +e

dir=$1
encryptionContext=$2
s3bucket=$3
s3folder=$4
masterKeyProvider="aws-kms"
metadataOutput="/tmp/metadata-$(date +%s)"

compress(){
    gzip -qf $1
}

encrypt(){
    # -e encrypt
    # -i input
    # -o output
    # --metadata-output unique file for metadata
    # -m masterKey read from environment variable
    # -c encryption context read from the second argument.
    # -v be verbose
    aws-encryption-cli -e -i ${1} -o $(dirname ${1}) --metadata-output
    ${metadataOutput} -m key="${masterKey}" provider="${masterKeyProvider}" -c
    "${encryptionContext}" -v
}

s3put (){
    # copy file argument 1 to s3 location passed into the script.
    aws s3 cp ${1} ${s3bucket}/${s3folder}
}

# Validate all required arguments are present.
if [ "${dir}" ] && [ "${encryptionContext}" ] && [ "${s3bucket}" ] &&
  [ "${s3folder}" ] && [ "${masterKey}" ]; then

# Is $dir a valid directory?
test -d "${dir}"
if [ $? -ne 0 ]; then
    echo "Input is not a directory; exiting"
    exit 1
fi
fi
```

```

fi

# Iterate over all the files in the directory, except *.gz and *encrypted (in case of
# a re-run).
for f in $(find ${dir} -type f \( -name "*" ! -name \*.gz ! -name \*encrypted \) );
do
    echo "Working on $f"
    compress ${f}
    encrypt ${f}.gz
    rm -f ${f}.gz
    s3put ${f}.gz.encrypted
done;
else
    echo "Arguments: <Directory> <encryption context> <s3://bucketname> <s3 folder>"
    echo " and ENV var \${masterKey} must be set"
    exit 255
fi

```

PowerShell

```

#Requires -Modules AWSPowerShell, Microsoft.PowerShell.Archive
Param
(
    [Parameter(Mandatory)]
    [ValidateScript({Test-Path $_})]
    [String[]]
    $FilePath,

    [Parameter()]
    [Switch]
    $Recurse,

    [Parameter(Mandatory=$true)]
    [String]
    $wrappingKeyID,

    [Parameter()]
    [String]
    $masterKeyProvider = 'aws-kms',

    [Parameter(Mandatory)]
    [ValidateScript({Test-Path $_})]
    [String]

```

```

    $ZipDirectory,

    [Parameter(Mandatory)]
    [ValidateScript({Test-Path $_})]
    [String]
    $EncryptDirectory,

    [Parameter()]
    [String]
    $EncryptionContext,

    [Parameter(Mandatory)]
    [ValidateScript({Test-Path $_})]
    [String]
    $MetadataDirectory,

    [Parameter(Mandatory)]
    [ValidateScript({Test-S3Bucket -BucketName $_})]
    [String]
    $S3Bucket,

    [Parameter()]
    [String]
    $S3BucketFolder
)

BEGIN {}
PROCESS {
    if ($files = dir $FilePath -Recurse:$Recurse)
    {

        # Step 1: Compress
        foreach ($file in $files)
        {
            $fileName = $file.Name
            try
            {
                Microsoft.PowerShell.Archive\Compress-Archive -Path $file.FullName -
DestinationPath $ZipDirectory\$filename.zip
            }
            catch
            {
                Write-Error "Zip failed on $file.FullName"
            }
        }
    }
}

```


Verwenden von Datenschlüssel-Caching

Dieses Beispiel verwendet das [Datenschlüssel-Caching](#) in einem Befehl, der eine große Anzahl von Dateien verschlüsselt.

Standardmäßig ist der AWS Encryption CLI (und andere Versionen des AWS Encryption SDK) generiert einen eindeutigen Datenschlüssel für jede Datei, die sie verschlüsselt. Obwohl die Verwendung eines eindeutigen Datenschlüssels für jede Operation ist eine bewährte Methode in der Kryptografie ist, ist eine begrenzte Wiederverwendung von Datenschlüsseln in einigen Situationen akzeptabel. Falls Sie ein Datenschlüssel-Caching in Betracht ziehen, beraten Sie sich mit einem Sicherheitstechniker, um die Sicherheitsanforderungen Ihrer Anwendung zu verstehen und die für Sie geeigneten Sicherheitsbarrieren zu bestimmen.

In diesem Beispiel beschleunigt das Datenschlüssel-Caching die Verschlüsselungsoperation, indem sie die Frequenz der Anfragen an den Masterschlüssel-Anbieter reduziert.

Der Befehl in diesem Beispiel verschlüsselt ein großes Verzeichnis mit mehreren Unterverzeichnissen, die insgesamt ungefähr 800 kleine Protokolldateien enthalten. Der erste Befehl speichert den ARN des AWS KMS key in einer `keyARN`-Variablen. Der zweite Befehl verschlüsselt alle Dateien im Eingabeverzeichnis (rekursiv) und schreibt sie in ein Archiv-Verzeichnis. Der Befehl verwendet den `--suffix` -Parameter, um das `.archive`-Suffix anzugeben.

Der `--caching`-Parameter aktiviert das Datenschlüssel-Caching. Die Attribut `capacity`, das die Anzahl der Datenschlüssel im Cache begrenzt, wird auf 1 gesetzt, da die serielle Dateiverarbeitung nie mehr als einen Datenschlüssel gleichzeitig verwendet. Das Attribut `max_age`, mit dem festgelegt wird, wie lange der Datenschlüssel im Cache verwendet werden kann, ist auf 10 Sekunden eingestellt.

Das optionale Attribut `max_messages_encrypted` ist auf 10 Nachrichten festgelegt, sodass eine einzelner Datenschlüssel nie verwendet werden kann, um mehr als 10 Dateien zu verschlüsseln. Die Begrenzung der Anzahl der von einem Datenschlüssel verschlüsselten Dateien reduziert die Anzahl der Dateien, die in dem unwahrscheinlichen Fall betroffen wären, wenn ein Datenschlüssel kompromittiert wird.

Um diesen Befehl für von Ihrem Betriebssystem generierte Protokolldateien auszuführen, müssen Sie möglicherweise Administratorberechtigungen besitzen (`sudo` in Linux; Als Administrator ausführen in Windows).

Bash

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

$ aws-encryption-cli --encrypt \
  --input /var/log/httpd --recursive \
  --output ~/archive --suffix .archive \
  --wrapping-keys key=$keyArn \
  --encryption-context class=log \
  --suppress-metadata \
  --caching capacity=1 max_age=10 max_messages_encrypted=10
```

PowerShell

```
PS C:\> $keyARN = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

PS C:\> aws-encryption-cli --encrypt `
  --input C:\Windows\Logs --recursive `
  --output $home\Archive --suffix '.archive' `
  --wrapping-keys key=$keyARN `
  --encryption-context class=log `
  --suppress-metadata `
  --caching capacity=1 max_age=10
max_messages_encrypted=10
```

Um die Wirkung des Datenschlüssel-Cachings zu testen, verwendet dieses Beispiel die [Measure-Command](#)-Cmdlet in PowerShell aus. Wenn Sie dieses Beispiel ohne Datenschlüssel-Caching ausführen, dauert seine Ausführung etwa 25 Sekunden. Dieser Prozess generiert einen neuen Datenschlüssel für jede Datei im Verzeichnis.

```
PS C:\> Measure-Command {aws-encryption-cli --encrypt `
  --input C:\Windows\Logs --recursive `
  --output $home\Archive --suffix '.archive' `
  --wrapping-keys key=$keyARN `
  --encryption-context class=log `
  --suppress-metadata }
```

```

Days           : 0
Hours          : 0
Minutes       : 0
Seconds       : 25
Milliseconds  : 453
Ticks         : 254531202
TotalDays     : 0.000294596298611111
TotalHours    : 0.00707031116666667
TotalMinutes  : 0.42421867
TotalSeconds  : 25.4531202
TotalMilliseconds : 25453.1202

```

Das Datenschlüssel-Caching beschleunigt das Verfahren, auch wenn Sie die einzelnen Datenschlüssel auf maximal 10 Dateien begrenzen. Der Befehl benötigt jetzt weniger als 12 Sekunden und reduziert die Anzahl der Aufrufe des Masterschlüssel-Anbieters auf ein Zehntel des ursprünglichen Werts.

```

PS C:\> Measure-Command {aws-encryption-cli --encrypt `
                                           --input C:\Windows\Logs --recursive `
                                           --output $home\Archive --suffix '.archive'
                                           `
                                           --wrapping-keys key=$keyARN `
                                           --encryption-context class=log `
                                           --suppress-metadata `
                                           --caching capacity=1 max_age=10
                                           max_messages_encrypted=10}

```

```

Days           : 0
Hours          : 0
Minutes       : 0
Seconds       : 11
Milliseconds  : 813
Ticks         : 118132640
TotalDays     : 0.000136727592592593
TotalHours    : 0.003281462222222222
TotalMinutes  : 0.1968877333333333
TotalSeconds  : 11.813264
TotalMilliseconds : 11813.264

```

Wenn Sie die `max_messages_encrypted`-Einschränkung weglassen, werden alle Dateien unter demselben Datenschlüssel verschlüsselt. Diese Änderung erhöht das Risiko einer

Wiederverwendung von Datenschlüsseln, ohne den Prozess sehr wesentlich schneller zu machen. Sie reduziert jedoch die Anzahl der Aufrufe des Masterschlüssel-Anbieters auf 1.

```
PS C:\> Measure-Command {aws-encryption-cli --encrypt `
    --input C:\Windows\Logs --recursive `
    --output $home\Archive --suffix '.archive' `
    --wrapping-keys key=$keyARN `
    --encryption-context class=log `
    --suppress-metadata `
    --caching capacity=1 max_age=10}

Days           : 0
Hours          : 0
Minutes       : 0
Seconds       : 10
Milliseconds  : 252
Ticks         : 102523367
TotalDays     : 0.000118661304398148
TotalHours    : 0.00284787130555556
TotalMinutes  : 0.170872278333333
TotalSeconds  : 10.2523367
TotalMilliseconds : 10252.3367
```

AWS Encryption SDK CLI – Syntax und Parameterreferenz

Dieses Thema enthält Syntaxdiagramme und kurze Parameterbeschreibungen, die Ihnen bei der Verwendung des AWS Encryption SDK Command Line Interface (CLI, Befehlszeilenschnittstelle) helfen. Hilfe zum Umschließen von Schlüsseln und anderen Parametern finden Sie unter [So verwenden Sie die AWS Encryption CLI](#). Beispiele finden Sie unter [Beispiele für die AWS CLI für Verschlüsselung](#). Eine vollständige Dokumentation finden Sie in [Read the Docs](#).

Themen

- [AWS Verschlüsselung, CLI-Syntax](#)
- [AWS Verschlüsselungs-CLI-Befehlszeilenparameter](#)
- [Erweiterte Parameter](#)

AWSVerschlüsselung, CLI-Syntax

Diese Syntaxdiagramme derAWS Verschlüsselungs-CLI zeigen die Syntax für jede Aufgabe, die Sie mit derAWS Verschlüsselungs-CLI ausführen. Sie stellen die empfohlene Syntax inAWS Encryption CLI Version 2.1 dar. x und später.

Neue Sicherheitsfunktionen wurden ursprünglich in den Versionen 1.7 derAWS Encryption CLI veröffentlicht. x und 2.0. x. AllerdingsAWS Encryption CLI Version 1.8. x ersetzt Version 1.7. x undAWS Encryption CLI 2.1. x ersetzt 2.0. x. Einzelheiten finden Sie in der entsprechenden [Sicherheitsempfehlung](#) im [aws-encryption-sdk-cli](#)Repository unter GitHub.

Note

Sofern in der Parameterbeschreibung nicht anders angegeben, kann jeder Parameter oder jedes Attribut in jedem Befehl nur einmal verwendet werden.
Wenn Sie ein Attribut verwenden, das ein Parameter nicht unterstützt, ignoriert dieAWS Verschlüsselungs-CLI dieses nicht unterstützte Attribut ohne Warnung oder Fehler.

Hilfe anfordern

Um die vollständigeAWS Verschlüsselungs-CLI-Syntax mit Parameterbeschreibungen zu erhalten, verwenden Sie`--help` oder`-h`.

```
aws-encryption-cli (--help | -h)
```

Die Version abrufen

Um die Versionsnummer IhrerAWS Encryption CLI-Installation zu erhalten, verwenden Sie`--version`. Geben Sie unbedingt die Version an, wenn Sie Fragen stellen, Probleme melden oder Tipps zur Verwendung derAWS Verschlüsselungs-CLI geben.

```
aws-encryption-cli --version
```

Daten verschlüsseln

Das folgende Syntaxdiagramm zeigt die Parameter, die ein `encrypt`-Befehl verwendet.

```
aws-encryption-cli --encrypt
```

```

--input <input> [--recursive] [--decode]
--output <output> [--interactive] [--no-overwrite] [--suffix
[<suffix>]] [--encode]
--wrapping-keys [--wrapping-keys] ...
    key=<keyID> [key=<keyID>] ...
    [provider=<provider-name>] [region=<aws-region>]
[profile=<aws-profile>]
--metadata-output <location> [--overwrite-metadata] | --suppress-
metadata]

[--commitment-policy <commitment-policy>]
[--encryption-context <encryption_context> [<encryption_context>
...]]

[--max-encrypted-data-keys <integer>]
[--algorithm <algorithm_suite>]
[--caching <attributes>]
[--frame-length <length>]
[-v | -vv | -vvv | -vvvv]
[--quiet]

```

Daten entschlüsseln

Das folgende Syntaxdiagramm zeigt die Parameter, die ein decrypt-Befehl verwendet.

In Version 1.8 x, der `--wrapping-keys` Parameter ist beim Entschlüsseln optional, wird aber empfohlen. Beginnend mit Version 2.1 x, der `--wrapping-keys` Parameter ist beim Verschlüsseln und Entschlüsseln erforderlich. Denn Sie können das Schlüsselattribut verwenden AWS KMS keys, um Wrapping-Schlüssel anzugeben (bewährte Methode), oder das Discovery-Attribut auf festgelegt `true`, wodurch die Wrapping-Schlüssel, die die AWS Verschlüsselungs-CLI verwenden kann, nicht eingeschränkt werden.

```

aws-encryption-cli --decrypt (or [--decrypt-unsigned])
--input <input> [--recursive] [--decode]
--output <output> [--interactive] [--no-overwrite] [--suffix
[<suffix>]] [--encode]
--wrapping-keys [--wrapping-keys] ...
    [key=<keyID>] [key=<keyID>] ...
    [discovery={true|false}] [discovery-partition=<aws-partition-
name>] [discovery-account=<aws-account-ID>] [discovery-account=<aws-account-ID>] ...]
    [provider=<provider-name>] [region=<aws-region>]
[profile=<aws-profile>]
--metadata-output <location> [--overwrite-metadata] | --suppress-
metadata]

[--commitment-policy <commitment-policy>]

```

```

...]]
    [--encryption-context <encryption_context> [<encryption_context>
    [--buffer]
    [--max-encrypted-data-keys <integer>]
    [--caching <attributes>]
    [--max-length <length>]
    [-v | -vv | -vvv | -vvvv]
    [--quiet]

```

Konfigurationsdateien verwenden

Sie können auf Konfigurationsdateien verweisen, die Parameter und deren Werte enthalten. Dies ist gleichwertig mit der Eingabe der Parameter und Werte im Befehl. Ein Beispiel finden Sie unter [Parameter in einer Konfigurationsdatei speichern](#).

```

aws-encryption-cli @<configuration_file>

# In a PowerShell console, use a backtick to escape the @.
aws-encryption-cli `@<configuration_file>

```

AWSVerschlüsselungs-CLI-Befehlszeilenparameter

Diese Liste enthält eine grundlegende Beschreibung der Befehlsparameter der AWS Verschlüsselungs-CLI. Eine vollständige Beschreibung finden Sie in der [aws-encryption-sdk-cli Dokumentation](#).

--encrypt (-e)

Verschlüsselt die Eingabedaten. Jeder Befehl muss einen `--decrypt-unsigned` Parameter `--encrypt`, oder `--decrypt`, oder haben.

--decrypt (-d)

Entschlüsselt die Eingabedaten. Jeder Befehl muss einen `--encrypt` `--decrypt`, oder `--decrypt-unsigned` -Parameter haben.

--decrypt-unsigned [Eingeführt in Versionen 1.9. x und 2.2. x]

Der `--decrypt-unsigned` Parameter entschlüsselt den Chiffretext und stellt sicher, dass Nachrichten vor der Entschlüsselung nicht signiert sind. Verwenden Sie diesen Parameter, wenn

Sie den `--algorithm` Parameter verwendet und eine Algorithmussuite ohne digitale Signatur zum Verschlüsseln von Daten ausgewählt haben. Wenn der Chiffretext signiert ist, schlägt die Entschlüsselung fehl.

Sie können `--decrypt` oder `--decrypt-unsigned` zur Entschlüsselung verwenden, aber nicht beides.

`--wrapping-keys (-w)` [In Version 1.8 eingeführt. x]

Gibt die [Wrapping-Schlüssel](#) (oder Master-Schlüssel) an, die bei Verschlüsselungs- und Entschlüsselungsvorgängen verwendet werden. Sie können in jedem Befehl [mehrere --wrapping-keys Parameter](#) verwenden.

Beginnend mit Version 2.1 x, der `--wrapping-keys` Parameter ist in Befehlen zum Verschlüsseln und Entschlüsseln erforderlich. In Version 1.8 x, Verschlüsselungsbefehle erfordern `--wrapping-keys` entweder einen `--master-keys` Oder-Parameter. In Version 1.8 x Befehle entschlüsseln, ein `--wrapping-keys` Parameter ist optional, wird aber empfohlen.

Wenn Sie einen benutzerdefinierten Master-Key-Anbieter verwenden, erfordern Verschlüsselungs- und Entschlüsselungsbefehle Schlüssel- und Anbieterattribute. Bei der Verwendung von AWS KMS keys Verschlüsselungsbefehlen ist ein Schlüsselattribut erforderlich. Entschlüsselungsbefehle erfordern ein Schlüsselattribut oder ein Discovery-Attribut mit einem Wert von `true` (aber nicht beiden). Die Verwendung des Schlüsselattributs beim Entschlüsseln ist eine [AWS Encryption SDK bewährte Methode](#). Dies ist besonders wichtig, wenn Sie Stapel unbekannter Nachrichten entschlüsseln, z. B. solche in einem Amazon S3 S3-Bucket oder einer Amazon SQS SQS-Warteschlange.

Ein Beispiel, das zeigt, wie Schlüssel für AWS KMS mehrere Regionen als Wrapping-Schlüssel verwendet werden, finden Sie unter [Verwenden Sie mehrere Regionen AWS KMS keys](#).

Attribute: Der Wert des `--wrapping-keys`-Parameters besteht aus den folgenden Attributen. Das Format ist `attribute_name=value`.

Schlüssel

Identifiziert den Wrapping-Schlüssel, der bei der Operation verwendet wurde. Das Format ist ein `key=ID-Paar`. Sie können mehrere `key`-Attribute in jedem `--wrapping-keys`-Parameterwert angeben.

- Verschlüsselungsbefehle: Alle Verschlüsselungsbefehle benötigen das Schlüsselattribut. Wenn Sie einen Befehl `AWS KMS key` in einem Schlüssel-ARN, kann der Wert des

Schlüssel-Attributs eine Schlüssel-ID, einen Schlüssel-ARN, einen Alias-Namen oder Alias-ARN sein. Eine Beschreibung der AWS KMS Schlüsselkennungen finden Sie unter [Schlüsselkennungen](#) im AWS Key Management Service Entwicklerhandbuch.

- Befehle entschlüsseln: Beim Entschlüsseln mit AWS KMS keys benötigt der `--wrapping-keys` Parameter ein Schlüsselattribut mit einem [Schlüssel-ARN-Wert](#) oder ein Discovery-Attribut mit einem Wert von `true` (aber nicht beiden). Die Verwendung des Schlüsselattributs ist eine [AWS Encryption SDK bewährte Methode](#). Bei der Entschlüsselung mit einem benutzerdefinierten Master-Key-Anbieter ist das Schlüsselattribut erforderlich.

Note

Um einen AWS KMS Wrapping-Schlüssel in einem Entschlüsselungsbefehl anzugeben, muss der Wert des Schlüsselattributs ein Schlüssel-ARN sein. Wenn Sie eine Schlüssel-ID, einen Alias-Namen oder einen Alias-ARN verwenden, erkennt die AWS Verschlüsselungs-CLI den Schlüssel-Schlüssel-Schlüssel-Schlüssel-Schlüssel-ARN nicht.

Sie können mehrere key-Attribute in jedem `--wrapping-keys`-Parameterwert angeben. Alle Anbieter-, Regions- und Profilattribute in einem `--wrapping-keys` Parameter gelten jedoch für alle Wrapping-Schlüssel in diesem Parameterwert. Verwenden Sie mehrere `--wrapping-keys` Parameter im Befehl, um Wrapping-Schlüssel mit unterschiedlichen Attributwerten anzugeben.

Entdeckung

Ermöglicht der AWS Verschlüsselungs-CLI, eine beliebige Methode AWS KMS key zum Entschlüsseln der Nachricht zu verwenden. Der Entdeckungswert kann `true` oder sein `false`. Der Standardwert ist `false`. Das Discovery-Attribut ist nur in Entschlüsselungsbefehlen gültig und nur, wenn der Master-Schlüsselanbieter dies ist AWS KMS.

Beim Entschlüsseln mit AWS KMS keys benötigt der `--wrapping-keys` Parameter ein Schlüsselattribut oder ein Discovery-Attribut mit einem Wert von `true` (aber nicht beiden). Wenn Sie das Schlüsselattribut verwenden, können Sie ein Discovery-Attribut mit dem Wert von verwenden, `false` um die Erkennung explizit abzulehnen.

- `false` (Standard) — Wenn das Discovery-Attribut nicht angegeben ist oder sein Wert ist `false`, entschlüsselt die AWS Verschlüsselungs-CLI die Nachricht nur mit dem durch den Schlüssel AWS KMS keys angegebenen Attribut des `--wrapping-keys` Parameters. Wenn Sie kein Schlüsselattribut angeben, wenn Discovery aktiviert ist `false`, schlägt der Befehl

decrypt fehl. Dieser Wert unterstützt eine [bewährte Methode](#) für die AWS Verschlüsselungs-CLI.

- `True`— Wenn der Wert des Discovery-Attributs auf `true` ist, ruft die AWS Verschlüsselungs-CLI die Metadaten AWS KMS keys aus der verschlüsselten Nachricht ab und verwendet diese, AWS KMS keys um die Nachricht zu entschlüsseln. Das Discovery-Attribut mit dem Wert `true` verhält sich wie Versionen der AWS Verschlüsselungs-CLI vor Version 1.8. x, das es Ihnen nicht erlaubte, beim Entschlüsseln einen Wrapping-Schlüssel anzugeben. Ihre Absicht, welche zu verwenden, AWS KMS key ist jedoch ausdrücklich. Wenn Sie bei Discovery ein Schlüsselattribut angeben `true`, schlägt der Entschlüsselungsbefehl fehl.

Der `true` Wert kann dazu führen, dass die AWS Verschlüsselungs-CLI AWS KMS keys in verschiedenen AWS-Konten Regionen verwendet wird oder versucht, eine Verwendung zu verwenden AWS KMS keys, für die der Benutzer nicht autorisiert ist.

Wenn Discovery aktiviert ist `true`, empfiehlt es sich, die Attribute `Discovery-Partition` und `Discovery-Account` zu verwenden, um die AWS KMS keys verwendeten Attribute auf die von AWS-Konten Ihnen angegebenen Attribute zu beschränken.

Discovery-Konto

Beschränkt die für die Entschlüsselung AWS KMS keys verwendeten Werte auf die angegebenen Werte AWS-Konto. Der einzig gültige Wert für dieses Attribut ist eine [AWS-KontoID](#).

Dieses Attribut ist optional und nur in Entschlüsselungsbefehlen gültig, bei AWS KMS keys denen das Discovery-Attribut auf `true` gesetzt und das `Discovery-Partition`-Attribut angegeben ist.

Jedes `Discovery-Account`-Attribut benötigt nur eine AWS-Konto ID, aber Sie können mehrere `Discovery-Account`-Attribute in demselben `--wrapping-keys` Parameter angeben. Alle in einem bestimmten `--wrapping-keys` Parameter angegebenen Konten müssen sich in der angegebenen AWS Partition befinden.

Discovery-Partition

Gibt die AWS Partition für die Konten im `Discovery-Account`-Attribut an. Sein Wert muss eine AWS Partition sein, z. B. `awsaws-cn`, oder `aws-gov-cloud`. Weitere Informationen finden Sie unter [Amazon-Ressourcennamen](#) in der Allgemeine AWS-Referenz.

Dieses Attribut ist erforderlich, wenn Sie das `Discovery-Account`-Attribut verwenden. Sie können in jedem `--wrapping keys` Parameter nur ein `Discovery-Partition`-Attribut angeben.

Verwenden Sie einen zusätzlichen `--wrapping-keys` Parameter, um AWS-Konten in mehreren Partitionen zu spezifizieren.

provider

Identifiziert den [Masterschlüssel-Anbieter](#). Das Format ist ein `provider=ID`-Paar. Der Standardwert ist `aws-kms` und steht für AWS KMS. Dieses Attribut ist nur erforderlich, wenn der Masterschlüssel-Anbieter nicht AWS KMS ist.

region

Identifiziert das AWS-Region von einem AWS KMS key. Dieses Attribut ist nur gültig für AWS KMS keys. Es wird nur verwendet, wenn die key-ID keine Region angibt, andernfalls wird es ignoriert. Wenn es verwendet wird, überschreibt es die Standardregion im benannten Profil der AWS-CLI.

Profil

Identifiziert ein AWS CLI [benanntes Profil](#). Dieses Attribut ist nur gültig für AWS KMS keys. Die Region im Profil wird nur verwendet, wenn die key-ID keine Region angibt und es kein `region-` Attribut im Befehl gibt.

`--input (-i)`

Gibt den Speicherort der zu ver- oder entschlüsselnden Daten an. Dieser Parameter muss angegeben werden. Der Wert kann ein Pfad zu einer Datei oder einem Verzeichnis oder ein Dateinamenmuster sein. Wenn Sie Eingaben an den Befehl weiterleiten (stdin), verwenden Sie `-`.

Wenn die Eingabe nicht vorhanden ist, wird der Befehl fehlerfrei ohne Fehlermeldung oder Warnung ausgeführt.

`--recursive (-r, -R)`

Führt die Operation für Dateien im Eingabeverzeichnis und seinen Unterverzeichnissen aus. Dieser Parameter ist erforderlich, wenn der Wert von `--input` ein Verzeichnis ist.

`--decode`

Decodiert Base64-codierte Eingaben.

Wenn Sie eine Nachricht entschlüsseln, die verschlüsselt und dann codiert wurde, müssen Sie die Nachricht decodieren, bevor Sie sie entschlüsseln. Dieser Parameter erledigt dies für Sie.

Wenn Sie beispielsweise den `--encode`-Parameter in einem Verschlüsselungsbefehl verwendet haben, verwenden Sie den `--decode`-Parameter in dem entsprechenden

Entschlüsselungsbefehl. Sie können diesen Parameter auch verwenden, um mit Base64 codierte Eingaben zu decodieren, bevor Sie sie verschlüsseln.

--output (-o)

Gibt einen Zielspeicherort für die Ausgabe an. Dieser Parameter muss angegeben werden. Der Wert kann ein Dateiname oder ein vorhandenes Verzeichnis sein, oder - sein, womit die Ausgabe in die Befehlszeile geschrieben wird (stdout).

Wenn das angegebene Ausgabeverzeichnis nicht vorhanden ist, schlägt der Befehl fehl. Wenn die Eingabe Unterverzeichnisse enthält, reproduziert die AWS Verschlüsselungs-CLI die Unterverzeichnisse unter dem von Ihnen angegebenen Ausgabeverzeichnis.

Standardmäßig überschreibt die AWS Verschlüsselungs-CLI Dateien mit demselben Namen. Dieses Verhalten ändern Sie mit den Parametern `--interactive` oder `--no-overwrite`. Um die Überschreibwarnung zu unterdrücken, verwenden Sie den Parameter `--quiet`.

Note

Wenn ein Befehl, der eine Ausgabedatei überschreiben würde, fehlschlägt, ist die Ausgabedatei bereits gelöscht.

--interactive

Informiert Sie, bevor die Datei überschrieben wird.

--no-overwrite

Überschreibt keine Dateien. Wenn die Ausgabedatei existiert, überspringt die AWS Verschlüsselungs-CLI stattdessen die entsprechende Eingabe.

--Suffix

Gibt ein benutzerdefiniertes Dateinamensuffix für Dateien an, die die AWS Verschlüsselungs-CLI erstellt. Wenn Sie kein Suffix angeben wollen, verwenden Sie den Parameter ohne Wert (`--suffix`).

Wenn der `--output`-Parameter keinen Dateinamen angibt, hat der Ausgabedateiname standardmäßig den gleichen Namen wie die Eingabedatei, jedoch mit dem Suffix. Das Suffix für Verschlüsselungsbefehle ist `.encrypted`. Das Suffix für Entschlüsselungsbefehle ist `.decrypted`.

--encode

Wendet die Base64-Codierung (binär in Text) auf die Ausgabe an. Die Kodierung verhindert, dass das Shell-Host-Programm Nicht-ASCII-Zeichen im Ausgabebetext falsch interpretiert.

Verwenden Sie diesen Parameter, wenn Sie verschlüsselte Ausgaben in stdout (`--output -`) schreiben, insbesondere in einer PowerShell Konsole, auch wenn Sie die Ausgabe an einen anderen Befehl weiterleiten oder in einer Variablen speichern.

--metadata-output

Gibt einen Speicherort für Metadaten über die kryptografischen Operationen an. Geben Sie einen Pfad und einen Dateinamen ein. Wenn das Verzeichnis nicht vorhanden ist, schlägt der Befehl fehl. Um die Metadaten in die Befehlszeile zu schreiben (stdout), verwenden Sie `-`.

Sie können die Befehlsausgabe (`--output`) und Metadatenausgaben (`--metadata-output`) nicht im selben Befehl auf stdout schreiben. Auch wenn der Wert von `--input` oder `--output` ein Verzeichnis (ohne Dateinamen) ist, können Sie die Metadatenausgabe nicht in das gleiche Verzeichnis oder in ein Unterverzeichnis dieses Verzeichnisses schreiben.

Wenn Sie eine vorhandene Datei angeben, hängt die AWS Verschlüsselungs-CLI standardmäßig neue Metadatenätze an alle Inhalte in der Datei an. Mit dieser Funktion können Sie eine einzige Datei erstellen, die die Metadaten für alle Ihre kryptografischen Operationen enthält. Um den Inhalt einer bestehenden Datei zu überschreiben, verwenden Sie den `--overwrite-metadata`-Parameter.

Die AWS Verschlüsselungs-CLI gibt für jeden Verschlüsselungs- oder Entschlüsselungsvorgang, den der Befehl ausführt, einen JSON-formatierten Metadatenatz zurück. Jeder Metadatenatz enthält die vollständigen Pfade zur Ein- und Ausgabedatei, den Verschlüsselungskontext, das Algorithmen-Paket und andere praktische Informationen, anhand derer Sie die Operation überprüfen und sicherstellen können, ob sie Ihren Sicherheitsstandards entspricht.

--overwrite-metadata

Überschreibt den Inhalt in der Metadaten-Ausgabedatei. Standardmäßig fügt der `--metadata-output`-Parameter Metadaten an vorhandenen Inhalt der Datei an.

--suppress-metadata (-S)

Unterdrückt die Metadaten über die Verschlüsselungs- oder Entschlüsselungsoperation.

--Engagementpolitik

Gibt die [Commitment-Richtlinie](#) für Verschlüsselungs- und Entschlüsselungsbefehle an. Die Commitment-Richtlinie legt fest, ob Ihre Nachricht mit der [Key Commitment-Sicherheitsfunktion](#) ver- und entschlüsselt wird.

Der `--commitment-policy` Parameter wurde in Version 1.8 eingeführt. x. Es ist gültig für Befehle zum Verschlüsseln und Entschlüsseln.

In Version 1.8. x, die AWS Verschlüsselungs-CLI verwendet die `forbid-encrypt-allow-decrypt` Commitment-Richtlinie für alle Verschlüsselungs- und Entschlüsselungsvorgänge. Wenn Sie den `--wrapping-keys` Parameter in einem Verschlüsselungs- oder Entschlüsselungsbefehl verwenden, ist ein `--commitment-policy` Parameter mit dem `forbid-encrypt-allow-decrypt` Wert erforderlich. Wenn Sie den `--wrapping-keys` Parameter nicht verwenden, ist der `--commitment-policy` Parameter ungültig. Durch die explizite Festlegung einer Commitment-Richtlinie wird verhindert, dass sich Ihre Commitment-Richtlinie automatisch ändert, `require-encrypt-require-decrypt` wenn Sie auf Version 2.1 aktualisieren. x

Beginnend mit Version 2.1 x, alle Werte der Engagementpolitik werden unterstützt. Der `--commitment-policy` Parameter ist optional und der Standardwert ist `require-encrypt-require-decrypt`.

Dieser Parameter hat die folgenden Werte:

- `forbid-encrypt-allow-decrypt`— Verschlüsselung mit Schlüsselzuweisung nicht möglich. Es kann Chiffretexte entschlüsseln, die mit oder ohne Schlüsselbindung verschlüsselt sind.

In Version 1.8 x, das ist der einzig gültige Wert. Die AWS Verschlüsselungs-CLI verwendet die `forbid-encrypt-allow-decrypt` Commitment-Richtlinie für alle Verschlüsselungs- und Entschlüsselungsvorgänge.

- `require-encrypt-allow-decrypt`— Verschlüsselt nur mit Schlüsselzuweisung. Entschlüsselt mit und ohne Schlüsselbindung. Dieser Wert wurde in Version 2.1 eingeführt. x.
- `require-encrypt-require-decrypt` (Standard) — Verschlüsselt und entschlüsselt nur mit Schlüsselzuweisung. Dieser Wert wurde in Version 2.1 eingeführt. x. Dies ist der Standardwert in den Versionen 2.1. x und später. Mit diesem Wert entschlüsselt die AWS Verschlüsselungs-CLI keinen Chiffretext, der mit früheren Versionen von verschlüsselt wurde AWS Encryption SDK.

Ausführliche Informationen zum Festlegen Ihrer Verpflichtungsrichtlinie finden Sie unter [Migrieren Sie Ihre AWS Encryption SDK](#).

`--encryption-context (-c)`

Gibt einen [Verschlüsselungskontext](#) für die Operation an. Dieser Parameter ist nicht erforderlich, wird jedoch empfohlen.

- In einem `--encrypt` -Befehl geben Sie ein oder mehrere `name=value` Paare an. Verwenden Sie Leerzeichen, um die zu trennen.
- Geben Sie in einem `--decrypt` Befehl `name=value` Paare, `name` Elemente ohne Werte oder beides ein.

Wenn der `name` oder `value` in einem `name=value`-Paar Leerzeichen oder Sonderzeichen enthält, schließen Sie gesamte Paar in Anführungszeichen ein. Zum Beispiel `--encryption-context "department=software development"`.

`--buffer (-b)` [Eingeführt in Versionen 1.9. x und 2.2. x]

Gibt Klartext erst zurück, nachdem alle Eingaben verarbeitet wurden, einschließlich der Überprüfung der digitalen Signatur, falls eine vorhanden ist.

`--max-encrypted-data-keys` [In den Versionen 1.9 eingeführt. x und 2.2. x]

Gibt die maximale Anzahl verschlüsselter Datenschlüssel in einer verschlüsselten Nachricht an. Dieser Parameter ist optional.

Gültige Werte sind 1 — 65.535. Wenn Sie diesen Parameter auslassen, erzwingt die AWS Verschlüsselungs-CLI kein Maximum. Eine verschlüsselte Nachricht kann bis zu 65.535 ($2^{16} - 1$) verschlüsselte Datenschlüssel enthalten.

Sie können diesen Parameter in Verschlüsselungsbefehlen verwenden, um eine fehlerhafte Nachricht zu verhindern. Sie können es in Entschlüsselungsbefehlen verwenden, um bösartige Nachrichten zu erkennen und zu vermeiden, dass Nachrichten mit zahlreichen verschlüsselten Datenschlüsseln entschlüsselt werden, die Sie nicht entschlüsseln können. Einzelheiten und ein Beispiel finden Sie unter [Beschränkung verschlüsselter Datenschlüssel](#).

`--help (-h)`

Gibt Verwendung und Syntax in der Befehlszeile aus.

`--version`

Ruft die Version der AWS Verschlüsselungs-CLI ab.


`-v | -vv | -vvv | -vvvv`

Zeigt ausführliche Informationen, Warnungen und Debugging-Nachrichten an. Die Details in der Ausgabe nehmen mit der Anzahl der `v` im Parameter zu. Die detaillierteste Einstellung (`-vvvv`) gibt Daten auf Debugging-Ebene aus der AWS Verschlüsselungs-CLI und allen verwendeten Komponenten zurück.

`--quiet (-q)`

Unterdrückt Warnmeldungen, z. B. die Nachricht, die angezeigt wird, wenn Sie eine Ausgabedatei überschreiben.

`--master-keys (-m) [Veraltet]`

 Note

Der Parameter `--master-keys` ist in 1.8 veraltet. `x` und in Version 2.1 entfernt. `x`. Verwenden Sie stattdessen den Parameter [--wrapping-keys](#).

Gibt die in Ver- und Entschlüsselungsoperationen verwendeten [Masterschlüssel](#) an. Sie können in einem Befehl mehrere Masterschlüsselparameter verwenden.

Der `--master-keys`-Parameter muss in Verschlüsselungsbefehle angegeben werden. Sie ist für Entschlüsselungsbefehle nur erforderlich, wenn Sie einen benutzerdefinierten (Nicht-AWS KMS) Master-Key-Anbieter verwenden.

Attribute: Der Wert des `--master-keys`-Parameters besteht aus den folgenden Attributen. Das Format ist `attribute_name=value`.

Schlüssel

Identifiziert den [Wrapping-Schlüssel](#), der bei der Operation verwendet wurde. Das Format ist ein `key=ID-Paar`. Das `key`-Attribut muss in allen Verschlüsselungsbefehlen angegeben werden.

Wenn Sie einen Befehl `AWS KMS key` in einem Schlüssel-ARN, kann der Wert des Schlüssel-Attributs eine Schlüssel-ID, einen Schlüssel-ARN, einen Alias-Namen oder Alias-ARN sein. Einzelheiten zu `AWS KMS Schlüsselkennungen` finden Sie unter [Schlüsselkennungen](#) im `AWS Key Management Service` `Entwicklerhandbuch`.

Das Schlüsselattribut ist in Entschlüsselungsbefehlen erforderlich, wenn der Master-Key-Anbieter dies nicht ist `AWS KMS`. Das Schlüsselattribut ist in Befehlen, die Daten entschlüsseln, die unter einem verschlüsselt wurden, nicht zulässig `AWS KMS key`.

Sie können mehrere `key-Attribute` in jedem `--master-keys`-Parameterwert angeben. Die `provider`-, `region`- und `profile`-Attribute gelten jedoch für alle Masterschlüssel im Parameterwert. Um Masterschlüssel mit unterschiedlichen Attributwerten anzugeben, verwenden Sie mehrere `--master-keys`-Parameter im Befehl.

`provider`

Identifiziert den [Masterschlüssel-Anbieter](#). Das Format ist ein `provider=ID`-Paar. Der Standardwert ist `aws-kms` und steht für AWS KMS. Dieses Attribut ist nur erforderlich, wenn der Masterschlüssel-Anbieter nicht AWS KMS ist.

`region`

Identifiziert die AWS-Region von einem AWS KMS key. Dieses Attribut ist nur gültig für AWS KMS keys. Es wird nur verwendet, wenn die `key-ID` keine Region angibt, andernfalls wird es ignoriert. Wenn es verwendet wird, überschreibt es die Standardregion im benannten Profil der AWS-CLI.

`Profile`

Identifiziert ein AWS CLI [benanntes Profil](#). Dieses Attribut ist nur gültig für AWS KMS keys. Die Region im Profil wird nur verwendet, wenn die `key-ID` keine Region angibt und es kein `region`-Attribut im Befehl gibt.

Erweiterte Parameter

`--algorithm`

Gibt ein alternatives [Algorithmen-Paket](#) an. Dieser Parameter ist optional und nur in Verschlüsselungsbefehlen gültig.

Wenn Sie diesen Parameter weglassen, verwendet die AWS Verschlüsselungs-CLI eine der Standard-Algorithmus-Suiten für die in Version 1.8 AWS Encryption SDK eingeführte. x. Beide Standardalgorithmen verwenden AES-GCM mit einem [HKDF](#), einer ECDSA-Signatur und einem 256-Bit-Verschlüsselungsschlüssel. Man setzt Schlüsselengagement ein, man tut es nicht. Die Wahl der Standard-Algorithmussuite wird durch die [Commitment-Richtlinie](#) für den Befehl bestimmt.

Die Standard-Algorithmus-Suites werden für die meisten Verschlüsselungsvorgänge empfohlen. Eine Liste gültiger Werte finden Sie unter den Werten für den `algorithm`-Parameter in [Read the Docs](#).

--frame-length

Erstellt die Ausgabe mit angegebenen Frame-Länge. Dieser Parameter ist optional und nur in Verschlüsselungsbefehlen gültig.

Geben Sie einen Wert in Bytes ein. Gültige Werte sind 0 und $1 - 2^{31} - 1$. Ein Wert von 0 zeigt an, dass Daten nicht gerahmt sind. Die Standardeinstellung ist 4096 (Byte).

Note

Verwenden Sie nach Möglichkeit gerahmte Daten. Der AWS Encryption SDK unterstützt ungerahmte Daten nur für ältere Zwecke. Einige Sprachimplementierungen von AWS Encryption SDK können immer noch Chiffretext ohne Rahmen generieren. Alle unterstützten Sprachimplementierungen können gerahmten und ungerahmten Chiffretext entschlüsseln.

--max-length

Gibt die maximale Frame-Größe (oder die maximale Inhaltslänge für Nachrichten ohne Frame) in Bytes an, die aus verschlüsselten Nachrichten gelesen werden. Dieser Parameter ist optional und nur in Entschlüsselungsbefehlen gültig. Es wurde entwickelt, um Sie vor der Entschlüsselung extrem großer bössartiger Verschlüsselungstexte zu schützen.

Geben Sie einen Wert in Bytes ein. Wenn Sie diesen Parameter AWS Encryption SDK nicht angeben, wird die Bildgröße beim Entschlüsseln nicht begrenzt.

--caching

Aktiviert das [Datenschlüssel-Caching](#), womit Datenschlüssel wiederverwendet werden können, statt für jede Eingabedatei einen neuen Datenschlüssel zu generieren. Dieser Parameter unterstützt ein erweitertes Szenario. Lesen Sie unbedingt die Dokumentation zum [Datenschlüssel-Caching](#), bevor Sie diese Funktion verwenden.

Der --caching-Parameter hat die folgenden Attribute.

capacity (erforderlich)

Legt die maximale Anzahl der Einträge im Cache fest.

Der minimale Wert beträgt 1. Es gibt keinen Höchstwert.

max_age (erforderlich)

Ermitteln Sie, wie lange Cache-Einträge in Sekunden verwendet werden, beginnend mit dem Hinzufügen zum Cache.

Geben Sie einen Wert größer als 0 ein. Es gibt keinen Höchstwert.

max_messages_encrypted (optional)

Legt die maximale Anzahl der Nachrichten fest, die ein im Cache befindlicher Eintrag verschlüsseln kann.

Gültige Werte sind $1 - 2^{32}$. Der Standardwert lautet 2^{32} (Nachrichten).

max_bytes_encrypted (optional)

Legt die maximale Anzahl der Bytes fest, die ein im Cache befindlicher Eintrag verschlüsseln kann.

Gültige Werte sind 0 und $1 - 2^{63} - 1$. Der Standardwert ist $2^{63} - 1$ (Nachrichten). Bei einem Wert von 0 können Sie Datenschlüssel-Caching nur verwenden, wenn Sie leere Nachrichtenzeichenfolgen verschlüsseln.

Versionen desAWSVerschlüsselung CLI

Wir empfehlen Ihnen, die neueste Version desAWSVerschlüsselungs-CLI.

Note

Versionen desAWSVerschlüsselungs-CLIs vor 4.0.0 befinden sich im [end-of-supportPhase](#). Sie können sicher von Version 2.1 aktualisieren und später zur neuesten Version desAWSVerschlüsselungs-CLI ohne Code- oder Datenänderungen. Allerdings [neue Sicherheitsfunktionen](#) eingeführt in Version 2.1 sind nicht abwärtskompatibel. Um von Version 1.7 zu aktualisieren oder früher müssen Sie zuerst auf die neueste Version 1 aktualisieren. Version desAWSVerschlüsselungs-CLI. Details hierzu finden Sie unter [Migrieren Sie IhreAWS Encryption SDK](#).

Neue Sicherheitsfunktionen wurden ursprünglich in veröffentlichtAWSEncryption-CLI-Versionen 1.7 und 2.0.x. AllerdingsAWSVerschlüsselung CLI Version 1.8 ersetzt Version 1.7 undAWSVerschlüsselung CLI 2.1 ersetzt 2.0.x. Details dazu finden Sie unter [Sicherheits-Beratung](#) in der [aws-encryption-sdk-cli](#)Repository auf GitHub.

Für Informationen über wichtige Versionen des AWS Encryption SDK, siehe [Versionen der AWS Encryption SDK](#).

Welche Version verwende ich?

Wenn für Sie neu ist AWS Encryption CLI, verwenden Sie die neueste Version.

So entschlüsseln Sie Daten, die mit einer Version des AWS Encryption SDK vor Version 1.7.x, migrieren zuerst auf die neueste Version des AWS Verschlüsselungs-CLI. Marke [alle empfohlenen Änderungen](#) vor dem Update auf Version 2.1.x oder später. Details hierzu finden Sie unter [Migrieren Sie Ihre AWS Encryption SDK](#).

Weitere Informationen

- Detaillierte Informationen zu den Änderungen und Anleitungen für die Migration auf diese neuen Versionen finden Sie unter [Migrieren Sie Ihre AWS Encryption SDK](#).
- Für Beschreibungen der neuen AWS Parameter und Attribute der Verschlüsselungs-CLI, siehe [AWS Encryption SDK CLI – Syntax und Parameterreferenz](#).

In den folgenden Listen wird die Änderung des AWS Verschlüsselungs-CLI in Version 1.8.x und 2.1.x.

Version 1.8.x Änderungen in der AWS Verschlüsselung CLI

- Veraltet den `-master-keys` Parameter. Verwenden Sie stattdessen den `--wrapping-keys` Parameter.
- Add die `--wrapping-keys(-w)` Parameter. Es unterstützt alle Attribute der `-master-keys` Parameter. Außerdem werden die folgenden optionalen Attribute hinzugefügt, die nur gültig sind, wenn mit AWS KMS keys.
 - Entdeckung
 - Discovery-Partition
 - Discovery-Konto

Für benutzerdefinierte Masterkey-Anbieter `--encrypt` und `--decrypt` Befehle erfordern entweder ein `--wrapping-keys` Parameter für den `-master-keys` Parameter (aber nicht beide). Außerdem ein `--encrypt` Befehl mit AWS KMS keys erfordert entweder ein `--wrapping-keys` Parameter für den `-master-keys` Parameter (aber nicht beide).

In einem `--decrypt` Befehl mit AWS KMS keys, das `--wrapping-keys` Parameter ist optional, wird aber empfohlen, da er in Version 2.1 erforderlich ist. Wenn Sie es verwenden, müssen Sie

entweder `SchlüsselAttribut` für den `-ParameterEntdeckungAttribut` mit einem Wert von `true` (aber nicht beide).

- Add die `--commitment-policy` Parameter. Der einzige gültige Wert ist `forbid-encrypt-allow-decrypt`. Die `forbid-encrypt-allow-decrypt` Die Commitment-Richtlinie wird in allen Befehlen zum Verschlüsseln und Entschlüsseln verwendet.

In Version 1.8x, wenn Sie die `--wrapping-keys` Parameter, den `-Parameter--commitment-policy` Parameter mit dem `forbid-encrypt-allow-decrypt` Wert ist erforderlich. Das explizite Festlegen des Werts verhindert [Committing-Politik](#) vom automatischen Wechsel zu `require-encrypt-require-decrypt` wenn auf Version 2.1 aktualisiert wird.

Version 2.1.x Änderungen in der AWS Verschlüsselung CLI

- Entfernt den `-Parameter--master-keys` Parameter. Verwenden Sie stattdessen den `--wrapping-keys` Parameter.
- Die `--wrapping-keys` ist in allen `Encrypt`- und `Decrypt`-Befehlen erforderlich. Sie müssen entweder angeben `SchlüsselAttribut` eines `-EntdeckungAttribut` mit einem Wert von `true` (aber nicht beide).
- Die `--commitment-policy` Parameter unterstützt die folgenden Werte. Details hierzu finden Sie unter [Festlegung Ihrer Verpflichtungspolitik](#).
 - `forbid-encrypt-allow-decrypt`
 - `require-encrypt-allow-decrypt`
 - `require-encrypt-require-decrypt` (Standard)
- Die `--commitment-policy` Parameter ist in Version 2.1 optional. Der Standardwert ist `require-encrypt-require-decrypt`.

Version 1.9.x und 2.2.x Änderungen in der AWS Verschlüsselung CLI

- Add die `--decrypt-unsigned` Parameter. Details hierzu finden Sie unter [Version 2.2. x](#).
- Add die `--buffer` Parameter. Details hierzu finden Sie unter [Version 2.2. x](#).
- Add die `--max-encrypted-data-keys` Parameter. Details hierzu finden Sie unter [Beschränkung verschlüsselter Datenschlüssel](#).

Version 3.0.x Änderungen in der AWS Verschlüsselung CLI

- Fügt Unterstützung hinzu für AWS KMS multiregionale Schlüssel. Details hierzu finden Sie unter [Verwenden Sie mehrere Regionen AWS KMS keys](#).

Datenschlüssel-Caching

Das Datenschlüssel-Caching speichert [Datenschlüssel](#) und zugehörige [kryptographische Materialien](#) in einem Cache. Wenn Sie Daten ver- oder entschlüsseln, AWS Encryption SDK sucht der nach einem passenden Datenschlüssel im Cache. Wenn eine Übereinstimmung gefunden wird, wird der im Cache gespeicherte Datenschlüssel verwendet, statt einen neuen zu generieren. Das Datenschlüssel-Caching kann die Leistung verbessern, die Kosten senken und Ihnen helfen, innerhalb der Service-Limits zu bleiben, wenn Ihre Anwendung größer wird.

Ihre Anwendung kann vom Datenschlüssel-Caching profitieren, wenn:

- Sie Datenschlüssel wiederverwenden kann.
- Sie zahlreiche Datenschlüssel generiert.
- Ihre kryptografischen Operationen unakzeptabel langsam, teuer, begrenzt oder ressourcen-intensiv sind.

Durch das Zwischenspeichern kann die Nutzung kryptografischer Dienste wie () reduziert werden. AWS Key Management Service AWS KMS Wenn Sie Ihr [AWS KMS requests-per-secondLimit](#) erreichen, kann Caching helfen. Ihre Anwendung kann zwischengespeicherte Schlüssel verwenden, um einige Ihrer Datenschüsselanforderungen zu bearbeiten, anstatt sie aufzurufen. AWS KMS(Sie können auch im [AWS Support Center](#) einen Fall erstellen, um das Limit für Ihr Konto zu erhöhen.)

Das AWS Encryption SDK hilft Ihnen dabei, Ihren Datenschlüssel-Cache zu erstellen und zu verwalten. [Es bietet einen lokalen Cache und einen Caching Cryptographic Materials Manager \(Caching CMM\), der mit dem Cache interagiert und die von Ihnen festgelegten Sicherheitsschwellenwerte durchsetzt.](#) Durch die Kombination dieser Komponenten profitieren Sie von der Effizienz der Wiederverwendung von Datenschlüsseln bei gleichzeitiger Wahrung der Sicherheit Ihres Systems.

Das Zwischenspeichern von Datenschlüsseln ist eine optionale Funktion von, die Sie mit Vorsicht verwenden sollten. AWS Encryption SDK Standardmäßig AWS Encryption SDK generiert der für jeden Verschlüsselungsvorgang einen neuen Datenschlüssel. Diese Technik unterstützt bewährte kryptografische Methoden gegen eine übermäßige Wiederverwendung von Datenschlüsseln. Im Allgemeinen sollten Sie das Datenschlüssel-Caching nur dann verwenden, wenn es erforderlich ist, um Ihre Leistungsziele zu erfüllen. Verwenden Sie dann die [Sicherheitsschwellenwerte](#) für das Datenschlüssel-Caching, um sicherzustellen, dass Sie die minimale Menge an Caching verwenden, die zur Erreichung Ihrer Kosten- und Leistungsziele erforderlich ist.

[Das Caching-CMM wird von for.NET nicht unterstützt.](#) [AWS Encryption SDK](#) Version 3. x of the unterstützt AWS-Verschlüsselungs-SDK for Java nur das Caching-CMM mit der alten Master-Key-Provider-Schnittstelle, nicht die Keyring-Schnittstelle. Version 4. x von AWS Encryption SDK für .NET und Version 3. x von ihnen AWS-Verschlüsselungs-SDK for Java unterstützt den [AWS KMS hierarchischen Schlüsselbund](#), eine alternative Lösung zum Zwischenspeichern kryptografischer Materialien. Mit dem hierarchischen Schlüsselbund verschlüsselte Inhalte können nur mit dem AWS KMS hierarchischen Schlüsselbund entschlüsselt werden. AWS KMS

Eine ausführliche Erläuterung dieser Sicherheitsaspekte finden Sie im Sicherheits-Blog unter [AWS Encryption SDK: So entscheiden Sie, ob das Zwischenspeichern von Datenschlüsseln für Ihre Anwendung geeignet ist.](#) AWS

Themen

- [Das Datenschlüssel-Caching verwenden](#)
- [Festlegen von Cache-Sicherheitsschwellenwerten](#)
- [Weitere Informationen zum Datenschlüssel-Caching](#)
- [Beispiel für das Datenschlüssel-Caching](#)

Das Datenschlüssel-Caching verwenden

In diesem Thema erfahren Sie, wie Sie das Datenschlüssel-Caching in Ihrer Anwendung verwenden. Es führt Sie Schritt für Schritt durch den Vorgang. Anschließend kombiniert es die Schritte in einem einfachen Beispiel, das das Datenschlüssel-Caching in einer Operation zum Verschlüsseln einer Zeichenfolge verwendet.

Die Beispiele in diesem Abschnitt zeigen, wie [Version 2.0 verwendet wird.](#) x und später von AWS Encryption SDK. Beispiele, die frühere Versionen verwenden, finden Sie in der [Release-Liste](#) des GitHub Repositories für Ihre [Programmiersprache](#).

Vollständige und getestete Beispiele für die Verwendung von Datenschlüssel-Caching in finden Sie AWS Encryption SDK unter:

- C/C++: [caching_cmm.cpp](#)
- Java: [SimpleDataKeyCachingExample.java](#)
- JavaScript [Browser: caching_cmm.ts](#)
- JavaScript Node.js: [caching_cmm.ts](#)

- Python: [data_key_caching_basic.py](#)

Das [AWS Encryption SDK für .NET](#) unterstützt das Zwischenspeichern von Datenschlüsseln nicht.

Themen

- [Verwenden von Datenschlüssel-Caching: S tep-by-step](#)
- [Beispiel für das Datenschlüssel-Caching: Verschlüsseln einer Zeichenfolge](#)

Verwenden von Datenschlüssel-Caching: S tep-by-step

Diese step-by-step Anweisungen zeigen Ihnen, wie Sie die Komponenten erstellen, die Sie für die Implementierung des Zwischenspeichers von Datenschlüsseln benötigen.

- [Erstellen Sie einen Datenschlüssel-Cache](#). In diesen Beispielen verwenden wir den lokalen Cache, den der AWS Encryption SDK bereitstellt. Wir beschränken den Cache auf zehn Datenschlüssel.

C

```
// Cache capacity (maximum number of entries) is required
size_t cache_capacity = 10;
struct aws_allocator *allocator = aws_default_allocator();

struct aws_cryptosdk_materials_cache *cache =
    aws_cryptosdk_materials_cache_local_new(allocator, cache_capacity);
```

Java

Das folgende Beispiel verwendet Version 2. x der AWS-Verschlüsselungs-SDK for Java. Ausführung 3. x von der AWS-Verschlüsselungs-SDK for Java missbilligt das Zwischenspeichern von Datenschlüsseln. CMM Mit Version 3. x, Sie können auch den [AWS KMS hierarchischen Schlüsselbund](#) verwenden, eine alternative Lösung zum Zwischenspeichern kryptografischer Materialien.

```
// Cache capacity (maximum number of entries) is required
int MAX_CACHE_SIZE = 10;

CryptoMaterialsCache cache = new LocalCryptoMaterialsCache(MAX_CACHE_SIZE);
```

JavaScript Browser

```
const capacity = 10

const cache = getLocalCryptographicMaterialsCache(capacity)
```

JavaScript Node.js

```
const capacity = 10

const cache = getLocalCryptographicMaterialsCache(capacity)
```

Python

```
# Cache capacity (maximum number of entries) is required
MAX_CACHE_SIZE = 10

cache = aws_encryption_sdk.LocalCryptoMaterialsCache(MAX_CACHE_SIZE)
```

- Erstellen Sie einen [Hauptschlüsselanbieter](#) (Java und Python) oder einen [Schlüsselbund](#) (C und JavaScript). In diesen Beispielen wird ein Hauptschlüsselanbieter AWS Key Management Service (AWS KMS) oder ein kompatibler [AWS KMS Schlüsselbund](#) verwendet.

C

```
// Create an AWS KMS keyring
// The input is the Amazon Resource Name (ARN)
// of an AWS KMS key
struct aws_cryptosdk_keyring *kms_keyring =
    Aws::Cryptosdk::KmsKeyring::Builder().Build(kms_key_arn);
```

Java

Das folgende Beispiel verwendet Version 2. x der AWS-Verschlüsselungs-SDK for Java. Ausführung 3. x von der AWS-Verschlüsselungs-SDK for Java missbilligt das Zwischenspeichern von Datenschlüsseln. CMM Mit Version 3. x, Sie können auch den

[AWS KMS hierarchischen Schlüsselbund](#) verwenden, eine alternative Lösung zum Zwischenspeichern kryptografischer Materialien.

```
// Create an AWS KMS master key provider
// The input is the Amazon Resource Name (ARN)
// of an AWS KMS key
MasterKeyProvider<KmsMasterKey> keyProvider =
    KmsMasterKeyProvider.builder().buildStrict(kmsKeyArn);
```

JavaScript Browser

Im Browser müssen Sie Ihre Anmeldeinformationen sicher injizieren. In diesem Beispiel werden Anmeldeinformationen in einem Webpack (`kms.webpack.config`) definiert, mit dem Anmeldeinformationen zur Laufzeit aufgelöst werden. Es erstellt eine AWS KMS Client-Provider-Instanz aus einem AWS KMS Client und den Anmeldeinformationen. Wenn es dann den Schlüsselbund erstellt, übergibt es den Client-Provider zusammen mit dem () an den Konstruktor. `AWS KMS key generatorKeyId`

```
const { accessKeyId, secretAccessKey, sessionToken } = credentials

const clientProvider = getClient(KMS, {
  credentials: {
    accessKeyId,
    secretAccessKey,
    sessionToken
  }
})

/* Create an AWS KMS keyring
 * You must configure the AWS KMS keyring with at least one AWS KMS key
 * The input is the Amazon Resource Name (ARN)
 */ of an AWS KMS key
const keyring = new KmsKeyringBrowser({
  clientProvider,
  generatorKeyId,
  keyIds,
})
```

JavaScript Node.js

```
/* Create an AWS KMS keyring
```

```
* The input is the Amazon Resource Name (ARN)
*/ of an AWS KMS key
const keyring = new KmsKeyringNode({ generatorKeyId })
```

Python

```
# Create an AWS KMS master key provider
# The input is the Amazon Resource Name (ARN)
# of an AWS KMS key
key_provider =
aws_encryption_sdk.StrictAwsKmsMasterKeyProvider(key_ids=[kms_key_arn])
```

- [Erstellen Sie einen Manager für das Zwischenspeichern von kryptografischen Materialien \(Caching\)](#). CMM

Ordnen Sie Ihr Caching CMM Ihrem Cache und Ihrem Hauptschlüsselanbieter oder Schlüsselbund zu. [Legen Sie dann Cache-Sicherheitsschwellenwerte für](#) das Caching fest. CMM

C

In der AWS-Verschlüsselungs-SDK for C können Sie ein Caching CMM aus einem BasiswertCMM, z. B. dem StandardCMM, oder aus einem Schlüsselbund erstellen. In diesem Beispiel wird das Caching CMM aus einem Schlüsselbund erstellt.

Nachdem Sie das Caching erstellt habenCMM, können Sie Ihre Verweise auf den Schlüsselbund und den Cache freigeben. Details hierzu finden Sie unter [the section called "Referenzählung"](#).

```
// Create the caching CMM
// Set the partition ID to NULL.
// Set the required maximum age value to 60 seconds.
struct aws_cryptosdk_cmm *caching_cmm =
aws_cryptosdk_caching_cmm_new_from_keyring(allocator, cache, kms_keyring, NULL,
60, AWS_TIMESTAMP_SECS);

// Add an optional message threshold
```

```
// The cached data key will not be used for more than 10 messages.
aws_status = aws_cryptosdk_caching_cmm_set_limit_messages(caching_cmm, 10);

// Release your references to the cache and the keyring.
aws_cryptosdk_materials_cache_release(cache);
aws_cryptosdk_keyring_release(kms_keyring);
```

Java

Das folgende Beispiel verwendet Version 2. x der AWS-Verschlüsselungs-SDK for Java. Ausführung 3. x of the unterstützt AWS-Verschlüsselungs-SDK for Java kein Zwischenspeichern von Datenschlüsseln, aber es unterstützt den [AWS KMS hierarchischen Schlüsselbund](#), eine alternative Lösung zum Zwischenspeichern kryptografischer Materialien.

```
/*
 * Security thresholds
 * Max entry age is required.
 * Max messages (and max bytes) per entry are optional
 */
int MAX_ENTRY_AGE_SECONDS = 60;
int MAX_ENTRY_MSGS = 10;

//Create a caching CMM
CryptoMaterialsManager cachingCmm =
    CachingCryptoMaterialsManager.newBuilder().withMasterKeyProvider(keyProvider)
        .withCache(cache)
        .withMaxAge(MAX_ENTRY_AGE_SECONDS,
            TimeUnit.SECONDS)
        .withMessageUseLimit(MAX_ENTRY_MSGS)
        .build();
```

JavaScript Browser

```
/*
 * Security thresholds
 * Max age (in milliseconds) is required.
 * Max messages (and max bytes) per entry are optional.
 */
const maxAge = 1000 * 60
const maxMessagesEncrypted = 10

/* Create a caching CMM from a keyring */
```

```
const cachingCmm = new WebCryptoCachingMaterialsManager({
  backingMaterials: keyring,
  cache,
  maxAge,
  maxMessagesEncrypted
})
```

JavaScript Node.js

```
/*
 * Security thresholds
 * Max age (in milliseconds) is required.
 * Max messages (and max bytes) per entry are optional.
 */
const maxAge = 1000 * 60
const maxMessagesEncrypted = 10

/* Create a caching CMM from a keyring */
const cachingCmm = new NodeCachingMaterialsManager({
  backingMaterials: keyring,
  cache,
  maxAge,
  maxMessagesEncrypted
})
```

Python

```
# Security thresholds
# Max entry age is required.
# Max messages (and max bytes) per entry are optional
#
MAX_ENTRY_AGE_SECONDS = 60.0
MAX_ENTRY_MESSAGES = 10

# Create a caching CMM
caching_cmm = CachingCryptoMaterialsManager(
    master_key_provider=key_provider,
    cache=cache,
    max_age=MAX_ENTRY_AGE_SECONDS,
    max_messages_encrypted=MAX_ENTRY_MESSAGES
)
```

Das ist alles. Lassen Sie sie dann den Cache für Sie AWS Encryption SDK verwalten oder fügen Sie Ihre eigene Cache-Verwaltungslogik hinzu.

Wenn Sie das Zwischenspeichern von Datenschlüsseln in einem Aufruf zum Verschlüsseln oder Entschlüsseln von Daten verwenden möchten, geben Sie CMM statt eines Masterschlüsselanbieters oder eines anderen das Caching an. CMM

Note

Wenn Sie Datenströme oder Daten unbekannter Größe verschlüsseln, stellen Sie sicher, dass Sie die Datengröße in der Anfrage angeben. Beim AWS Encryption SDK Verschlüsseln von Daten unbekannter Größe wird kein Datenschlüssel-Caching verwendet.

C

In der AWS-Verschlüsselungs-SDK for C erstellen Sie eine Sitzung mit dem Caching CMM und verarbeiten dann die Sitzung.

Standardmäßig werden Datenschlüssel AWS Encryption SDK nicht zwischengespeichert, wenn die Nachrichtengröße unbekannt und unbegrenzt ist. Um eine Zwischenspeicherung zuzulassen, wenn Sie die genaue Datengröße nicht kennen, verwenden Sie die `aws_cryptosdk_session_set_message_bound`-Methode, um eine maximale Größe für die Nachricht festzulegen. Legen Sie die Grenze größer als die geschätzte Nachrichtengröße fest. Wenn die tatsächliche Nachrichtengröße die Grenze überschreitet, schlägt die Verschlüsselungsoperation fehl.

```
/* Create a session with the caching CMM. Set the session mode to encrypt. */
struct aws_cryptosdk_session *session =
    aws_cryptosdk_session_new_from_cmm_2(allocator, AWS_CRYPTOSDK_ENCRYPT,
    caching_cmm);

/* Set a message bound of 1000 bytes */
aws_status = aws_cryptosdk_session_set_message_bound(session, 1000);

/* Encrypt the message using the session with the caching CMM */
aws_status = aws_cryptosdk_session_process(
    session, output_buffer, output_capacity, &output_produced,
    input_buffer, input_len, &input_consumed);

/* Release your references to the caching CMM and the session. */
```



```
aws_cryptosdk_cmm_release(caching_cmm);  
aws_cryptosdk_session_destroy(session);
```

Java

Im folgenden Beispiel wird Version 2 verwendet. x der AWS-Verschlüsselungs-SDK for Java. Ausführung 3. x von der AWS-Verschlüsselungs-SDK for Java missbilligt das Zwischenspeichern von Datenschlüsseln. CMM Mit Version 3. x, Sie können auch den [AWS KMS hierarchischen Schlüsselbund](#) verwenden, eine alternative Lösung zum Zwischenspeichern kryptografischer Materialien.

```
// When the call to encryptData specifies a caching CMM,  
// the encryption operation uses the data key cache  
final AwsCrypto encryptionSdk = AwsCrypto.standard();  
return encryptionSdk.encryptData(cachingCmm, plaintext_source).getResult();
```

JavaScript Browser

```
const { result } = await encrypt(cachingCmm, plaintext)
```

JavaScript Node.js

Wenn Sie die Zwischenspeicherung CMM in der Datei AWS-Verschlüsselungs-SDK for JavaScript für Node.js verwenden, benötigt die `encrypt` Methode die Länge des Klartextes. Wenn Sie sie nicht angeben, wird der Datenschlüssel nicht zwischengespeichert. Wenn Sie eine Länge angeben, aber die von Ihnen bereitgestellten Klartextdaten diese Länge überschreiten, schlägt die Verschlüsselungsoperation fehl. Wenn Sie die genaue Länge des Klartextes nicht kennen, z. B. beim Streamen von Daten, geben Sie den größten erwarteten Wert an.

```
const { result } = await encrypt(cachingCmm, plaintext, { plaintextLength:  
  plaintext.length })
```

Python

```
# Set up an encryption client  
client = aws_encryption_sdk.EncryptionSDKClient()  
  
# When the call to encrypt specifies a caching CMM,  
# the encryption operation uses the data key cache  
#  
encrypted_message, header = client.encrypt(
```

```
    source=plaintext_source,  
    materials_manager=caching_cmm  
)
```

Beispiel für das Datenschlüssel-Caching: Verschlüsseln einer Zeichenfolge

Dieses einfache Codebeispiel verwendet das Datenschlüssel-Caching beim Verschlüsseln einer Zeichenfolge. Sie kombiniert den Code aus der [step-by-step Prozedur](#) zu Testcode, den Sie ausführen können.

Das Beispiel erstellt einen [lokalen Cache](#) und einen [Hauptschlüsselanbieter](#) oder [Schlüsselbund](#) für einen AWS KMS key. [Anschließend werden der lokale Cache und der Hauptschlüsselanbieter oder der Schlüsselbund verwendet, um eine Zwischenspeicherung CMM mit entsprechenden Sicherheitsschwellenwerten zu erstellen.](#) [In Java und Python spezifiziert die Verschlüsselungsanforderung das CachingCMM, die zu verschlüsselnden Klartextdaten und einen Verschlüsselungskontext.](#) In C CMM wird das Caching in der Sitzung angegeben, und die Sitzung wird für die Verschlüsselungsanforderung bereitgestellt.

Um diese Beispiele auszuführen, müssen Sie den [Amazon-Ressourcennamen \(ARN\) eines angeben AWS KMS key](#). Stellen Sie sicher, dass Sie die [Berechtigung für die Verwendung des AWS KMS key](#) zum Generieren eines Datenschlüssels besitzen.

Ausführlichere Beispiele aus der Praxis für die Erstellung und Verwendung eines Datenschlüssel-Caches finden Sie unter [Beispielcode für das Zwischenspeichern von Datenschlüsseln](#).

C

```
/*  
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.  
 *  
 * Licensed under the Apache License, Version 2.0 (the "License"). You may not use  
 * this file except in compliance with the License. A copy of the License is  
 * located at  
 *  
 *     http://aws.amazon.com/apache2.0/  
 *  
 * or in the "license" file accompanying this file. This file is distributed on an  
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or  
 * implied. See the License for the specific language governing permissions and  
 * limitations under the License.
```

```
*/

#include <aws/cryptosdk/cache.h>
#include <aws/cryptosdk/cpp/kms_keyring.h>
#include <aws/cryptosdk/session.h>

void encrypt_with_caching(
    uint8_t *ciphertext,    // output will go here (assumes ciphertext_capacity
bytes already allocated)
    size_t *ciphertext_len, // length of output will go here
    size_t ciphertext_capacity,
    const char *kms_key_arn,
    int max_entry_age,
    int cache_capacity) {
    const uint64_t MAX_ENTRY_MSGS = 100;

    struct aws_allocator *allocator = aws_default_allocator();

    // Load error strings for debugging
    aws_cryptosdk_load_error_strings();

    // Create a keyring
    struct aws_cryptosdk_keyring *kms_keyring =
Aws::Cryptosdk::KmsKeyring::Builder().Build(kms_key_arn);

    // Create a cache
    struct aws_cryptosdk_materials_cache *cache =
aws_cryptosdk_materials_cache_local_new(allocator, cache_capacity);

    // Create a caching CMM
    struct aws_cryptosdk_cmm *caching_cmm =
aws_cryptosdk_caching_cmm_new_from_keyring(
    allocator, cache, kms_keyring, NULL, max_entry_age, AWS_TIMESTAMP_SECS);
    if (!caching_cmm) abort();

    if (aws_cryptosdk_caching_cmm_set_limit_messages(caching_cmm, MAX_ENTRY_MSGS))
abort();

    // Create a session
    struct aws_cryptosdk_session *session =
    aws_cryptosdk_session_new_from_cmm_2(allocator, AWS_CRYPTOSDK_ENCRYPT,
caching_cmm);
    if (!session) abort();
}
```

```
// Encryption context
struct aws_hash_table *enc_ctx =
aws_cryptosdk_session_get_enc_ctx_ptr_mut(session);
if (!enc_ctx) abort();
AWS_STATIC_STRING_FROM_LITERAL(enc_ctx_key, "purpose");
AWS_STATIC_STRING_FROM_LITERAL(enc_ctx_value, "test");
if (aws_hash_table_put(enc_ctx, enc_ctx_key, (void *)enc_ctx_value, NULL))
abort();

// Plaintext data to be encrypted
const char *my_data = "My plaintext data";
size_t my_data_len = strlen(my_data);
if (aws_cryptosdk_session_set_message_size(session, my_data_len)) abort();

// When the session uses a caching CMM, the encryption operation uses the data
key cache
// specified in the caching CMM.
size_t bytes_read;
if (aws_cryptosdk_session_process(
    session,
    ciphertext,
    ciphertext_capacity,
    ciphertext_len,
    (const uint8_t *)my_data,
    my_data_len,
    &bytes_read))
    abort();
if (!aws_cryptosdk_session_is_done(session) || bytes_read != my_data_len)
abort();

aws_cryptosdk_session_destroy(session);
aws_cryptosdk_cmm_release(caching_cmm);
aws_cryptosdk_materials_cache_release(cache);
aws_cryptosdk_keyring_release(kms_keyring);
}
```

Java

Im folgenden Beispiel wird Version 2 verwendet. x der AWS-Verschlüsselungs-SDK for Java. Ausführung 3. x von der AWS-Verschlüsselungs-SDK for Java missbilligt das Zwischenspeichern von Datenschlüsseln. CMM Mit Version 3. x, Sie können auch den [AWS KMS hierarchischen Schlüsselbund](#) verwenden, eine alternative Lösung zum Zwischenspeichern kryptografischer Materialien.

```
// Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

package com.amazonaws.crypto.examples;

import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CryptoMaterialsManager;
import com.amazonaws.encryptionsdk.MasterKeyProvider;
import com.amazonaws.encryptionsdk.caching.CachingCryptoMaterialsManager;
import com.amazonaws.encryptionsdk.caching.CryptoMaterialsCache;
import com.amazonaws.encryptionsdk.caching.LocalCryptoMaterialsCache;
import com.amazonaws.encryptionsdk.kmsdkv2.KmsMasterKey;
import com.amazonaws.encryptionsdk.kmsdkv2.KmsMasterKeyProvider;
import java.nio.charset.StandardCharsets;
import java.util.Collections;
import java.util.Map;
import java.util.concurrent.TimeUnit;

/**
 * <p>
 * Encrypts a string using an &KMS; key and data key caching
 *
 * <p>
 * Arguments:
 * <ol>
 * <li>KMS Key ARN: To find the Amazon Resource Name of your &KMS; key,
 *     see 'Find the key ID and ARN' at https://docs.aws.amazon.com/kms/latest/developerguide/find-cmk-id-arn.html
 * <li>Max entry age: Maximum time (in seconds) that a cached entry can be used
 * <li>Cache capacity: Maximum number of entries in the cache
 * </ol>
 */
public class SimpleDataKeyCachingExample {

    /**
     * Security thresholds
     * Max entry age is required.
     * Max messages (and max bytes) per data key are optional
     */
    private static final int MAX_ENTRY_MSGS = 100;

    public static byte[] encryptWithCaching(String kmsKeyArn, int maxEntryAge, int
cacheCapacity) {
```

```

// Plaintext data to be encrypted
byte[] myData = "My plaintext data".getBytes(StandardCharsets.UTF_8);

// Encryption context
// Most encrypted data should have an associated encryption context
// to protect integrity. This sample uses placeholder values.
// For more information see:
// blogs.aws.amazon.com/security/post/Tx2LZ6WBJJANTNW/How-to-Protect-the-
Integrity-of-Your-Encrypted-Data-by-Using-AWS-Key-Management
final Map<String, String> encryptionContext =
Collections.singletonMap("purpose", "test");

// Create a master key provider
MasterKeyProvider<KmsMasterKey> keyProvider =
KmsMasterKeyProvider.builder()
    .buildStrict(kmsKeyArn);

// Create a cache
CryptoMaterialsCache cache = new LocalCryptoMaterialsCache(cacheCapacity);

// Create a caching CMM
CryptoMaterialsManager cachingCmm =
CachingCryptoMaterialsManager.newBuilder().withMasterKeyProvider(keyProvider)
    .withCache(cache)
    .withMaxAge(maxEntryAge, TimeUnit.SECONDS)
    .withMessageUseLimit(MAX_ENTRY_MSGS)
    .build();

// When the call to encryptData specifies a caching CMM,
// the encryption operation uses the data key cache
final AwsCrypto encryptionSdk = AwsCrypto.standard();
return encryptionSdk.encryptData(cachingCmm, myData,
encryptionContext).getResult();
    }
}

```

JavaScript Browser

```

// Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

/* This is a simple example of using a caching CMM with a KMS keyring

```

```
* to encrypt and decrypt using the AWS Encryption SDK for Javascript in a browser.
*/

import {
  KmsKeyringBrowser,
  KMS,
  getClient,
  buildClient,
  CommitmentPolicy,
  WebCryptoCachingMaterialsManager,
  getLocalCryptographicMaterialsCache,
} from '@aws-crypto/client-browser'
import { toBase64 } from '@aws-sdk/util-base64-browser'

/* This builds the client with the REQUIRE_ENCRYPT_REQUIRE_DECRYPT commitment
policy,
* which enforces that this client only encrypts using committing algorithm suites
* and enforces that this client
* will only decrypt encrypted messages
* that were created with a committing algorithm suite.
* This is the default commitment policy
* if you build the client with `buildClient()`.
*/
const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

/* This is injected by webpack.
* The webpack.DefinePlugin or @aws-sdk/karma-credential-loader will replace the
values when bundling.
* The credential values are pulled from @aws-sdk/credential-provider-node
* Use any method you like to get credentials into the browser.
* See kms.webpack.config
*/
declare const credentials: {
  accessKeyId: string
  secretAccessKey: string
  sessionToken: string
}

/* This is done to facilitate testing. */
export async function testCachingCMMEExample() {
  /* This example uses an &KMS; keyring. The generator key in a &KMS; keyring
generates and encrypts the data key.
```

```
* The caller needs kms:GenerateDataKey permission on the &KMS; key in
generatorKeyId.
*/
const generatorKeyId =
  'arn:aws:kms:us-west-2:658956600833:alias/EncryptDecrypt'

/* Adding additional KMS keys that can decrypt.
* The caller must have kms:Encrypt permission for every &KMS; key in keyIds.
* You might list several keys in different AWS Regions.
* This allows you to decrypt the data in any of the represented Regions.
* In this example, the generator key
* and the additional key are actually the same &KMS; key.
* In `generatorId`, this &KMS; key is identified by its alias ARN.
* In `keyIds`, this &KMS; key is identified by its key ARN.
* In practice, you would specify different &KMS; keys,
* or omit the `keyIds` parameter.
* This is *only* to demonstrate how the &KMS; key ARNs are configured.
*/
const keyIds = [
  'arn:aws:kms:us-west-2:658956600833:key/b3537ef1-d8dc-4780-9f5a-55776cbb2f7f',
]

/* Need a client provider that will inject correct credentials.
* The credentials here are injected by webpack from your environment bundle is
created
* The credential values are pulled using @aws-sdk/credential-provider-node.
* See kms.webpack.config
* You should inject your credential into the browser in a secure manner
* that works with your application.
*/
const { accessKeyId, secretAccessKey, sessionToken } = credentials

/* getClient takes a KMS client constructor
* and optional configuration values.
* The credentials can be injected here,
* because browsers do not have a standard credential discovery process the way
Node.js does.
*/
const clientProvider = getClient(KMS, {
  credentials: {
    accessKeyId,
    secretAccessKey,
    sessionToken,
  },
},
```



```
    })

    /* You must configure the KMS keyring with your &KMS; keys */
    const keyring = new KmsKeyringBrowser({
        clientProvider,
        generatorKeyId,
        keyIds,
    })

    /* Create a cache to hold the data keys (and related cryptographic material).
     * This example uses the local cache provided by the Encryption SDK.
     * The `capacity` value represents the maximum number of entries
     * that the cache can hold.
     * To make room for an additional entry,
     * the cache evicts the oldest cached entry.
     * Both encrypt and decrypt requests count independently towards this threshold.
     * Entries that exceed any cache threshold are actively removed from the cache.
     * By default, the SDK checks one item in the cache every 60 seconds (60,000
    milliseconds).
     * To change this frequency, pass in a `proactiveFrequency` value
     * as the second parameter. This value is in milliseconds.
     */
    const capacity = 100
    const cache = getLocalCryptographicMaterialsCache(capacity)

    /* The partition name lets multiple caching CMMs share the same local
    cryptographic cache.
     * By default, the entries for each CMM are cached separately. However, if you
    want these CMMs to share the cache,
     * use the same partition name for both caching CMMs.
     * If you don't supply a partition name, the Encryption SDK generates a random
    name for each caching CMM.
     * As a result, sharing elements in the cache MUST be an intentional operation.
     */
    const partition = 'local partition name'

    /* maxAge is the time in milliseconds that an entry will be cached.
     * Elements are actively removed from the cache.
     */
    const maxAge = 1000 * 60

    /* The maximum number of bytes that will be encrypted under a single data key.
     * This value is optional,
     * but you should configure the lowest practical value.
```

```
*/
const maxBytesEncrypted = 100

/* The maximum number of messages that will be encrypted under a single data key.
 * This value is optional,
 * but you should configure the lowest practical value.
 */
const maxMessagesEncrypted = 10

const cachingCMM = new WebCryptoCachingMaterialsManager({
  backingMaterials: keyring,
  cache,
  partition,
  maxAge,
  maxBytesEncrypted,
  maxMessagesEncrypted,
})

/* Encryption context is a very powerful tool for controlling
 * and managing access.
 * When you pass an encryption context to the encrypt function,
 * the encryption context is cryptographically bound to the ciphertext.
 * If you don't pass in the same encryption context when decrypting,
 * the decrypt function fails.
 * The encryption context is ***not*** secret!
 * Encrypted data is opaque.
 * You can use an encryption context to assert things about the encrypted data.
 * The encryption context helps you to determine
 * whether the ciphertext you retrieved is the ciphertext you expect to decrypt.
 * For example, if you are only expecting data from 'us-west-2',
 * the appearance of a different AWS Region in the encryption context can indicate
malicious interference.
 * See: https://docs.aws.amazon.com/encryption-sdk/latest/developer-guide/
concepts.html#encryption-context
 *
 * Also, cached data keys are reused ***only*** when the encryption contexts
passed into the functions are an exact case-sensitive match.
 * See: https://docs.aws.amazon.com/encryption-sdk/latest/developer-guide/data-
caching-details.html#caching-encryption-context
 */
const encryptionContext = {
  stage: 'demo',
  purpose: 'simple demonstration app',
  origin: 'us-west-2',
```

```
}

/* Find data to encrypt. */
const plainText = new Uint8Array([1, 2, 3, 4, 5])

/* Encrypt the data.
 * The caching CMM only reuses data keys
 * when it know the length (or an estimate) of the plaintext.
 * However, in the browser,
 * you must provide all of the plaintext to the encrypt function.
 * Therefore, the encrypt function in the browser knows the length of the
plaintext
 * and does not accept a plaintextLength option.
 */
const { result } = await encrypt(cachingCMM, plainText, { encryptionContext })

/* Log the plain text
 * only for testing and to show that it works.
 */
console.log('plainText:', plainText)
document.write('</br>plainText:' + plainText + '</br>')

/* Log the base64-encoded result
 * so that you can try decrypting it with another AWS Encryption SDK
implementation.
 */
const resultBase64 = toBase64(result)
console.log(resultBase64)
document.write(resultBase64)

/* Decrypt the data.
 * NOTE: This decrypt request will not use the data key
 * that was cached during the encrypt operation.
 * Data keys for encrypt and decrypt operations are cached separately.
 */
const { plaintext, messageHeader } = await decrypt(cachingCMM, result)

/* Grab the encryption context so you can verify it. */
const { encryptionContext: decryptedContext } = messageHeader

/* Verify the encryption context.
 * If you use an algorithm suite with signing,
 * the Encryption SDK adds a name-value pair to the encryption context that
contains the public key.
```

```

    * Because the encryption context might contain additional key-value pairs,
    * do not include a test that requires that all key-value pairs match.
    * Instead, verify that the key-value pairs that you supplied to the `encrypt`
function are included in the encryption context that the `decrypt` function
returns.
    */
Object.entries(encryptionContext).forEach(([key, value]) => {
  if (decryptedContext[key] !== value)
    throw new Error('Encryption Context does not match expected values')
})

/* Log the clear message
 * only for testing and to show that it works.
 */
document.write('</br>Decrypted:' + plaintext)
console.log(plaintext)

/* Return the values to make testing easy. */
return { plainText, plaintext }
}

```

JavaScript Node.js

```

// Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  KmsKeyringNode,
  buildClient,
  CommitmentPolicy,
  NodeCachingMaterialsManager,
  getLocalCryptographicMaterialsCache,
} from '@aws-crypto/client-node'

/* This builds the client with the REQUIRE_ENCRYPT_REQUIRE_DECRYPT commitment
policy,
 * which enforces that this client only encrypts using committing algorithm suites
 * and enforces that this client
 * will only decrypt encrypted messages
 * that were created with a committing algorithm suite.
 * This is the default commitment policy
 * if you build the client with `buildClient()`.
 */

```

```
const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

export async function cachingCMMNodeSimpleTest() {
  /* An &KMS; key is required to generate the data key.
   * You need kms:GenerateDataKey permission on the &KMS; key in generatorKeyId.
   */
  const generatorKeyId =
    'arn:aws:kms:us-west-2:658956600833:alias/EncryptDecrypt'

  /* Adding alternate &KMS; keys that can decrypt.
   * Access to kms:Encrypt is required for every &KMS; key in keyIds.
   * You might list several keys in different AWS Regions.
   * This allows you to decrypt the data in any of the represented Regions.
   * In this example, the generator key
   * and the additional key are actually the same &KMS; key.
   * In `generatorId`, this &KMS; key is identified by its alias ARN.
   * In `keyIds`, this &KMS; key is identified by its key ARN.
   * In practice, you would specify different &KMS; keys,
   * or omit the `keyIds` parameter.
   * This is *only* to demonstrate how the &KMS; key ARNs are configured.
   */
  const keyIds = [
    'arn:aws:kms:us-west-2:658956600833:key/b3537ef1-d8dc-4780-9f5a-55776cbb2f7f',
  ]

  /* The &KMS; keyring must be configured with the desired &KMS; keys
   * This example passes the keyring to the caching CMM
   * instead of using it directly.
   */
  const keyring = new KmsKeyringNode({ generatorKeyId, keyIds })

  /* Create a cache to hold the data keys (and related cryptographic material).
   * This example uses the local cache provided by the Encryption SDK.
   * The `capacity` value represents the maximum number of entries
   * that the cache can hold.
   * To make room for an additional entry,
   * the cache evicts the oldest cached entry.
   * Both encrypt and decrypt requests count independently towards this threshold.
   * Entries that exceed any cache threshold are actively removed from the cache.
   * By default, the SDK checks one item in the cache every 60 seconds (60,000
  milliseconds).
   * To change this frequency, pass in a `proactiveFrequency` value
```

```
* as the second parameter. This value is in milliseconds.
*/
const capacity = 100
const cache = getLocalCryptographicMaterialsCache(capacity)

/* The partition name lets multiple caching CMMs share the same local
cryptographic cache.
* By default, the entries for each CMM are cached separately. However, if you
want these CMMs to share the cache,
* use the same partition name for both caching CMMs.
* If you don't supply a partition name, the Encryption SDK generates a random
name for each caching CMM.
* As a result, sharing elements in the cache MUST be an intentional operation.
*/
const partition = 'local partition name'

/* maxAge is the time in milliseconds that an entry will be cached.
* Elements are actively removed from the cache.
*/
const maxAge = 1000 * 60

/* The maximum amount of bytes that will be encrypted under a single data key.
* This value is optional,
* but you should configure the lowest value possible.
*/
const maxBytesEncrypted = 100

/* The maximum number of messages that will be encrypted under a single data key.
* This value is optional,
* but you should configure the lowest value possible.
*/
const maxMessagesEncrypted = 10

const cachingCMM = new NodeCachingMaterialsManager({
  backingMaterials: keyring,
  cache,
  partition,
  maxAge,
  maxBytesEncrypted,
  maxMessagesEncrypted,
})

/* Encryption context is a very powerful tool for controlling
* and managing access.
```

```

* When you pass an encryption context to the encrypt function,
* the encryption context is cryptographically bound to the ciphertext.
* If you don't pass in the same encryption context when decrypting,
* the decrypt function fails.
* The encryption context is ***not*** secret!
* Encrypted data is opaque.
* You can use an encryption context to assert things about the encrypted data.
* The encryption context helps you to determine
* whether the ciphertext you retrieved is the ciphertext you expect to decrypt.
* For example, if you are only expecting data from 'us-west-2',
* the appearance of a different AWS Region in the encryption context can indicate
malicious interference.
  * See: https://docs.aws.amazon.com/encryption-sdk/latest/developer-guide/
concepts.html#encryption-context
  *
  * Also, cached data keys are reused ***only*** when the encryption contexts
passed into the functions are an exact case-sensitive match.
  * See: https://docs.aws.amazon.com/encryption-sdk/latest/developer-guide/data-
caching-details.html#caching-encryption-context
  */
const encryptionContext = {
  stage: 'demo',
  purpose: 'simple demonstration app',
  origin: 'us-west-2',
}

/* Find data to encrypt. A simple string. */
const cleartext = 'asdf'

/* Encrypt the data.
* The caching CMM only reuses data keys
* when it know the length (or an estimate) of the plaintext.
* If you do not know the length,
* because the data is a stream
* provide an estimate of the largest expected value.
*
* If your estimate is smaller than the actual plaintext length
* the AWS Encryption SDK will throw an exception.
*
* If the plaintext is not a stream,
* the AWS Encryption SDK uses the actual plaintext length
* instead of any length you provide.
*/
const { result } = await encrypt(cachingCMM, cleartext, {

```

```

    encryptionContext,
    plaintextLength: 4,
  })

  /* Decrypt the data.
   * NOTE: This decrypt request will not use the data key
   * that was cached during the encrypt operation.
   * Data keys for encrypt and decrypt operations are cached separately.
   */
  const { plaintext, messageHeader } = await decrypt(cachingCMM, result)

  /* Grab the encryption context so you can verify it. */
  const { encryptionContext: decryptedContext } = messageHeader

  /* Verify the encryption context.
   * If you use an algorithm suite with signing,
   * the Encryption SDK adds a name-value pair to the encryption context that
   contains the public key.
   * Because the encryption context might contain additional key-value pairs,
   * do not include a test that requires that all key-value pairs match.
   * Instead, verify that the key-value pairs that you supplied to the `encrypt`
   function are included in the encryption context that the `decrypt` function
   returns.
   */
  Object.entries(encryptionContext).forEach(([key, value]) => {
    if (decryptedContext[key] !== value)
      throw new Error('Encryption Context does not match expected values')
  })

  /* Return the values so the code can be tested. */
  return { plaintext, result, cleartext, messageHeader }
}

```

Python

```

# Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You
# may not use this file except in compliance with the License. A copy of
# the License is located at
#
# http://aws.amazon.com/apache2.0/
#

```



```
# or in the "license" file accompanying this file. This file is
# distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF
# ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.
"""Example of encryption with data key caching."""
import aws_encryption_sdk
from aws_encryption_sdk import CommitmentPolicy

def encrypt_with_caching(kms_key_arn, max_age_in_cache, cache_capacity):
    """Encrypts a string using an &KMS; key and data key caching.

    :param str kms_key_arn: Amazon Resource Name (ARN) of the &KMS; key
    :param float max_age_in_cache: Maximum time in seconds that a cached entry can
    be used
    :param int cache_capacity: Maximum number of entries to retain in cache at once
    """
    # Data to be encrypted
    my_data = "My plaintext data"

    # Security thresholds
    # Max messages (or max bytes per) data key are optional
    MAX_ENTRY_MESSAGES = 100

    # Create an encryption context
    encryption_context = {"purpose": "test"}

    # Set up an encryption client with an explicit commitment policy. Note that if
    you do not explicitly choose a
    # commitment policy, REQUIRE_ENCRYPT_REQUIRE_DECRYPT is used by default.
    client =
aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_R

    # Create a master key provider for the &KMS; key
    key_provider =
aws_encryption_sdk.StrictAwsKmsMasterKeyProvider(key_ids=[kms_key_arn])

    # Create a local cache
    cache = aws_encryption_sdk.LocalCryptoMaterialsCache(cache_capacity)

    # Create a caching CMM
    caching_cmm = aws_encryption_sdk.CachingCryptoMaterialsManager(
        master_key_provider=key_provider,
        cache=cache,
```

```
        max_age=max_age_in_cache,  
        max_messages_encrypted=MAX_ENTRY_MESSAGES,  
    )  
  
    # When the call to encrypt data specifies a caching CMM,  
    # the encryption operation uses the data key cache specified  
    # in the caching CMM  
    encrypted_message, _header = client.encrypt(  
        source=my_data, materials_manager=caching_cmm,  
    encryption_context=encryption_context  
    )  
  
    return encrypted_message
```

Festlegen von Cache-Sicherheitsschwellenwerten

Wenn Sie das Datenschlüssel-Caching implementieren möchten, konfigurieren Sie die Sicherheitsschwellenwerte, die der [CMM zwischenspeichern](#) erzwingt.

Die Sicherheitsschwellenwerte helfen Ihnen, die Verwendungsdauer jedes im Cache gespeicherten Datenschlüssels und die Menge der unter jedem Datenschlüssel geschützten Daten zu begrenzen. Das Caching-CMM gibt im Cache gespeicherte Datenschlüssel nur dann zurück, wenn der Cache-Eintrag allen Sicherheitsschwellenwerten entspricht. Wenn der Cache-Eintrag einen vorgegebenen Schwellenwert überschreitet, wird der Eintrag für die aktuelle Operation nicht verwendet und so bald wie möglich aus dem Cache entfernt. Die erste Verwendung der einzelnen Datenschlüssel (vor dem Caching) ist davon ausgenommen.

Verwenden Sie als Regel die minimale Menge an Caching, die erforderlich ist, um Ihre Kosten- und Leistungsziele zu erreichen.

Das AWS Encryption SDK speichert nur Datenschlüssel im Cache, die mit einer [Schlüsselableitungsfunktion](#) verschlüsselt wurden. Außerdem richtet es Obergrenzen für einige Schwellenwerte ein. Diese Einschränkungen stellen sicher, dass Datenschlüssel nicht über ihre kryptografischen Obergrenzen hinaus wiederverwendet werden. Da Ihre Klartext-Datenschlüssel jedoch im Cache gespeichert werden (standardmäßig im Arbeitsspeicher), sollten Sie versuchen, die Zeit zu minimieren, wie lange die Schlüssel gespeichert werden. Versuchen Sie auch, die Datenmenge zu begrenzen, die offengelegt werden könnte, wenn ein Schlüssel verletzt wird.

Beispiele zum Festlegen von Cache-Sicherheitsschwellenwerten finden Sie unter [AWS Encryption SDK: Wie Sie feststellen, ob das Datenschlüssel-Caching für Ihre Anwendung geeignet ist](#) im [AWS Blog](#) zur Sicherheit.

Note

Der Caching-CMM erzwingt alle folgenden Schwellenwerte. Wenn Sie keinen optionalen Wert angeben, verwendet der Caching-CMM den Standardwert.

Um das Datenschlüssel-Caching vorübergehend zu deaktivieren, bieten Java- und Python-Implementierungen des AWS Encryption SDK Stellen Sie eine bereitCache für kryptografische Materialien (Null-Cache). Der Null-Cache gibt einen Fehler für alle GET-Anfragen zurück und reagiert nicht auf PUT-Anfragen. Wir empfehlen Ihnen die Verwendung des Null-Cache, anstatt die [Cache-Kapazität](#) oder den Sicherheitsschwellenwert auf 0 zu setzen. Weitere Informationen finden Sie im Null-Cache in [Java](#) und [Python](#).

Höchstalter (erforderlich)

Legt fest, wie lange ein im Cache gespeicherter Eintrag verwendet werden kann, beginnend ab dem Zeitpunkt, zu dem er hinzugefügt wurde. Dieser Wert ist erforderlich. Geben Sie einen Wert größer als 0 ein. Das AWS Encryption SDK beschränkt das Höchstalter nicht.

Alle Sprachimplementierungen des AWS Encryption SDK definieren Sie das maximale Alter in Sekunden, mit Ausnahme des AWS-Verschlüsselungs-SDK for JavaScript, das Millisekunden verwendet.

Verwenden Sie das kürzeste Intervall, bei dem Ihre Anwendung noch vom Cache profitieren kann. Sie können die maximale Altersgrenze wie eine Schlüsselrotationsrichtlinie verwenden. Verwenden Sie sie, um die Wiederverwendung von Datenschlüsseln einzuschränken, die Offenlegung von kryptographischem Material zu minimieren und Datenschlüssel zu entfernen, deren Richtlinien sich möglicherweise geändert haben, während sie im Cache gespeichert waren.

Maximale Anzahl verschlüsselter Nachrichten (optional)

Gibt die maximale Anzahl der Nachrichten an, die ein im Cache gespeicherter Datenschlüssel verschlüsseln kann. Dieser Wert ist optional. Geben Sie einen Wert zwischen 1 und 2^{32} Nachrichten ein. Der Standardwert beträgt 2^{32} Nachrichten.

Stellen Sie die Anzahl der durch jeden im Cache gespeicherten Schlüssel geschützten Nachrichten so ein, dass sie groß genug ist, um Nutzen aus der Wiederverwendung zu erhalten,

aber klein genug, um die Anzahl der Nachrichten zu begrenzen, die bei einer Gefährdung eines Schlüssels offengelegt werden könnten.

Maximale Anzahl verschlüsselter Bytes (optional)

Gibt die maximale Anzahl der Bytes an, die ein im Cache gespeicherter Datenschlüssel verschlüsseln kann. Dieser Wert ist optional. Geben Sie einen Wert zwischen 0 und $2^{63} - 1$ ein. Der Standardwert lautet $2^{63} - 1$. Bei einem Wert von 0 können Sie Datenschlüssel-Caching nur verwenden, wenn Sie leere Nachrichtenzeichenfolgen verschlüsseln.

Die Bytes in der aktuellen Anfrage sind bei der Auswertung dieses Schwellenwerts enthalten. Wenn die verarbeiteten Bytes plus die aktuellen Bytes den Schwellenwert überschreiten, wird der im Cache gespeicherte Datenschlüssel aus dem Cache entfernt, auch wenn er bei einer kleineren Anfrage verwendet werden hätte können.

Weitere Informationen zum Datenschlüssel-Caching

Für die meisten Anwendungen ist die Standard-Implementierung des Datenschlüssel-Cachings ausreichend, ohne dass benutzerdefinierter Code geschrieben werden muss. Dieser Abschnitt beschreibt die Standard-Implementierung und einige Details zu Optionen.

Themen

- [Wie das Datenschlüssel-Caching funktioniert](#)
- [Erstellen eines Cache für kryptografische Materialien](#)
- [Erstellen eines Managers von kryptographischen Materialien, der Caching verwendet](#)
- [Was befindet sich in einem Datenschlüssel-Cache-Eintrag?](#)
- [Verschlüsselungskontext: Wie Cache-Einträge ausgewählt werden](#)
- [Benutzt meine Anwendung zwischengespeicherte Datenschlüssel?](#)

Wie das Datenschlüssel-Caching funktioniert

Wenn Sie Datenschlüssel-Caching in einer Anfrage verwenden, um Daten zu verschlüsseln oder zu entschlüsseln, sucht das AWS Encryption SDK zuerst im Cache nach einem Datenschlüssel, der der Anfrage entspricht. Wenn es eine gültige Übereinstimmung findet, verwendet es den zwischengespeicherten Datenschlüssel, um die Daten zu verschlüsseln. Andernfalls erzeugt es einen neuen Datenschlüssel, genau wie ohne den Cache.

Das Datenschlüssel-Caching wird nicht für Daten unbekannter Größe verwendet, wie z. B. gestreamte Daten. Auf diese Weise kann der Caching-CMM den ordnungsgemäß erzwingen [maximaler Byte-Schwellenwert](#) aus. Um dieses Verhalten zu vermeiden, fügen Sie der Verschlüsselungsanfrage die Nachrichtengröße hinzu.

Zusätzlich zu einem Cache verwendet das Datenschlüssel-Caching einen [Manager von kryptographischen Materialien](#) (CMM zwischenspeichern). Das Caching CMM ist ein spezialisiertes [Manager von kryptographischen Materialien \(CMM\)](#) das interagiert mit einem [Zwischenspeicher](#) und ein Basiswert [CMM](#) aus. (Wenn Sie ein [Masterschlüsselanbieter](#) oder Schlüsselbund, der AWS Encryption SDK erstellt ein Standard-CMM für Sie.) Der Caching-CMM speichert die Datenschlüssel, die sein zugrunde liegender CMM zurückgibt, im Cache. Der Caching-CMM erzwingt auch die Sicherheitsschwellenwerte, die Sie festgelegt haben.

Um zu verhindern, dass der falsche Datenschlüssel aus dem Cache ausgewählt wird, erfordern alle kompatiblen Caching-CMMs, dass die folgenden Eigenschaften der im Cache gespeicherten kryptographischen Materialien mit der Materialanforderung übereinstimmen.

- [Algorithmen-Paket](#)
- [Verschlüsselungskontext](#) (auch, wenn dieser leer ist)
- Partitionsname (eine Zeichenfolge, die das Caching-CMM identifiziert)
- (Nur Entschlüsselung) Verschlüsselte Datenschlüssel

Note

Das AWS Encryption SDK speichert Datenschlüssel nur im Cache, wenn das [Algorithmen-Paket](#) eine [Schlüsselableitungsfunktion](#) verwendet.

Die folgenden Workflows zeigen, wie eine Anforderung zum Verschlüsseln von Daten mit und ohne Datenschlüssel-Caching verarbeitet wird. Sie zeigen, wie die Caching-Komponenten, die Sie erstellen, einschließlich des Caches und des Caching-CMM dabei verwendet werden.

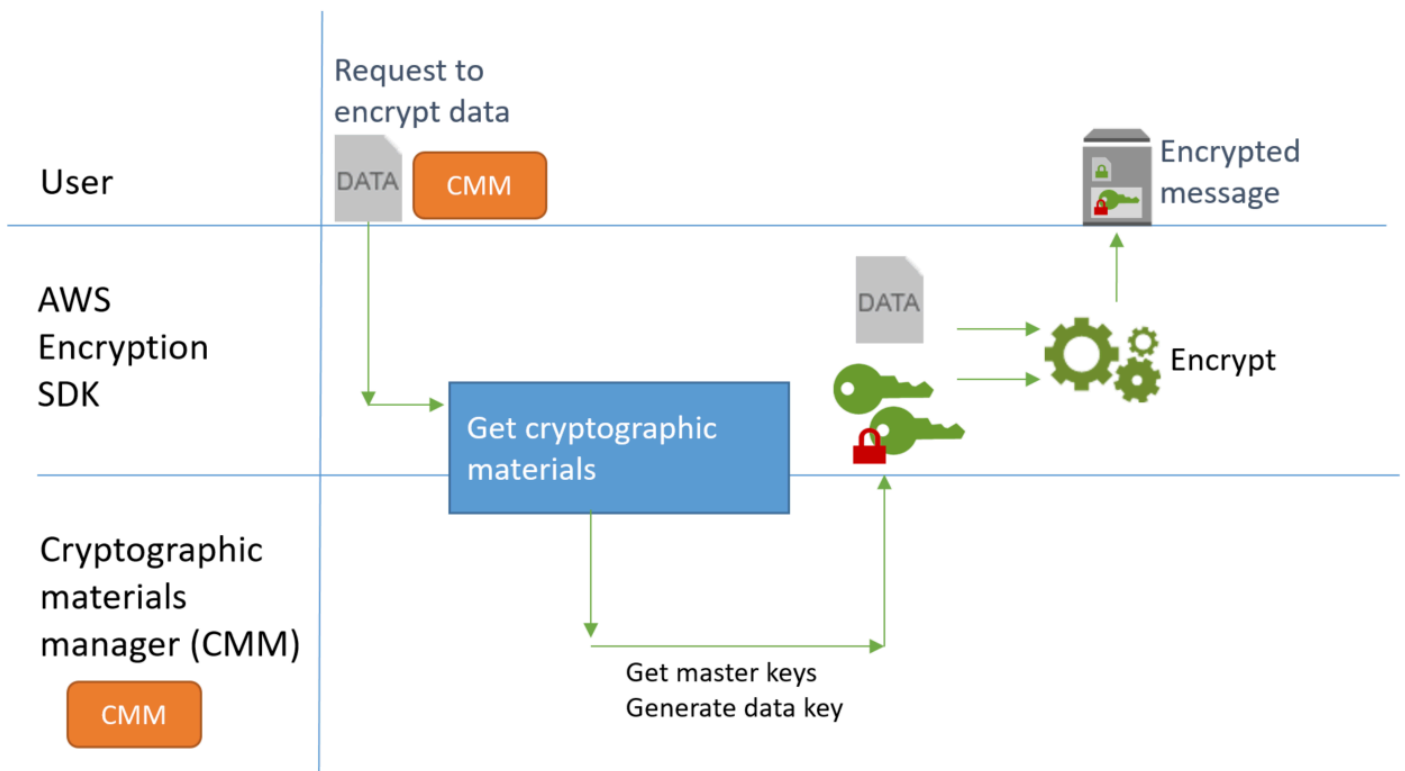
Verschlüsseln von Daten ohne Caching

So rufen Sie Verschlüsselungsmaterialien ohne Caching ab:

1. Eine Anwendung fordert das AWS Encryption SDK auf, Daten zu verschlüsseln.

Die Anforderung gibt einen Masterschlüssel-Anbieter oder Schlüsselbund an. Die AWS Encryption SDK erstellt ein standardmäßiges CMM, das mit Ihrem Masterschlüssel-Anbieter oder Schlüsselbund interagiert.

- Die AWS Encryption SDK fordert vom CMM Verschlüsselungsmaterialien an (Rufen Sie kryptografische Materialien ab).
- Das CMM fragt [Schlüsselring](#) (C und JavaScript) oder [Masterschlüsselanbieter](#) (Java und Python) für kryptografische Materialien. Dabei könnte es sich um einen Aufruf eines kryptographischen Dienstes handeln, z. B. AWS Key Management Service (AWS KMS). Der CMM gibt die Verschlüsselungsmaterialien an das zurück AWS Encryption SDK aus.
- Das AWS Encryption SDK verschlüsselt die Daten mithilfe des Klartext-Datenschlüssels. Es speichert die verschlüsselten Daten und verschlüsselten Datenschlüssel in einer [verschlüsselten Nachricht](#), die an den Benutzer zurückgegeben wird.



Verschlüsseln von Daten mit Caching

So rufen Sie Verschlüsselungsmaterialien mit Datenschlüssel-Caching ab:

- Eine Anwendung fordert das AWS Encryption SDK auf, Daten zu verschlüsseln.

Die Anfrage gibt einen [Manager von kryptographischen Materialien \(Caching Caching\)](#) Der ist einem zugrundeliegenden Manager von kryptographischen Materialien (CMM) zugeordnet. Wenn Sie einen Masterschlüssel-Anbieter oder Schlüsselbund angeben, wird der AWS Encryption SDK ein Standard-CMM für Sie erstellt.

2. Das SDK fordert vom angegebenen Caching-CMM nach Verschlüsselungsmaterialien an.
3. Der Caching-CMM fordert Verschlüsselungsmaterialien aus dem Cache an.
 - a. Wenn der Cache eine Übereinstimmung findet, aktualisiert er das Alter und verwendet Werte des übereinstimmenden Cache-Eintrags, und gibt die im Cache gespeicherten Verschlüsselungsmaterialien an den Caching-CMM zurück.

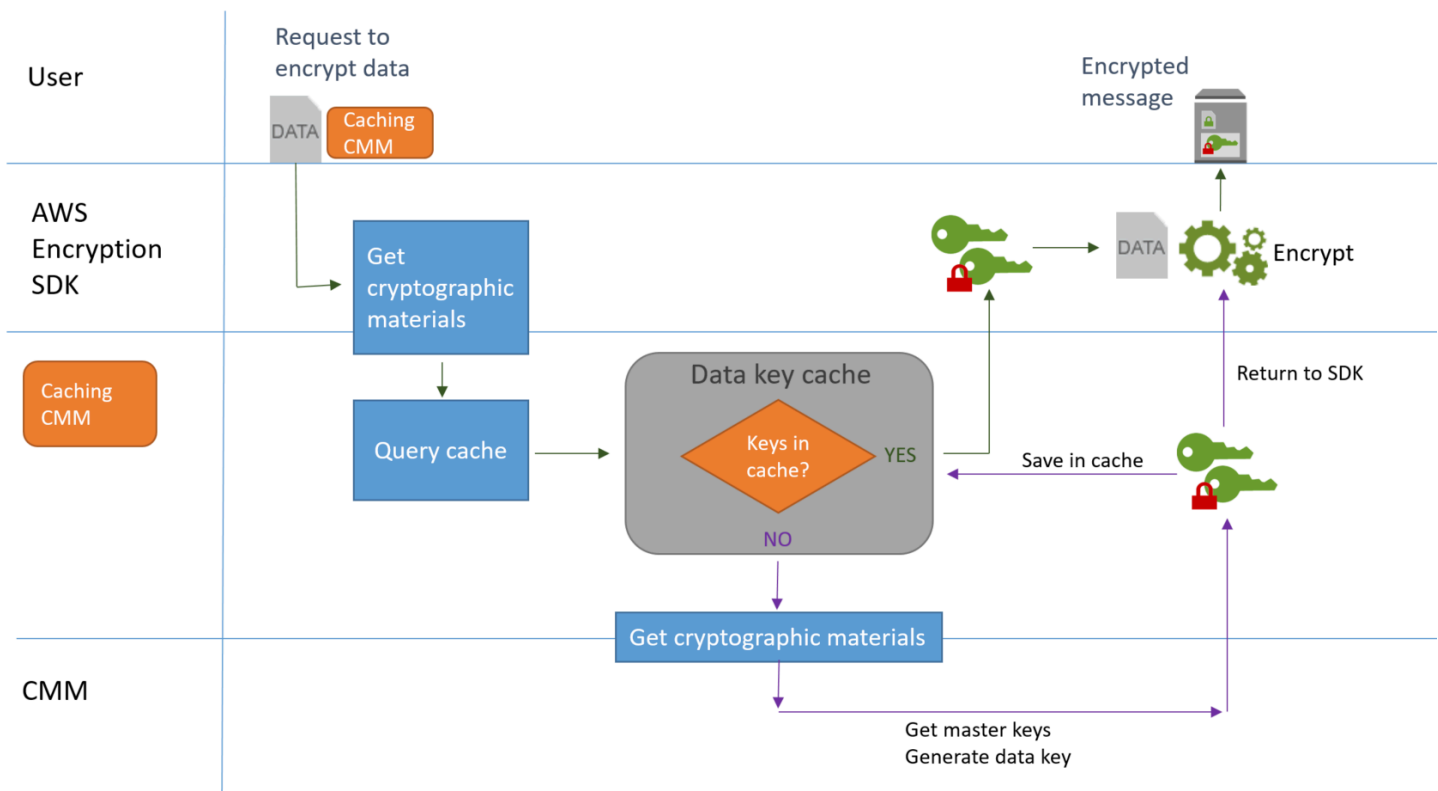
Wenn der Cache-Eintrag seinem entspricht [Sicherheitsschwellenwerte](#) Der Caching-CMM gibt es an das SDK zurück. Andernfalls weist es den Cache an, den Eintrag zu entfernen, und geht so vor, als gäbe es keine Übereinstimmung.

- b. Wenn der Cache keine gültige Übereinstimmung findet, fordert das Caching-CMM seinen zugrundeliegenden CMM auf, einen neuen Datenschlüssel zu generieren.

Der zugrundeliegende CMM ruft die kryptografischen Materialien aus seinem Schlüsselbund (C und JavaScript) oder Masterschlüssel-Anbieter (Java und Python). Dabei könnte es sich um einen Aufruf eines Dienstes handeln, z. B. AWS Key Management Service. Der zugrundeliegende CMM gibt eine Klartextkopie und eine verschlüsselte Kopie des Datenschlüssels an das Caching-CMM zurück.

Der Caching-CMM speichert die neuen Verschlüsselungsmaterialien im Cache.

4. Der Caching-CMM gibt die Verschlüsselungsmaterialien an das zurück AWS Encryption SDK aus.
5. Das AWS Encryption SDK verschlüsselt die Daten mithilfe des Klartext-Datenschlüssels. Es speichert die verschlüsselten Daten und verschlüsselten Datenschlüssel in einer [verschlüsselten Nachricht](#), die an den Benutzer zurückgegeben wird.



Erstellen eines Cache für kryptografische Materialien

Das AWS Encryption SDK definiert die Anforderungen für einen Cache für kryptografische Materialien, der für das Datenschlüssel-Caching verwendet wird. Es stellt auch einen lokalen Cache bereit, der einen konfigurierbaren In-Memory-In-Memory-ist [Der am wenigsten verwendete \(LRU\) Cache](#) aus. Um eine Instanz des lokalen Caches zu erstellen, verwenden Sie die `LocalCryptoMaterialsCache` Konstruktor in Java und `PythonGetLocalCryptographicMaterialsCache` Funktion in JavaScript oder das `saws_cryptosdk_materials_cache_local_new` Konstruktor in C.

Der lokale Cache enthält eine Logik für die grundlegende Cache-Verwaltung, einschließlich des Hinzufügens, Bereinigens und Vergleichen von Cache-Einträgen und der Wartung des Caches. Sie müssen keine benutzerdefinierte Cache-Verwaltungslogik schreiben. Sie können den lokalen Cache unverändert einsetzen oder ihn anpassen oder durch einen beliebigen kompatiblen Cache ersetzen.

Wenn Sie einen lokalen Cache erstellen, legen Sie dessen fest Kapazität, d. h. die maximale Anzahl von Einträgen, die der Cache aufnehmen kann. Diese Einstellung hilft Ihnen, einen effizienten Cache mit begrenzter Wiederverwendung von Datenschlüsseln zu entwerfen.

Die AWS-Verschlüsselungs-SDK for Java und die AWS-Verschlüsselungs-SDK for Python stellen Sie auch eine Cache für kryptografische Materialien (`NullCryptoMaterialsCache`) enthalten. Die `NullCryptoMaterialsCache` gibt einen Fehler für alle GET-Operationen zurück und reagiert nicht auf PUT-Operationen. Sie können die `NullCryptoMaterialsCache` beim Testen verwenden oder damit das Caching in einer Anwendung, die Caching-Code enthält, vorübergehend deaktivieren.

In der AWS Encryption SDK ist jeder kryptografische Material-Cache einem zugeordnet [Manager von kryptografischen Materialien](#) (CMM zwischenspeichern). Der Caching-CMM ruft Datenschlüssel aus dem Cache ab, fügt Datenschlüssel in den Cache ein und erzwingt [Sicherheitsschwellenwert](#) das du eingestellt hast. Beim Anlegen eines Caching-CMM geben Sie den verwendeten Cache und den zugrunde liegenden CMM- oder Masterschlüssel-Anbieter an, der die von ihm gecachten Datenschlüssel erzeugt.

Erstellen eines Managers von kryptografischen Materialien, der Caching verwendet

Um das Datenschlüssel-Caching zu aktivieren, erstellen Sie einen [Zwischenspeicher](#) und einen [Manager von kryptografischen Materialien](#) (CMM zwischenspeichern). Dann geben Sie in Ihren Anfragen zum Verschlüsseln oder Entschlüsseln von Daten einen Caching-CMM an, statt eines standardmäßigen [Manager von kryptografischen Materialien \(CMM\)](#), oder [Masterschlüsselanbieter](#) oder [Schlüsselring](#) aus.

Es gibt zwei Arten von CMMs. Beide rufen Datenschlüssel (und verwandtes kryptografisches Material) ab, aber auf unterschiedliche Weise, wie folgt:

- Ein CMM ist mit einem Schlüsselbund verbunden (C oder JavaScript) oder einen Masterschlüssel-Anbieter (Java und Python). Wenn das SDK vom CMM Verschlüsselungs- oder Entschlüsselungsmaterialien anfordert, ruft das CMM die Materialien von seinem Schlüsselbund oder von seinem Masterschlüssel-Anbieter ab. In Java und Python verwendet der CMM die Masterschlüssel zum Generieren, Verschlüsseln oder Entschlüsseln der Datenschlüssel. In C und JavaScript generiert und verschlüsselt der Schlüsselbund die kryptografischen Materialien und gibt sie zurück.
- Ein Caching-CMM ist einem Cache zugeordnet, wie z. B. [Lokaler Cache](#) und ein zugrunde liegendes CMM. Wenn das SDK vom Caching-CMM kryptografische Materialien anfordert, versucht das Caching-CMM diese aus dem Cache zu holen. Wenn keine Übereinstimmung gefunden werden kann, fragt das Caching-CMM seinen zugrundeliegenden CMM nach den Materialien. Anschließend speichert er die neuen kryptografischen Materialien im Cache, bevor er sie an den Aufrufer zurückgibt.

Das Caching CMM erzwingt auch [Sicherheitsschwellenwerte](#) Die Sie für einen einzelnen Cache-Eintrag festgelegt haben. Da die Sicherheitsschwellen durch den Cache gespeicherten CMM gesetzt und durchgesetzt werden, können Sie jeden kompatiblen Cache verwenden, auch wenn der Cache nicht für empfindliches Material ausgelegt ist.

Was befindet sich in einem Datenschlüssel-Cache-Eintrag?

Das Datenschlüssel-Caching speichert Datenschlüssel und zugehörige kryptographische Materialien in einem Cache. Jeder Eintrag enthält die folgenden Elemente. Sie finden diese Informationen möglicherweise nützlich, wenn Sie entscheiden, ob Sie die Datenschlüssel-Caching-Funktion verwenden möchten und wenn Sie Sicherheitsschwellenwerte für einen cachenden kryptografischen Materialmanager (Caching-CMM) festlegen.

Im Cache gespeicherte Einträge für Verschlüsselungsanfragen

Die Einträge, die einem Datenschlüssel-Cache durch eine Verschlüsselungsoperation hinzugefügt werden, umfassen folgende Elemente:

- Klartext-Datenschlüssel
- Verschlüsselte Datenschlüssel (einen oder mehrere)
- [Verschlüsselungskontext](#)
- Nachrichtensignaturschlüssel (falls verwendet)
- [Algorithmen-Paket](#)
- Metadaten, einschließlich Nutzungszähler für die Durchsetzung von Sicherheitsschwellenwerten

Im Cache gespeicherte Einträge für Entschlüsselungsanfragen

Die Einträge, die einem Datenschlüssel-Cache durch eine Entschlüsselungsoperation hinzugefügt werden, umfassen folgende Elemente:

- Klartext-Datenschlüssel
- Signaturverifizierungsschlüssel (falls verwendet)
- Metadaten, einschließlich Nutzungszähler für die Durchsetzung von Sicherheitsschwellenwerten

Verschlüsselungskontext: Wie Cache-Einträge ausgewählt werden

Sie können in jeder Anfrage zur Verschlüsselung von Daten einen Verschlüsselungskontext angeben. Der Verschlüsselungskontext spielt eine spezielle Rolle beim Datenschlüssel-Caching. Sie können Untergruppen aus Datenschlüsseln in Ihrem Cache anlegen, auch wenn die Datenschlüssel aus demselben Caching-CMM stammen.

Ein [Verschlüsselungskontext](#) ist eine Gruppe von Schlüssel/Wert-Paaren mit zufälligen, nicht geheimen Daten. Bei der Entschlüsselung wird der Verschlüsselungskontext kryptographisch an die verschlüsselten Daten, sodass derselbe Verschlüsselungskontext zur Entschlüsselung der Daten benötigt wird. Im AWS Encryption SDK wird der Verschlüsselungskontext in der [verschlüsselten Nachricht](#) mit den verschlüsselten Daten und den Datenschlüsseln gespeichert.

Wenn Sie einen Datenschlüssel-Cache verwenden, können Sie auch den Verschlüsselungskontext verwenden, um bestimmte im Cache gespeicherte Datenschlüssel für Ihre Verschlüsselungsvorgänge auszuwählen. Der Verschlüsselungskontext wird im Cache-Eintrag mit dem Datenschlüssel gespeichert (er ist Teil der Cache-Eintrag-ID). Im Cache gespeicherte Datenschlüssel werden nur wiederverwendet, wenn ihre Verschlüsselungskontexte übereinstimmen. Wenn Sie bestimmte Datenschlüssel für eine Verschlüsselungsanfrage wiederverwenden möchten, geben Sie den gleichen Verschlüsselungskontext an. Wenn Sie diese Datenschlüssel vermeiden möchten, geben Sie einen anderen Verschlüsselungskontext an.

Der Verschlüsselungskontext ist immer optional, wird aber empfohlen. Wenn Sie in Ihrer Anfrage keinen Verschlüsselungskontext angeben, wird ein leerer Verschlüsselungskontext in die Cache-Eintrag-ID aufgenommen und jeder Anfrage zugeordnet.

Benutzt meine Anwendung zwischengespeicherte Datenschlüssel?

Das Datenschlüssel-Caching ist eine Optimierungsstrategie, die für bestimmte Anwendungen und Workloads sehr effektiv ist. Da es jedoch ein gewisses Risiko mit sich bringt, ist es wichtig zu bestimmen, wie effektiv es für Ihre Situation ist, und dann zu entscheiden, ob der Nutzen die Risiken überwiegt.

Da das Datenschlüssel-Caching Datenschlüssel wiederverwendet, ist der offensichtlichste Effekt die Verringerung der Anzahl der Aufrufe, um neue Datenschlüssel zu generieren. Wenn das Datenschlüssel-Caching implementiert ist, ruft das AWS Encryption SDK die AWS KMS-Operation `GenerateDataKey` nur auf, um den ursprünglichen Datenschlüssel zu erstellen und wenn der Cache fehlt. Das Caching verbessert jedoch die Leistung nur in Anwendungen, die

zahlreiche Datenschlüssel mit den gleichen Eigenschaften generieren, einschließlich des gleichen Verschlüsselungskontexts und des gleichen Algorithmen-Pakets.

Versuchen Sie die folgenden Techniken, um festzustellen, ob Ihre Implementierung des AWS Encryption SDK tatsächlich Datenschlüssel aus dem Cache verwendet.

- Überprüfen Sie in den Protokollen Ihrer Masterschlüssel-Infrastruktur die Häufigkeit von Aufrufen, um neue Datenschlüssel zu erstellen. Wenn das Datenschlüssel-Caching wirksam ist, sollte die Anzahl der Aufrufe zum Erstellen neuer Schlüssel spürbar fallen. Zum Beispiel, wenn Sie einen `generateDataKey` verwenden, suchen Sie die `CloudTrail`-Protokolle für [GenerateDataKey](#) auf.
- Vergleichen Sie die [verschlüsselten Nachrichten](#), die das AWS Encryption SDK als Antwort auf verschiedene Verschlüsselungsanforderungen zurückgibt. Zum Beispiel, wenn Sie den `decrypt` des AWS Encryption SDK for Java verwenden, vergleichen Sie den `ParsedCiphertext`-Objekt aus verschiedenen `decrypt`-Aufrufen. In der `decrypt` des AWS Encryption SDK for JavaScript vergleichen Sie den Inhalt des `decryptedDataKey`-Eigentums der `MessageHeader` aus. Wenn Datenschlüssel wiederverwendet werden, sind die verschlüsselten Datenschlüssel in der verschlüsselten Nachricht identisch.

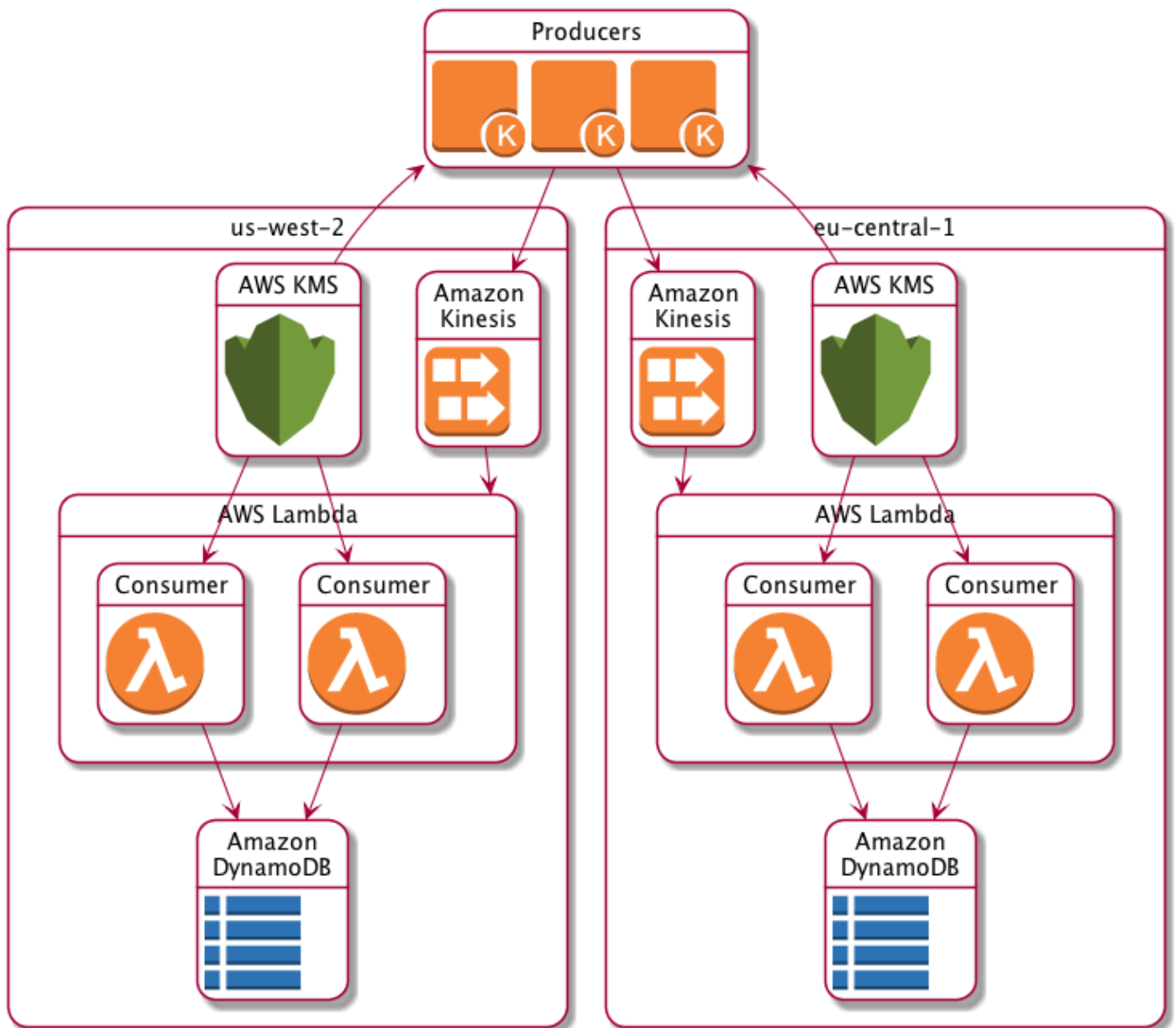
Beispiel für das Datenschlüssel-Caching

In diesem Beispiel wird verwendet [Datenschlüssel-Caching](#) mit einem [Lokaler Cache](#) um eine Anwendung zu beschleunigen, in der die Daten, die von mehreren Geräten generiert werden, in verschiedenen Regionen verschlüsselt und gespeichert werden.

In diesem Szenario generieren mehrere Datenproduzenten, verschlüsseln sie und schreiben sie in einen [Kinesis-Stream](#) in jeder Region. [AWS Lambda](#)-Funktionen (Verbraucher) entschlüsseln die Streams und schreiben Klartextdaten in eine `DynamoDB` Tabelle in der Region.

Datenproduzenten und -konsumenten verwenden den `decrypt` des AWS Encryption SDK und ein [AWS KMS-Masterschlüsselanbieter](#) aus. Um Anrufe von KMS zu reduzieren, hat jeder Produzent und Konsument seinen eigenen lokalen Cache.

Der Quellcode für diese Beispiele befindet sich in [Java und Python](#) aus. Das Beispiel enthält auch eine `AWS CloudFormation`-Vorlage, in der die Ressourcen für die Beispiele definiert sind.



Lokale Cache-Ergebnisse

Die folgende Tabelle zeigt, dass ein Lokaler Cache die Gesamtaufrufe an KMS (pro Sekunde pro Region) in diesem Beispiel auf 1% des ursprünglichen Wertes reduziert.

Produzentenanfragen

Anforderungen pro Sekunde pro Client			Clients pro Region	Durchschnittliche Anfragen pro
Datenschlüssel	Datenschlüssel	Gesamt (pro Region)		
üssele	üssel			Anfragen pro

	generieren (us-west-2)	verschlüsseln (eu-central-1)			Sekunde pro Region.
Kein Cache	1	1	1	500	500
Lokaler Cache	1 RPS/100 Anwendung en	1 RPS/100 Anwendung en	1 RPS/100 Anwendung en	500	5

Konsumentenfragen

	Anforderungen pro Sekunde pro Client			Client pro Region	Durchschn ittliche Anfragen pro Sekunde pro Region.
	Datenschl üssel entschlüsseln	Produzenten	Gesamtsum me		
Kein Cache	1 RPS pro Produzent	500	500	2	1.000
Lokaler Cache	1 RPS pro Produzent /100 Anwendung en	500	5	2	10

Beispielcode für das Zwischenspeichern von Datenschlüsseln

Dieses Codebeispiel erstellt eine einfache Implementierung von Datenschlüssel-Caching mit einem [lokalen Cache](#) in Java und Python. Der Code erstellt zwei Instanzen eines lokalen Caches: eine für [Datenproduzenten](#), die Daten verschlüsseln, und eine weitere für [Datenverbraucher](#) (AWS Lambda Funktionen), die Daten entschlüsseln. Einzelheiten zur Implementierung von Datenschlüssel-Caching in den einzelnen Sprachen finden Sie in der [Javadoc](#) - und [Python-Dokumentation](#) für AWS Encryption SDK

Das Zwischenspeichern von Datenschlüsseln ist für alle [Programmiersprachen](#) verfügbar, die von unterstützt werden. AWS Encryption SDK

Vollständige und getestete Beispiele für die Verwendung von Datenschlüssel-Caching in der finden Sie unter AWS Encryption SDK:

- C/C++: [caching_cmm.cpp](#)
- Java: [SimpleDataKeyCachingExample.java](#)
- JavaScript [Browser: caching_cmm.ts](#)
- JavaScript Node.js: [caching_cmm.ts](#)
- Python: [data_key_caching_basic.py](#)

Produzent

Der Producer ruft eine Map ab, konvertiert sie inJSON, verwendet sie, um sie AWS Encryption SDK zu verschlüsseln, und überträgt den Chiffretext-Datensatz jeweils in einen [Kinesis-Stream](#). AWS-Region

[Der Code definiert einen Cache-Manager für kryptografisches Material \(CachingCMM\) und ordnet ihn einem lokalen Cache und einem zugrunde liegenden Hauptschlüsselanbieter zu.](#) AWS KMS Beim CMM Zwischenspeichern werden die Datenschlüssel (und das [zugehörige kryptografische Material](#)) vom Hauptschlüsselanbieter zwischengespeichert. Außerdem interagiert es im Namen von mit dem Cache SDK und setzt die von Ihnen festgelegten Sicherheitsschwellenwerte durch.

Da beim Aufruf der Verschlüsselungsmethode eine Zwischenspeicherung CMM und nicht ein regulärer [Manager \(CMM\) oder ein Hauptschlüsselanbieter angegeben wird](#), wird bei der Verschlüsselung das Zwischenspeichern von Datenschlüsseln verwendet.

Java

Im folgenden Beispiel wird Version 2 verwendet. x der AWS-Verschlüsselungs-SDK for Java. Ausführung 3. x von der AWS-Verschlüsselungs-SDK for Java missbilligt das Zwischenspeichern von Datenschlüsseln. CMM Mit Version 3. x, Sie können auch den [AWS KMS hierarchischen Schlüsselbund](#) verwenden, eine alternative Lösung zum Zwischenspeichern kryptografischer Materialien.

```
/*
 * Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License"). You may not use
 * this file except
 * in compliance with the License. A copy of the License is located at
```

```
*
* http://aws.amazon.com/apache2.0
*
* or in the "license" file accompanying this file. This file is distributed on an
"AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
License for the
* specific language governing permissions and limitations under the License.
*/
package com.amazonaws.crypto.examples.kinesisdatakeycaching;

import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CommitmentPolicy;
import com.amazonaws.encryptionsdk.CryptoResult;
import com.amazonaws.encryptionsdk.MasterKeyProvider;
import com.amazonaws.encryptionsdk.caching.CachingCryptoMaterialsManager;
import com.amazonaws.encryptionsdk.caching.LocalCryptoMaterialsCache;
import com.amazonaws.encryptionsdk.kmsdkv2.KmsMasterKey;
import com.amazonaws.encryptionsdk.kmsdkv2.KmsMasterKeyProvider;
import com.amazonaws.encryptionsdk.multi.MultipleProviderFactory;
import com.amazonaws.util.json.Jackson;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.concurrent.TimeUnit;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisClient;
import software.amazon.awssdk.services.kms.KmsClient;

/**
 * Pushes data to Kinesis Streams in multiple Regions.
 */
public class MultiRegionRecordPusher {

    private static final long MAX_ENTRY_AGE_MILLISECONDS = 300000;
    private static final long MAX_ENTRY_USES = 100;
    private static final int MAX_CACHE_ENTRIES = 100;
    private final String streamName_;
    private final ArrayList<KinesisClient> kinesisClients_;
```



```
private final CachingCryptoMaterialsManager cachingMaterialsManager_;
private final AwsCrypto crypto_;

/**
 * Creates an instance of this object with Kinesis clients for all target
Regions and a cached
 * key provider containing KMS master keys in all target Regions.
 */
public MultiRegionRecordPusher(final Region[] regions, final String
kmsAliasName,
    final String streamName) {
    streamName_ = streamName;
    crypto_ = AwsCrypto.builder()
        .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
        .build();
    kinesisClients_ = new ArrayList<>();

    AwsCredentialsProvider credentialsProvider =
DefaultCredentialsProvider.builder().build();

    // Build KmsMasterKey and AmazonKinesisClient objects for each target region
List<KmsMasterKey> masterKeys = new ArrayList<>();
for (Region region : regions) {
    kinesisClients_.add(KinesisClient.builder()
        .credentialsProvider(credentialsProvider)
        .region(region)
        .build());

    KmsMasterKey regionMasterKey = KmsMasterKeyProvider.builder()
        .defaultRegion(region)
        .builderSupplier(() ->
KmsClient.builder().credentialsProvider(credentialsProvider))
        .buildStrict(kmsAliasName)
        .getMasterKey(kmsAliasName);

    masterKeys.add(regionMasterKey);
}

// Collect KmsMasterKey objects into single provider and add cache
MasterKeyProvider<?> masterKeyProvider =
MultipleProviderFactory.buildMultiProvider(
    KmsMasterKey.class,
    masterKeys
);
```

```

        cachingMaterialsManager_ = CachingCryptoMaterialsManager.newBuilder()
            .withMasterKeyProvider(masterKeyProvider)
            .withCache(new LocalCryptoMaterialsCache(MAX_CACHE_ENTRIES))
            .withMaxAge(MAX_ENTRY_AGE_MILLISECONDS, TimeUnit.MILLISECONDS)
            .withMessageUseLimit(MAX_ENTRY_USES)
            .build();
    }

    /**
     * JSON serializes and encrypts the received record data and pushes it to all
     target streams.
     */
    public void putRecord(final Map<Object, Object> data) {
        String partitionKey = UUID.randomUUID().toString();
        Map<String, String> encryptionContext = new HashMap<>();
        encryptionContext.put("stream", streamName_);

        // JSON serialize data
        String jsonData = Jackson.toJsonString(data);

        // Encrypt data
        CryptoResult<byte[], ?> result = crypto_.encryptData(
            cachingMaterialsManager_,
            jsonData.getBytes(),
            encryptionContext
        );
        byte[] encryptedData = result.getResult();

        // Put records to Kinesis stream in all Regions
        for (KinesisClient regionalKinesisClient : kinesisClients_) {
            regionalKinesisClient.putRecord(builder ->
                builder.streamName(streamName_)
                    .data(SdkBytes.fromByteArray(encryptedData))
                    .partitionKey(partitionKey));
        }
    }
}

```

Python

```

"""
Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.

```

Licensed under the Apache License, Version 2.0 (the "License"). You may not use this file except in compliance with the License. A copy of the License is located at

<https://aws.amazon.com/apache-2-0/>

or in the "license" file accompanying this file. This file is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

"""

```
import json
```

```
import uuid
```

```
from aws_encryption_sdk import EncryptionSDKClient, StrictAwsKmsMasterKeyProvider,
    CachingCryptoMaterialsManager, LocalCryptoMaterialsCache, CommitmentPolicy
from aws_encryption_sdk.key_providers.kms import KMSMasterKey
import boto3
```

```
class MultiRegionRecordPusher(object):
```

```
    """Pushes data to Kinesis Streams in multiple Regions."""
```

```
    CACHE_CAPACITY = 100
```

```
    MAX_ENTRY_AGE_SECONDS = 300.0
```

```
    MAX_ENTRY_MESSAGES_ENCRYPTED = 100
```

```
    def __init__(self, regions, kms_alias_name, stream_name):
```

```
        self._kinesis_clients = []
```

```
        self._stream_name = stream_name
```

```
        # Set up EncryptionSDKClient
```

```
        _client =
```

```
EncryptionSDKClient(CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT)
```

```
        # Set up KMSMasterKeyProvider with cache
```

```
        _key_provider = StrictAwsKmsMasterKeyProvider(kms_alias_name)
```

```
        # Add MasterKey and Kinesis client for each Region
```

```
        for region in regions:
```

```
            self._kinesis_clients.append(boto3.client('kinesis',
region_name=region))
```

```
            regional_master_key = KMSMasterKey(
```

```
        client=boto3.client('kms', region_name=region),
        key_id=kms_alias_name
    )
    _key_provider.add_master_key_provider(regional_master_key)

    cache = LocalCryptoMaterialsCache(capacity=self.CACHE_CAPACITY)
    self._materials_manager = CachingCryptoMaterialsManager(
        master_key_provider=_key_provider,
        cache=cache,
        max_age=self.MAX_ENTRY_AGE_SECONDS,
        max_messages_encrypted=self.MAX_ENTRY_MESSAGES_ENCRYPTED
    )

    def put_record(self, record_data):
        """JSON serializes and encrypts the received record data and pushes it to
        all target streams.

        :param dict record_data: Data to write to stream
        """
        # Kinesis partition key to randomize write load across stream shards
        partition_key = uuid.uuid4().hex

        encryption_context = {'stream': self._stream_name}

        # JSON serialize data
        json_data = json.dumps(record_data)

        # Encrypt data
        encrypted_data, _header = _client.encrypt(
            source=json_data,
            materials_manager=self._materials_manager,
            encryption_context=encryption_context
        )

        # Put records to Kinesis stream in all Regions
        for client in self._kinesis_clients:
            client.put_record(
                StreamName=self._stream_name,
                Data=encrypted_data,
                PartitionKey=partition_key
            )
```

Konsument

Der Datenverbraucher ist eine [AWS Lambda](#)-Funktion, die durch [Kinesis-Ereignisse](#) ausgelöst wird. Es entschlüsselt und deserialisiert jeden Datensatz und schreibt den Klartext-Datensatz in eine [Amazon DynamoDB-Tabelle](#) in derselben Region.

Wie der Herstellercode ermöglicht auch der Verbrauchercode das Zwischenspeichern von Datenschlüsseln, indem er bei Aufrufen der Entschlüsselungsmethode einen Manager für kryptografisches Material (Caching) für die Zwischenspeicherung verwendet. CMM

Der Java-Code erstellt einen Hauptschlüsselanbieter im strikten Modus mit einem bestimmten Wert. AWS KMS key Der strikte Modus ist beim Entschlüsseln nicht erforderlich, hat sich aber [bewährt](#). Der Python-Code verwendet den Discovery-Modus, der es ermöglicht, jeden Wrapping-Schlüssel zu AWS Encryption SDK verwenden, der einen Datenschlüssel verschlüsselt hat, um ihn zu entschlüsseln.

Java

Im folgenden Beispiel wird Version 2 verwendet. x der AWS-Verschlüsselungs-SDK for Java. Ausführung 3. x von der AWS-Verschlüsselungs-SDK for Java missbilligt das Zwischenspeichern von Datenschlüsseln. CMM Mit Version 3. x, Sie können auch den [AWS KMS hierarchischen Schlüsselbund](#) verwenden, eine alternative Lösung zum Zwischenspeichern kryptografischer Materialien.

Dieser Code erstellt einen Hauptschlüsselanbieter für die Entschlüsselung im strikten Modus. Er AWS Encryption SDK kann nur den von AWS KMS keys Ihnen angegebenen verwenden, um Ihre Nachricht zu entschlüsseln.

```
/*
 * Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License"). You may not use
 * this file except
 * in compliance with the License. A copy of the License is located at
 *
 * http://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed on an
 * "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
 * License for the
 * specific language governing permissions and limitations under the License.
 */
```

```
package com.amazonaws.crypto.examples.kinesisdatakeycaching;

import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CommitmentPolicy;
import com.amazonaws.encryptionsdk.CryptoResult;
import com.amazonaws.encryptionsdk.caching.CachingCryptoMaterialsManager;
import com.amazonaws.encryptionsdk.caching.LocalCryptoMaterialsCache;
import com.amazonaws.encryptionsdk.kmsdkv2.KmsMasterKeyProvider;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent.KinesisEventRecord;
import com.amazonaws.util.BinaryUtils;
import java.io.UnsupportedEncodingException;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.concurrent.TimeUnit;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;

/**
 * Decrypts all incoming Kinesis records and writes records to DynamoDB.
 */
public class LambdaDecryptAndWrite {

    private static final long MAX_ENTRY_AGE_MILLISECONDS = 600000;
    private static final int MAX_CACHE_ENTRIES = 100;
    private final CachingCryptoMaterialsManager cachingMaterialsManager_;
    private final AwsCrypto crypto_;
    private final DynamoDbTable<Item> table_;

    /**
     * Because the cache is used only for decryption, the code doesn't set the max
     bytes or max
     * message security thresholds that are enforced only on on data keys used for
     encryption.
     */
    public LambdaDecryptAndWrite() {
        String kmsKeyArn = System.getenv("CMK_ARN");
        cachingMaterialsManager_ = CachingCryptoMaterialsManager.newBuilder()

.withMasterKeyProvider(KmsMasterKeyProvider.builder().buildStrict(kmsKeyArn))
        .withCache(new LocalCryptoMaterialsCache(MAX_CACHE_ENTRIES))
        .withMaxAge(MAX_ENTRY_AGE_MILLISECONDS, TimeUnit.MILLISECONDS)
    }
}
```

```
        .build();

    crypto_ = AwsCrypto.builder()
        .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
        .build();

    String tableName = System.getenv("TABLE_NAME");
    DynamoDbEnhancedClient dynamodb = DynamoDbEnhancedClient.builder().build();
    table_ = dynamodb.table(tableName, TableSchema.fromClass(Item.class));
}

/**
 * @param event
 * @param context
 */
public void handleRequest(KinesisEvent event, Context context)
    throws UnsupportedOperationException {
    for (KinesisEventRecord record : event.getRecords()) {
        ByteBuffer ciphertextBuffer = record.getKinesis().getData();
        byte[] ciphertext = BinaryUtils.copyAllBytesFrom(ciphertextBuffer);

        // Decrypt and unpack record
        CryptoResult<byte[], ?> plaintextResult =
crypto_.decryptData(cachingMaterialsManager_,
                    ciphertext);

        // Verify the encryption context value
        String streamArn = record.getEventSourceARN();
        String streamName = streamArn.substring(streamArn.indexOf("/") + 1);
        if (!
streamName.equals(plaintextResult.getEncryptionContext().get("stream"))) {
            throw new IllegalStateException("Wrong Encryption Context!");
        }

        // Write record to DynamoDB
        String jsonItem = new String(plaintextResult.getResult(),
StandardCharsets.UTF_8);
        System.out.println(jsonItem);
        table_.putItem(Item.fromJSON(jsonItem));
    }
}

private static class Item {
```

```
        static Item fromJSON(String jsonText) {
            // Parse JSON and create new Item
            return new Item();
        }
    }
}
```

Python

Dieser Python-Code wird mit einem Master-Key-Anbieter im Discovery-Modus entschlüsselt. Es ermöglicht die AWS Encryption SDK Verwendung eines beliebigen Umschließungsschlüssels, der einen Datenschlüssel verschlüsselt hat, um ihn zu entschlüsseln. Der strikte Modus, in dem Sie die Umschließungsschlüssel angeben, die für die Entschlüsselung verwendet werden können, ist eine [bewährte](#) Methode.

```
"""
Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Licensed under the Apache License, Version 2.0 (the "License"). You may not use this
file except
in compliance with the License. A copy of the License is located at

https://aws.amazon.com/apache-2-0/

or in the "license" file accompanying this file. This file is distributed on an "AS
IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
License for the
specific language governing permissions and limitations under the License.
"""

import base64
import json
import logging
import os

from aws_encryption_sdk import EncryptionSDKClient,
    DiscoveryAwsKmsMasterKeyProvider, CachingCryptoMaterialsManager,
    LocalCryptoMaterialsCache, CommitmentPolicy
import boto3

_LOGGER = logging.getLogger(__name__)
_is_setup = False
CACHE_CAPACITY = 100
```



```
MAX_ENTRY_AGE_SECONDS = 600.0

def setup():
    """Sets up clients that should persist across Lambda invocations."""
    global encryption_sdk_client
    encryption_sdk_client =
    EncryptionSDKClient(CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT)

    global materials_manager
    key_provider = DiscoveryAwsKmsMasterKeyProvider()
    cache = LocalCryptoMaterialsCache(capacity=CACHE_CAPACITY)

    # Because the cache is used only for decryption, the code doesn't set
    # the max bytes or max message security thresholds that are enforced
    # only on on data keys used for encryption.
    materials_manager = CachingCryptoMaterialsManager(
        master_key_provider=key_provider,
        cache=cache,
        max_age=MAX_ENTRY_AGE_SECONDS
    )
    global table
    table_name = os.environ.get('TABLE_NAME')
    table = boto3.resource('dynamodb').Table(table_name)
    global _is_setup
    _is_setup = True

def lambda_handler(event, context):
    """Decrypts all incoming Kinesis records and writes records to DynamoDB."""
    _LOGGER.debug('New event:')
    _LOGGER.debug(event)
    if not _is_setup:
        setup()
    with table.batch_writer() as batch:
        for record in event.get('Records', []):
            # Record data base64-encoded by Kinesis
            ciphertext = base64.b64decode(record['kinesis']['data'])

            # Decrypt and unpack record
            plaintext, header = encryption_sdk_client.decrypt(
                source=ciphertext,
                materials_manager=materials_manager
            )
            item = json.loads(plaintext)
```

```
# Verify the encryption context value
stream_name = record['eventSourceARN'].split('/', 1)[1]
if stream_name != header.encryption_context['stream']:
    raise ValueError('Wrong Encryption Context!')

# Write record to DynamoDB
batch.put_item(Item=item)
```

Beispiel für das Datenschlüssel-Caching: AWS CloudFormationMuster

Dieser AWS CloudFormation-Vorlage richtet alles Notwendige ein AWS Ressourcen zur Reproduktion des [Beispiel für das Datenschlüssel-Caching](#) aus.

JSON

```
{
  "Parameters": {
    "SourceCodeBucket": {
      "Type": "String",
      "Description": "S3 bucket containing Lambda source code zip files"
    },
    "PythonLambdaS3Key": {
      "Type": "String",
      "Description": "S3 key containing Python Lambda source code zip file"
    },
    "PythonLambdaObjectVersionId": {
      "Type": "String",
      "Description": "S3 version id for S3 key containing Python Lambda source
code zip file"
    },
    "JavaLambdaS3Key": {
      "Type": "String",
      "Description": "S3 key containing Python Lambda source code zip file"
    },
    "JavaLambdaObjectVersionId": {
      "Type": "String",
      "Description": "S3 version id for S3 key containing Python Lambda source
code zip file"
    },
    "KeyAliasSuffix": {
```

```
    "Type": "String",
    "Description": "Suffix to use for KMS key Alias (ie: alias/
<KeyAliasSuffix>)"
  },
  "StreamName": {
    "Type": "String",
    "Description": "Name to use for Kinesis Stream"
  }
},
"Resources": {
  "InputStream": {
    "Type": "AWS::Kinesis::Stream",
    "Properties": {
      "Name": {
        "Ref": "StreamName"
      },
      "ShardCount": 2
    }
  },
  "PythonLambdaOutputTable": {
    "Type": "AWS::DynamoDB::Table",
    "Properties": {
      "AttributeDefinitions": [
        {
          "AttributeName": "id",
          "AttributeType": "S"
        }
      ],
      "KeySchema": [
        {
          "AttributeName": "id",
          "KeyType": "HASH"
        }
      ],
      "ProvisionedThroughput": {
        "ReadCapacityUnits": 1,
        "WriteCapacityUnits": 1
      }
    }
  },
  "PythonLambdaRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
```

```

    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
          "Service": "lambda.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
      }
    ],
    "ManagedPolicyArns": [
      "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
    ],
    "Policies": [
      {
        "PolicyName": "PythonLambdaAccess",
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:BatchWriteItem"
              ],
              "Resource": {
                "Fn::Sub": "arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${PythonLambdaOutputTable}"
              }
            },
            {
              "Effect": "Allow",
              "Action": [
                "dynamodb:PutItem"
              ],
              "Resource": {
                "Fn::Sub": "arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${PythonLambdaOutputTable}*"
              }
            },
            {
              "Effect": "Allow",

```

```

        "Action": [
            "kinesis:GetRecords",
            "kinesis:GetShardIterator",
            "kinesis:DescribeStream",
            "kinesis:ListStreams"
        ],
        "Resource": {
            "Fn::Sub": "arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}"
        }
    ]
}
}
}
},
"PythonLambdaFunction": {
    "Type": "AWS::Lambda::Function",
    "Properties": {
        "Description": "Python consumer",
        "Runtime": "python2.7",
        "MemorySize": 512,
        "Timeout": 90,
        "Role": {
            "Fn::GetAtt": [
                "PythonLambdaRole",
                "Arn"
            ]
        },
        "Handler":
"aws_crypto_examples.kinesis_datakey_caching.consumer.lambda_handler",
        "Code": {
            "S3Bucket": {
                "Ref": "SourceCodeBucket"
            },
            "S3Key": {
                "Ref": "PythonLambdaS3Key"
            },
            "S3ObjectVersion": {
                "Ref": "PythonLambdaObjectVersionId"
            }
        },
        "Environment": {

```

```

        "Variables": {
            "TABLE_NAME": {
                "Ref": "PythonLambdaOutputTable"
            }
        }
    },
    "PythonLambdaSourceMapping": {
        "Type": "AWS::Lambda::EventSourceMapping",
        "Properties": {
            "BatchSize": 1,
            "Enabled": true,
            "EventSourceArn": {
                "Fn::Sub": "arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}"
            },
            "FunctionName": {
                "Ref": "PythonLambdaFunction"
            },
            "StartingPosition": "TRIM_HORIZON"
        }
    },
    "JavaLambdaOutputTable": {
        "Type": "AWS::DynamoDB::Table",
        "Properties": {
            "AttributeDefinitions": [
                {
                    "AttributeName": "id",
                    "AttributeType": "S"
                }
            ],
            "KeySchema": [
                {
                    "AttributeName": "id",
                    "KeyType": "HASH"
                }
            ],
            "ProvisionedThroughput": {
                "ReadCapacityUnits": 1,
                "WriteCapacityUnits": 1
            }
        }
    },
},

```

```

    "JavaLambdaRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Service": "lambda.amazonaws.com"
              },
              "Action": "sts:AssumeRole"
            }
          ]
        },
        "ManagedPolicyArns": [
          "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
        ],
        "Policies": [
          {
            "PolicyName": "JavaLambdaAccess",
            "PolicyDocument": {
              "Version": "2012-10-17",
              "Statement": [
                {
                  "Effect": "Allow",
                  "Action": [
                    "dynamodb:DescribeTable",
                    "dynamodb:BatchWriteItem"
                  ],
                  "Resource": {
                    "Fn::Sub": "arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${JavaLambdaOutputTable}"
                  }
                },
                {
                  "Effect": "Allow",
                  "Action": [
                    "dynamodb:PutItem"
                  ],
                  "Resource": {
                    "Fn::Sub": "arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${JavaLambdaOutputTable}*"

```

```

    }
    },
    {
        "Effect": "Allow",
        "Action": [
            "kinesis:GetRecords",
            "kinesis:GetShardIterator",
            "kinesis:DescribeStream",
            "kinesis:ListStreams"
        ],
        "Resource": {
            "Fn::Sub": "arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}"
        }
    }
]
}
}
]
}
},
"JavaLambdaFunction": {
    "Type": "AWS::Lambda::Function",
    "Properties": {
        "Description": "Java consumer",
        "Runtime": "java8",
        "MemorySize": 512,
        "Timeout": 90,
        "Role": {
            "Fn::GetAtt": [
                "JavaLambdaRole",
                "Arn"
            ]
        },
        "Handler":
"com.amazonaws.crypto.examples.kinesisdatakeycaching.LambdaDecryptAndWrite::handleRequest",
        "Code": {
            "S3Bucket": {
                "Ref": "SourceCodeBucket"
            },
            "S3Key": {
                "Ref": "JavaLambdaS3Key"
            },
            "S3ObjectVersion": {

```



```

        "Ref": "JavaLambdaObjectVersionId"
      }
    },
    "Environment": {
      "Variables": {
        "TABLE_NAME": {
          "Ref": "JavaLambdaOutputTable"
        },
        "CMK_ARN": {
          "Fn::GetAtt": [
            "RegionKinesisCMK",
            "Arn"
          ]
        }
      }
    }
  },
  "JavaLambdaSourceMapping": {
    "Type": "AWS::Lambda::EventSourceMapping",
    "Properties": {
      "BatchSize": 1,
      "Enabled": true,
      "EventSourceArn": {
        "Fn::Sub": "arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}"
      },
      "FunctionName": {
        "Ref": "JavaLambdaFunction"
      },
      "StartingPosition": "TRIM_HORIZON"
    }
  },
  "RegionKinesisCMK": {
    "Type": "AWS::KMS::Key",
    "Properties": {
      "Description": "Used to encrypt data passing through Kinesis Stream
in this region",
      "Enabled": true,
      "KeyPolicy": {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",

```

```

    "Principal": {
      "AWS": {
        "Fn::Sub": "arn:aws:iam::${AWS::AccountId}:root"
      }
    },
    "Action": [
      "kms:Encrypt",
      "kms:GenerateDataKey",
      "kms:CreateAlias",
      "kms>DeleteAlias",
      "kms:DescribeKey",
      "kms:DisableKey",
      "kms:EnableKey",
      "kms:PutKeyPolicy",
      "kms:ScheduleKeyDeletion",
      "kms:UpdateAlias",
      "kms:UpdateKeyDescription"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Principal": {
      "AWS": [
        {
          "Fn::GetAtt": [
            "PythonLambdaRole",
            "Arn"
          ]
        },
        {
          "Fn::GetAtt": [
            "JavaLambdaRole",
            "Arn"
          ]
        }
      ]
    },
    "Action": "kms:Decrypt",
    "Resource": "*"
  }
]
}

```

```

    },
    "RegionKinesisCMKAlias": {
      "Type": "AWS::KMS::Alias",
      "Properties": {
        "AliasName": {
          "Fn::Sub": "alias/${KeyAliasSuffix}"
        },
        "TargetKeyId": {
          "Ref": "RegionKinesisCMK"
        }
      }
    }
  }
}

```

YAML

```

Parameters:
  SourceCodeBucket:
    Type: String
    Description: S3 bucket containing Lambda source code zip files
  PythonLambdaS3Key:
    Type: String
    Description: S3 key containing Python Lambda source code zip file
  PythonLambdaObjectVersionId:
    Type: String
    Description: S3 version id for S3 key containing Python Lambda source code
zip file
  JavaLambdaS3Key:
    Type: String
    Description: S3 key containing Python Lambda source code zip file
  JavaLambdaObjectVersionId:
    Type: String
    Description: S3 version id for S3 key containing Python Lambda source code
zip file
  KeyAliasSuffix:
    Type: String
    Description: 'Suffix to use for KMS CMK Alias (ie: alias/<KeyAliasSuffix>)'
  StreamName:
    Type: String
    Description: Name to use for Kinesis Stream
Resources:
  InputStream:

```

```
Type: AWS::Kinesis::Stream
Properties:
  Name: !Ref StreamName
  ShardCount: 2
PythonLambdaOutputTable:
  Type: AWS::DynamoDB::Table
  Properties:
    AttributeDefinitions:
      -
        AttributeName: id
        AttributeType: S
    KeySchema:
      -
        AttributeName: id
        KeyType: HASH
    ProvisionedThroughput:
      ReadCapacityUnits: 1
      WriteCapacityUnits: 1
PythonLambdaRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: 2012-10-17
      Statement:
        -
          Effect: Allow
          Principal:
            Service: lambda.amazonaws.com
          Action: sts:AssumeRole
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
    Policies:
      -
        PolicyName: PythonLambdaAccess
        PolicyDocument:
          Version: 2012-10-17
          Statement:
            -
              Effect: Allow
              Action:
                - dynamodb:DescribeTable
                - dynamodb:BatchWriteItem
              Resource: !Sub arn:aws:dynamodb:${AWS::Region}:
                ${AWS::AccountId}:table/${PythonLambdaOutputTable}
```

```

      -
        Effect: Allow
        Action:
          - dynamodb:PutItem
        Resource: !Sub arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${PythonLambdaOutputTable}*
      -
        Effect: Allow
        Action:
          - kinesis:GetRecords
          - kinesis:GetShardIterator
          - kinesis:DescribeStream
          - kinesis:ListStreams
        Resource: !Sub arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}
    PythonLambdaFunction:
      Type: AWS::Lambda::Function
      Properties:
        Description: Python consumer
        Runtime: python2.7
        MemorySize: 512
        Timeout: 90
        Role: !GetAtt PythonLambdaRole.Arn
        Handler:
aws_crypto_examples.kinesis_datakey_caching.consumer.lambda_handler
        Code:
          S3Bucket: !Ref SourceCodeBucket
          S3Key: !Ref PythonLambdaS3Key
          S3ObjectVersion: !Ref PythonLambdaObjectVersionId
        Environment:
          Variables:
            TABLE_NAME: !Ref PythonLambdaOutputTable
    PythonLambdaSourceMapping:
      Type: AWS::Lambda::EventSourceMapping
      Properties:
        BatchSize: 1
        Enabled: true
        EventSourceArn: !Sub arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}
        FunctionName: !Ref PythonLambdaFunction
        StartingPosition: TRIM_HORIZON
    JavaLambdaOutputTable:
      Type: AWS::DynamoDB::Table
      Properties:

```

```

AttributeDefinitions:
  -
    AttributeName: id
    AttributeType: S
KeySchema:
  -
    AttributeName: id
    KeyType: HASH
ProvisionedThroughput:
  ReadCapacityUnits: 1
  WriteCapacityUnits: 1
JavaLambdaRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: 2012-10-17
      Statement:
        -
          Effect: Allow
          Principal:
            Service: lambda.amazonaws.com
          Action: sts:AssumeRole
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
    Policies:
      -
        PolicyName: JavaLambdaAccess
        PolicyDocument:
          Version: 2012-10-17
          Statement:
            -
              Effect: Allow
              Action:
                - dynamodb:DescribeTable
                - dynamodb:BatchWriteItem
              Resource: !Sub arn:aws:dynamodb:${AWS::Region}:
                ${AWS::AccountId}:table/${JavaLambdaOutputTable}
            -
              Effect: Allow
              Action:
                - dynamodb:PutItem
              Resource: !Sub arn:aws:dynamodb:${AWS::Region}:
                ${AWS::AccountId}:table/${JavaLambdaOutputTable}*

```

```

        Effect: Allow
        Action:
            - kinesis:GetRecords
            - kinesis:GetShardIterator
            - kinesis:DescribeStream
            - kinesis:ListStreams
        Resource: !Sub arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}
    JavaLambdaFunction:
        Type: AWS::Lambda::Function
        Properties:
            Description: Java consumer
            Runtime: java8
            MemorySize: 512
            Timeout: 90
            Role: !GetAtt JavaLambdaRole.Arn
            Handler:
com.amazonaws.crypto.examples.kinesisdatakeycaching.LambdaDecryptAndWrite::handleRequest
            Code:
                S3Bucket: !Ref SourceCodeBucket
                S3Key: !Ref JavaLambdaS3Key
                S3ObjectVersion: !Ref JavaLambdaObjectVersionId
            Environment:
                Variables:
                    TABLE_NAME: !Ref JavaLambdaOutputTable
                    CMK_ARN: !GetAtt RegionKinesisCMK.Arn
    JavaLambdaSourceMapping:
        Type: AWS::Lambda::EventSourceMapping
        Properties:
            BatchSize: 1
            Enabled: true
            EventSourceArn: !Sub arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}
            FunctionName: !Ref JavaLambdaFunction
            StartingPosition: TRIM_HORIZON
    RegionKinesisCMK:
        Type: AWS::KMS::Key
        Properties:
            Description: Used to encrypt data passing through Kinesis Stream in this
region
            Enabled: true
            KeyPolicy:
                Version: 2012-10-17
                Statement:

```

```
-
  Effect: Allow
  Principal:
    AWS: !Sub arn:aws:iam::${AWS::AccountId}:root
  Action:
    # Data plane actions
    - kms:Encrypt
    - kms:GenerateDataKey
    # Control plane actions
    - kms:CreateAlias
    - kms>DeleteAlias
    - kms:DescribeKey
    - kms:DisableKey
    - kms:EnableKey
    - kms:PutKeyPolicy
    - kms:ScheduleKeyDeletion
    - kms:UpdateAlias
    - kms:UpdateKeyDescription
  Resource: '*'
-
  Effect: Allow
  Principal:
    AWS:
      - !GetAtt PythonLambdaRole.Arn
      - !GetAtt JavaLambdaRole.Arn
  Action: kms:Decrypt
  Resource: '*'
RegionKinesisCMKAlias:
  Type: AWS::KMS::Alias
  Properties:
    AliasName: !Sub alias/${KeyAliasSuffix}
    TargetKeyId: !Ref RegionKinesisCMK
```


Versionen der AWS Encryption SDK

Die AWS Encryption SDK Sprachimplementierungen verwenden [semantische Versionierung](#), damit Sie den Umfang der Änderungen in den einzelnen Versionen leichter erkennen können. Eine Änderung der Hauptversionsnummer, z. B. 1. x. x bis 2. x. x steht für eine grundlegende Änderung, die wahrscheinlich Codeänderungen und eine geplante Bereitstellung erfordert. Wichtige Änderungen in einer neuen Version wirken sich möglicherweise nicht auf jeden Anwendungsfall aus. Lesen Sie in den Versionshinweisen nach, ob Sie davon betroffen sind. Eine Änderung in einer Nebenversion, wie z. B. x. x auf x .2. x ist immer abwärtskompatibel, kann aber veraltete Elemente enthalten.

Verwenden Sie nach Möglichkeit die neueste Version von AWS Encryption SDK in der von Ihnen gewählten Programmiersprache. Die [Wartungs- und Supportrichtlinien](#) für jede Version unterscheiden sich je nach Implementierung der Programmiersprache. Einzelheiten zu den unterstützten Versionen in Ihrer bevorzugten Programmiersprache finden Sie in der SUPPORT_POLICY.rst Datei im zugehörigen [GitHubRepository](#).

Wenn Upgrades neue Funktionen beinhalten, die eine spezielle Konfiguration erfordern, um Verschlüsselungs- oder Entschlüsselungsfehler zu vermeiden, stellen wir eine Zwischenversion und detaillierte Anweisungen zu deren Verwendung zur Verfügung. Zum Beispiel Versionen 1.7. x und 1.8. x sind als Übergangsversionen konzipiert, mit denen Sie ein Upgrade von Versionen vor 1.7 durchführen können. x auf Versionen 2.0. x und später. Details hierzu finden Sie unter [Migrieren Sie IhreAWS Encryption SDK](#).

Note

Das X in einer Versionsnummer steht für einen beliebigen Patch der Haupt- und Nebenversion. Zum Beispiel Version 1.7. x steht für alle Versionen, die mit 1.7 beginnen, einschließlich 1.7.1 und 1.7.9.

Neue Sicherheitsfunktionen wurden ursprünglich in den AWS CLI Verschlüsselungsversionen 1.7 veröffentlicht. x und 2.0. x. AWS CLIVerschlüsselungsversion 1.8 jedoch. x ersetzt Version 1.7. x und AWS Verschlüsselung CLI 2.1. x ersetzt 2.0. x. Einzelheiten finden Sie in der entsprechenden [Sicherheitsempfehlung](#) im [aws-encryption-sdk-cliRepository](#) unter GitHub.

Die folgenden Tabellen bieten einen Überblick über die wichtigsten Unterschiede zwischen den unterstützten Versionen von AWS Encryption SDK für die einzelnen Programmiersprachen.

C

Eine ausführliche Beschreibung aller Änderungen finden Sie in der Datei [CHANGELOG.md](#) im [aws-encryption-sdk-c](#)Repository unter GitHub.

Hauptversion	Details	SDKLebenszyklusphase der Hauptversion	
1.x	1,0	Ende der Unterstützungsphase	
	1,7		Erstversion. Updates für, AWS Encryption SDK die Benutzern früherer Versionen helfen, auf Versionen 2.0 zu aktualisieren. x und später. Weitere Informationen finden Sie in Version 1.7. x .
2.x	2.0	Allgemeine Verfügbarkeit (GA)	
	2.2		Aktualisierungen der AWS Encryption SDK. Weitere Informationen finden Sie unter Version 2.0. x . Verbesserungen am Prozess der Nachricht enentschlüsselung.
	2.3		Fügt Unterstützung für Schlüssel AWS KMS mit mehreren Regionen hinzu.

C#/. NET

Eine ausführliche Beschreibung aller Änderungen finden Sie in der Datei [CHANGELOG.md](#) im [aws-encryption-sdk-net](#) Repository unter GitHub.

Hauptversion	Details	SDKLebenszyklusphase der Hauptversion
3.x	3.0	Erstversion. Allgemeine Verfügbarkeit (GA) Version 3.x des AWS Encryption SDK für .NET wird am 13. Mai 2024 in den Wartungsmodus wechseln.
4.x	4,0	Allgemeine Verfügbarkeit (GA) Integriert die Unterstützung für den AWS KMS hierarchischen Schlüsselbund, den erforderlichen Verschlüsselungskontext und asymmetrische CMM Schlüsselbunde. RSA AWS KMS

Befehlszeilenschnittstelle (CLI)

Eine ausführliche Beschreibung aller Änderungen finden Sie unter [Versionen des AWS Verschlüsselung CLI](#) und in der Datei [CHANGELOG.rst](#) im [aws-encryption-sdk-cli](#) Repository. GitHub

Hauptversion	Details	SDKLebenszyklusphase der Hauptversion
1.x	1,0	Erstversion.
	1,7	Updates für, AWS Encryption SDK die Benutzern früherer Versionen helfen, auf Versionen 2.0 zu aktualisieren. x und später. Weitere Informationen finden Sie in Version 1.7. x .
2.x	2.0	Aktualisierungen der AWS Encryption SDK. Weitere Informationen finden Sie unter Version 2.0. x .
	2.1	Entfernt den <code>--discovery</code> Parameter und ersetzt ihn durch das <code>discovery</code> Attribut des Parameters <code>--wrapping-keys</code> . Version 2.1.0 der AWS Verschlüsselung CLI entspricht Version 2.0 in anderen Programmiersprachen.

[Ende der Unterstützungphase](#)

[Ende der Unterstützungphase](#)

	2.2	Verbesserungen am Prozess der Nachricht enentschlüsselung.	
3.x	3.0	Fügt Unterstützung für Schlüssel AWS KMS mit mehreren Regionen hinzu.	Ende der Unterstützungsphase
4.x	4,0	Die AWS Verschlüsselung unterstützt Python 2 oder Python 3.4 nicht CLI mehr. Ab Hauptversion 4. x der AWS VerschlüsselungCLI, nur Python 3.5 oder höher wird unterstützt.	Allgemeine Verfügbarkeit (GA)
	4.1	Die AWS Verschlüsselung unterstützt Python 3.5 nicht CLI mehr. Ab Version 4.1. x der AWS VerschlüsselungCLI, nur Python 3.6 oder höher wird unterstützt.	
	4.2	Die AWS Verschlüsselung unterstützt Python 3.6 nicht CLI mehr. Ab Version 4.2. x der AWS VerschlüsselungCLI, nur Python 3.7 oder höher wird unterstützt.	

Java

Eine ausführliche Beschreibung aller Änderungen finden Sie in der Datei [CHANGELOG.rst](#) im [aws-encryption-sdk-java](#) Repository unter. GitHub

Hauptversion	Details	SDKLebenszyklusphase der Hauptversion
1.x	1,0	Erstversion.
	1.3	Integriert die Unterstützung für den Manager für kryptografische Materialien und das Zwischenspeichern von Datenschlüsseln. Auf die deterministische IV-Generation umgestellt.
	1.6.1	Verwirft und <code>AwsCryptor.encryptString()</code> und ersetzt sie durch <code>AwsCryptor.decryptString()</code> und <code>AwsCryptor.encryptData()</code> <code>AwsCryptor.decryptData()</code>
	1,7	Updates für, AWS Encryption SDK die Benutzern früherer Versionen beim Upgrade auf Version

[Ende der Unterstützungsphase](#)

		2.0 helfen. x und später. Weitere Informationen finden Sie in Version 1.7. x .	
2.x	2.0	Aktualisierungen der AWS Encryption SDK. Weitere Informationen finden Sie unter Version 2.0. x .	Allgemeine Verfügbarkeit (GA) Version 2.x von AWS-Verschlüsselungs-SDK for Java wird 2024 in den Wartungsmodus wechseln.
	2.2	Verbesserungen beim Entschlüsselungsprozess von Nachrichten.	
	2.3	Fügt Unterstützung für Schlüssel AWS KMS mit mehreren Regionen hinzu.	
	2.4	Fügt Unterstützung für AWS SDK for Java 2.x hinzu.	

3.x	3.0	Integriert die Bibliothek AWS-Verschlüsselungs-SDK for Java mit den Materialanbietern.	Allgemeine Verfügbarkeit (GA)
		Integriert die Unterstützung für symmetrische und asymmetrische RSA AWS KMS Schlüsselringe, AWS KMS hierarchische Schlüsselringe, Raw-Schlüsselringe, AES Raw-Schlüsselringe, Mehrfachschlüsselringe und den RSA erforderlichen Verschlüsselungskontext. CMM	

JavaScript

Eine ausführliche Beschreibung aller Änderungen finden Sie in der Datei [CHANGELOG.md](#) im [aws-encryption-sdk-javascript](#) Repository unter GitHub.

Hauptversion	Details	SDKLebenszyklusphase der Hauptversion
1.x	1,0	Erstversion.
	1,7	Updates für, AWS Encryption SDK die Benutzern früherer Versionen helfen, auf Versionen 2.0

[Ende der Unterstützungsphase](#)

		zu aktualisieren. x und später. Weitere Informationen finden Sie in Version 1.7. x .	
2.x	2.0	Aktualisierungen der AWS Encryption SDK. Weitere Informationen finden Sie unter Version 2.0. x .	Ende der Unterstützungsphase
	2.2	Verbesserungen am Prozess der Nachricht enentschlüsselung.	
	2.3	Fügt Unterstützung für Schlüssel AWS KMS mit mehreren Regionen hinzu.	
3.x	3.0	Entfernt die CI-Abdeckung für Knoten 10. Führt ein Upgrade der Abhängigkeiten durch, sodass Knoten 8 und Knoten 10 nicht mehr unterstützt werden.	Wartung Der Support für Version 3.x von AWS-Verschlüsselungs-SDK for JavaScript endet am 17. Januar 2024.
4.x	4,0	Erfordert Version 3 der AWS-Verschlüsselungs-SDK for JavaScript s, kms-client um den AWS KMS Schlüssel bund verwenden zu können.	Allgemeine Verfügbarkeit (GA)

Python

Eine ausführliche Beschreibung aller Änderungen finden Sie in der Datei [CHANGELOG.rst](#) im [aws-encryption-sdk-python](#) Repository unter. GitHub

Hauptversion	Details	SDKLebenszyklusphase der Hauptversion
1.x	1,0	Erstversion.
	1.3	Integriert die Unterstützung für den Manager für kryptografische Materialien und das Zwischenspeichern von Datenschlüsseln. Auf die deterministische IV-Generation umgestellt.
	1,7	Updates für AWS Encryption SDK , die Benutzern früherer Versionen beim Upgrade auf Version 2.0 helfen. x und später. Weitere Informationen finden Sie in Version 1.7. x .
2.x	2.0	Aktualisierungen der AWS Encryption SDK. Weitere Informationen finden Sie unter Version 2.0. x .

	2.2	Verbesserungen am Prozess der Nachricht enentschlüsselung.	
	2.3	Fügt Unterstützung für Schlüssel AWS KMS mit mehreren Regionen hinzu.	
3.x	3.0	Das unterstützt Python 2 oder Python 3.4 nicht AWS-Verschlüsselungs-SDK for Python mehr. Ab Hauptversion 3. x von der AWS-Verschlüsselungs-SDK for Python, nur Python 3.5 oder höher wird unterstützt.	Allgemeine Verfügbarkeit (GA)

Einzelheiten zur Version

In der folgenden Liste werden die Hauptunterschiede zwischen den unterstützten Versionen von beschrieben AWS Encryption SDK.

Themen

- [Versionen vor 1.7. x](#)
- [Version 1.7. x](#)
- [Version 2.0. x](#)
- [Version 2.2. x](#)
- [Version 2.3. x](#)

Versionen vor 1.7. x

Note

Alle 1. x. x Versionen von AWS Encryption SDK befinden sich in der [end-of-supportPhase](#). Aktualisieren Sie so schnell wie möglich auf die neueste verfügbare Version von AWS Encryption SDK für Ihre Programmiersprache. Um ein Upgrade von einer AWS Encryption SDK Version vor 1.7 durchzuführen. x, Sie müssen zuerst auf 1.7 aktualisieren. x. Details hierzu finden Sie unter [Migrieren Sie IhreAWS Encryption SDK](#).

Versionen von AWS Encryption SDK früher als 1.7. x bieten wichtige Sicherheitsfunktionen, darunter die Verschlüsselung mit dem Advanced Encryption Standard-Algorithmus im Galois/Counter-Modus (AES-GCM), eine HMAC basierte extract-and-expand Schlüsselableitungsfunktion (HKDF), Signierung und einen 256-Bit-Verschlüsselungsschlüssel. [Diese Versionen unterstützen jedoch nicht die von uns empfohlenen Best Practices, einschließlich Key Commitment](#).

Version 1.7. x

Note

Alle 1. x. x Versionen von AWS Encryption SDK befinden sich in der [end-of-supportPhase](#).

Version 1.7. x wurde entwickelt, um Benutzern früherer Versionen von das Upgrade AWS Encryption SDK auf Version 2.0 zu erleichtern. x und später. Wenn Sie mit dem noch nicht vertraut sind AWS Encryption SDK, können Sie diese Version überspringen und mit der neuesten verfügbaren Version in Ihrer Programmiersprache beginnen.

Version 1.7. x ist vollständig abwärtskompatibel; es führt keine grundlegenden Änderungen ein und ändert auch nicht das Verhalten von. AWS Encryption SDK Es ist auch vorwärtskompatibel; es ermöglicht Ihnen, Ihren Code so zu aktualisieren, dass er mit Version 2.0 kompatibel ist. x. Es enthält neue Funktionen, aktiviert sie jedoch nicht vollständig. Und es erfordert Konfigurationswerte, die verhindern, dass Sie sofort alle neuen Funktionen übernehmen, bis Sie bereit sind.

Version 1.7. x beinhaltet die folgenden Änderungen:

AWS KMS Aktualisierungen des Master Key Providers (erforderlich)

Version 1.7. x führt neue Konstruktoren für AWS-Verschlüsselungs-SDK for Java und ein AWS-Verschlüsselungs-SDK for Python , die explizit AWS KMS Master-Key-Anbieter entweder im strikten Modus oder im Discovery-Modus erstellen. Diese Version fügt ähnliche Änderungen an der AWS Encryption SDK Befehlszeilenschnittstelle () hinzu. CLI Details hierzu finden Sie unter [Aktualisierung der AWS KMS Hauptschlüsselanbieter](#).

- Im strikten Modus benötigen AWS KMS Hauptschlüsselanbieter eine Liste von Wrapping-Schlüsseln, und sie verschlüsseln und entschlüsseln nur mit den von Ihnen angegebenen Wrapping-Schlüsseln. Dies ist eine AWS Encryption SDK bewährte Methode, mit der sichergestellt wird, dass Sie die Umschließungsschlüssel verwenden, die Sie verwenden möchten.
- Im Discovery-Modus AWS KMS akzeptieren Master-Key-Anbieter keine Wrap-Schlüssel. Sie können sie nicht zum Verschlüsseln verwenden. Beim Entschlüsseln können sie einen beliebigen Umschließungsschlüssel verwenden, um einen verschlüsselten Datenschlüssel zu entschlüsseln. Sie können jedoch die für die Entschlüsselung verwendeten Wrapping-Schlüssel auf diese beschränken. AWS-Konten Die Kontofilterung ist optional, aber es ist eine [bewährte Methode](#), die wir empfehlen.

Die Konstruktoren, die frühere Versionen von AWS KMS Master-Key-Providern erstellen, sind in Version 1.7 veraltet. x und wurde in Version 2.0 entfernt. x. Diese Konstruktoren instanziierten Masterschlüsselanbieter, die mit den von Ihnen angegebenen Wrapping-Schlüsseln verschlüsseln. Sie entschlüsseln jedoch verschlüsselte Datenschlüssel mithilfe des Umschließungsschlüssels, mit dem sie verschlüsselt wurden, ohne Rücksicht auf die angegebenen Umschließungsschlüssel. Benutzer können unbeabsichtigt Nachrichten mit Umschließungsschlüsseln entschlüsseln, die sie nicht verwenden wollen, auch AWS KMS keys in anderen AWS-Konten Regionen.

An den Konstruktoren für AWS KMS Hauptschlüssel wurden keine Änderungen vorgenommen. Beim Verschlüsseln und Entschlüsseln verwenden die AWS KMS Hauptschlüssel nur die, AWS KMS key die Sie angeben.

AWS KMS Aktualisierungen des Schlüsselbundes (optional)

Version 1.7. x fügt den AWS-Verschlüsselungs-SDK for JavaScript Implementierungen AWS-Verschlüsselungs-SDK for C und einen neuen Filter hinzu, der die [AWS KMS Erkennung von Schlüsselanhängern](#) auf bestimmte Bereiche beschränkt. AWS-Konten Dieser neue Kontofilter ist optional, aber es handelt sich dabei um eine [bewährte Methode](#), die wir empfehlen. Details hierzu finden Sie unter [Wird aktualisiertAWS KMSSchlüsselringe](#).

Es wurden keine Änderungen an den Konstruktoren für AWS KMS Schlüsselanhänger vorgenommen. AWS KMS Standardschlüsselringe verhalten sich im strikten Modus wie Hauptschlüsselanbieter. AWS KMS Discovery-Schlüsselringe werden explizit im Discovery-Modus erstellt.

Übergabe einer Schlüssel-ID an Decrypt AWS KMS

Ab Version 1.7. x, wenn verschlüsselte Datenschlüssel entschlüsselt werden, gibt der AWS KMS key in seinen Aufrufen der AWS KMS [Decrypt-Operation AWS Encryption SDK](#) immer an. Der AWS Encryption SDK ruft den Schlüssel-ID-Wert für AWS KMS key aus den Metadaten in jedem verschlüsselten Datenschlüssel ab. Für diese Funktion sind keine Codeänderungen erforderlich.

[Die Angabe der Schlüssel-ID von AWS KMS key ist nicht erforderlich, um Chiffretext zu entschlüsseln, der mit einem symmetrischen KMS Verschlüsselungsschlüssel verschlüsselt wurde, aber es hat sich bewährt.](#) [AWS KMS](#) Wie bei der Angabe von Wrapping Keys in Ihrem Schlüsselanbieter wird bei dieser Vorgehensweise sichergestellt, dass AWS KMS nur mit dem Wrapping-Schlüssel entschlüsselt wird, den Sie verwenden möchten.

Entschlüsseln Sie Chiffretext mit Schlüsselbindung

Version 1.7. x [kann Chiffretext entschlüsseln, der mit oder ohne Schlüsselbindung verschlüsselt wurde](#). Es kann jedoch keinen Chiffretext mit Schlüsselbindung verschlüsseln. Mit dieser Eigenschaft können Sie Anwendungen vollständig bereitstellen, die mit Key Commitment verschlüsselten Chiffretext entschlüsseln können, bevor sie jemals auf einen solchen Chiffretext stoßen. Da diese Version Nachrichten entschlüsselt, die ohne Schlüsselbindung verschlüsselt wurden, müssen Sie keinen Chiffretext erneut verschlüsseln.

Um dieses Verhalten zu implementieren, Version 1.7. x enthält eine neue Konfigurationseinstellung für [Commitment-Richtlinien, die festlegt, ob sie mit Key Commitment ver- oder entschlüsseln AWS Encryption SDK können](#). In Version 1.7. x, der einzig gültige Wert für die Commitment-Richtlinie `ForbidEncryptAllowDecrypt`, wird bei allen Verschlüsselungs- und Entschlüsselungsvorgängen verwendet. Dieser Wert verhindert, dass der AWS Encryption SDK mit einer der neuen Algorithmus-Suiten verschlüsselt wird, die Key Commitment beinhalten. Er ermöglicht die AWS Encryption SDK Entschlüsselung von Chiffretext mit und ohne Schlüsselbindung.

In Version 1.7 gibt es zwar nur einen gültigen Wert für die Verpflichtungspolitik. x, wir verlangen, dass Sie diesen Wert explizit festlegen können, wenn Sie den in dieser Version APIs eingeführten neuen Wert verwenden. Wenn Sie den Wert explizit festlegen, wird verhindert, dass sich Ihre Verpflichtungsrichtlinie automatisch ändert, `require-encrypt-require-decrypt` wenn Sie

auf Version 2.1 aktualisieren. x. Stattdessen können Sie [Ihre Verpflichtungspolitik schrittweise migrieren](#).

Algorithmus-Suiten mit hohem Engagement

Version 1.7. x enthält zwei neue [Algorithmus-Suiten](#), die Key Commitment unterstützen. Eine beinhaltet das Signieren, die andere nicht. Wie bereits früher unterstützte Algorithmus-Suiten beinhalten diese beiden neuen Algorithmus-Suiten eine Verschlüsselung mit AES - GCM, einen 256-Bit-Verschlüsselungsschlüssel und eine HMAC basierte extract-and-expand Schlüsselableitungsfunktion (). HKDF

Die standardmäßige Algorithmussuite, die für die Verschlüsselung verwendet wird, ändert sich jedoch nicht. Diese Algorithmus-Suiten wurden zu Version 1.7 hinzugefügt. x, um Ihre Anwendung für die Verwendung in den Versionen 2.0 vorzubereiten. x und später.

CMMÄnderungen an der Implementierung

Version 1.7. x führt Änderungen an der Standardoberfläche des Cryptographic Materials Manager (CMM) ein, um Key Commitment zu unterstützen. Diese Änderung wirkt sich nur auf Sie aus, wenn Sie eine benutzerdefinierte CMM Version geschrieben haben. Einzelheiten finden Sie in der API Dokumentation oder im GitHub Repository für Ihre [Programmiersprache](#).

Version 2.0. x

Ausführung 2.0. x unterstützt die neuen Sicherheitsfunktionen AWS Encryption SDK, die in der angeboten werden, einschließlich spezifizierter Wrapping Keys und Key Commitment. Zur Unterstützung dieser Funktionen, Version 2.0. x enthält wichtige Änderungen für frühere Versionen von AWS Encryption SDK. Sie können sich auf diese Änderungen vorbereiten, indem Sie Version 1.7 bereitstellen. x. Ausführung 2.0. x enthält alle neuen Funktionen, die in Version 1.7 eingeführt wurden. x mit den folgenden Ergänzungen und Änderungen.

Note

Version 2. x. x von AWS-Verschlüsselungs-SDK for Python AWS-Verschlüsselungs-SDK for JavaScript, und die AWS Verschlüsselung CLI befinden sich in der [end-of-supportPhase](#). Informationen zur [Unterstützung und Wartung](#) dieser AWS Encryption SDK Version in Ihrer bevorzugten Programmiersprache finden Sie in der SUPPORT_POLICY.rst Datei im zugehörigen [GitHubRepository](#).

AWS KMS Hauptschlüsselanbieter

Die ursprünglichen Konstruktoren des AWS KMS Hauptschlüsselanbieters, die in Version 1.7 veraltet waren, wurden in Version 2.0 entfernt. Sie müssen AWS KMS Master-Key-Anbieter explizit im [strikten Modus oder im Discovery-Modus](#) erstellen.

Verschlüsseln und entschlüsseln Sie Chiffretext mit Schlüsselzuweisung

Version 2.0. x [kann Chiffretext mit oder ohne Schlüsselbindung ver- und entschlüsseln](#). Sein Verhalten wird durch die Richtlinieneinstellung „Commitment“ bestimmt. Standardmäßig verschlüsselt es immer mit Key Commitment und entschlüsselt nur Chiffretext, der mit Key Commitment verschlüsselt wurde. Sofern Sie die Verpflichtungsrichtlinie nicht ändern, AWS Encryption SDK werden Chiffretexte, die mit einer früheren Version von, einschließlich Version 1.7, verschlüsselt wurden, nicht entschlüsselt. AWS Encryption SDKx.

Important

Standardmäßig Version 2.0. x entschlüsselt keinen Chiffretext, der ohne Schlüsselbindung verschlüsselt wurde. Wenn Ihre Anwendung möglicherweise auf einen Chiffretext stößt, der ohne Schlüsselzuweisung verschlüsselt wurde, legen Sie einen Wert für die Commitment-Richtlinie mit fest. `AllowDecrypt`

In Version 2.0. x, die Richtlinieneinstellung für Verpflichtungen hat drei gültige Werte:

- `ForbidEncryptAllowDecrypt`— Sie AWS Encryption SDK können nicht mit dem Schlüssel verschlüsseln. Es kann Chiffretexte entschlüsseln, die mit oder ohne Schlüsselbindung verschlüsselt wurden.
- `RequireEncryptAllowDecrypt`— Sie AWS Encryption SDK müssen mit Schlüsselbindung verschlüsseln. Es kann Chiffretexte entschlüsseln, die mit oder ohne Schlüsselbindung verschlüsselt wurden.
- `RequireEncryptRequireDecrypt(Standard)` — Sie AWS Encryption SDK müssen mit Schlüsselzuweisung verschlüsseln. Es entschlüsselt nur Chiffretexte mit Schlüsselbindung.

Wenn Sie von einer früheren Version von auf Version 2.0 migrieren. AWS Encryption SDK x, setzen Sie die Commitment-Richtlinie auf einen Wert, der sicherstellt, dass Sie alle vorhandenen Chiffretexte entschlüsseln können, auf die Ihre Anwendung stoßen könnte. Sie werden diese Einstellung wahrscheinlich im Laufe der Zeit anpassen.

Version 2.2. x

Integriert die Unterstützung für digitale Signaturen und die Beschränkung verschlüsselter Datenschlüssel.

Note

Version 2. x. x von AWS-Verschlüsselungs-SDK for Python AWS-Verschlüsselungs-SDK for JavaScript, und die AWS Verschlüsselung CLI befinden sich in der [end-of-supportPhase](#). Informationen zur [Unterstützung und Wartung](#) dieser AWS Encryption SDK Version in Ihrer bevorzugten Programmiersprache finden Sie in der SUPPORT_POLICY.rst Datei im zugehörigen [GitHubRepository](#).

Digitale Signaturen

Um den Umgang mit [digitalen Signaturen](#) beim Entschlüsseln zu verbessern, AWS Encryption SDK umfasst das die folgenden Funktionen:

- Nicht-Streaming-Modus — gibt Klartext erst zurück, nachdem alle Eingaben verarbeitet wurden, einschließlich der Überprüfung der digitalen Signatur, falls vorhanden. Diese Funktion verhindert, dass Sie vor der Überprüfung der digitalen Signatur Klartext verwenden. Verwenden Sie diese Funktion immer dann, wenn Sie mit digitalen Signaturen verschlüsselte Daten entschlüsseln (die standardmäßige Algorithmussuite). Da die AWS Verschlüsselung beispielsweise Daten CLI immer im Streaming-Modus verarbeitet, sollten Sie den `--buffer` Parameter verwenden, wenn Sie Chiffretext mit digitalen Signaturen entschlüsseln.
- Entschlüsselungsmodus „Nur unsigniert“ — mit dieser Funktion wird nur unsignierter Chiffretext entschlüsselt. Wenn bei der Entschlüsselung eine digitale Signatur im Chiffretext gefunden wird, schlägt der Vorgang fehl. Verwenden Sie diese Funktion, um zu verhindern, dass unbeabsichtigt Klartext aus signierten Nachrichten verarbeitet wird, bevor die Signatur überprüft wird.

Beschränkung verschlüsselter Datenschlüssel

Sie können [die Anzahl der verschlüsselten Datenschlüssel in einer verschlüsselten Nachricht einschränken](#). Mit dieser Funktion können Sie beim Verschlüsseln einen falsch konfigurierten Hauptschlüsselanbieter oder einen falsch konfigurierten Schlüsselbund oder beim Entschlüsseln einen böartigen Chiffretext erkennen.

Sie sollten verschlüsselte Datenschlüssel einschränken, wenn Sie Nachrichten aus einer nicht vertrauenswürdigen Quelle entschlüsseln. Dadurch werden unnötige, teure und potenziell erschöpfende Zugriffe auf Ihre Schlüsselinfrastruktur verhindert.

Version 2.3. x

Fügt Unterstützung für Schlüssel AWS KMS mit mehreren Regionen hinzu. Details hierzu finden Sie unter [Verwenden Sie mehrere Regionen AWS KMS keys](#).

Note

Die AWS Verschlüsselung CLI unterstützt ab Version 3.0 Schlüssel für mehrere Regionen. x. Ausführung 2. x. x von AWS-Verschlüsselungs-SDK for Python AWS-Verschlüsselungs-SDK for JavaScript, und die AWS Verschlüsselung CLI befinden sich in der [end-of-supportPhase](#). Informationen zur [Unterstützung und Wartung](#) dieser AWS Encryption SDK Version in Ihrer bevorzugten Programmiersprache finden Sie in der SUPPORT_POLICY.rst Datei im zugehörigen [GitHubRepository](#).

Migrieren Sie Ihre AWS Encryption SDK

Die AWS Encryption SDK unterstützt mehrere interoperable [Implementierungen von Programmiersprachen](#), von denen jedes in einem Open-Source-Repository auf GitHub. Als [Bewährte Methode](#) empfehlen wir, die neueste Version des AWS Encryption SDK für jede Sprache.

Sie können sicher von Version 2.0 aktualisieren. X-x oder später von AWS Encryption SDK auf die neueste Version. Allerdings 2.0. X-x Version des AWS Encryption SDK führt wichtige neue Sicherheitsfunktionen ein, von denen einige Änderungen durchbrechen. Um ein Upgrade von Versionen vor 1.7 durchzuführen. X-x auf Version 2.0. X-x und später müssen Sie zuerst auf die neueste Version 1 aktualisieren. X-x Ausführung. Die Themen in diesem Abschnitt sollen Ihnen helfen, die Änderungen zu verstehen, die richtige Version für Ihre Anwendung auszuwählen und sicher und erfolgreich auf die neuesten Versionen der AWS Encryption SDK.

Für Informationen über wichtige Versionen des AWS Encryption SDK, siehe [Versionen der AWS Encryption SDK](#).

Important

Führen Sie kein direktes Upgrade von einer Version vor 1.7 durch. X-x auf Version 2.0. X-x oder später, ohne vorher auf die neueste Version zu aktualisieren 1. X-x Ausführung. Wenn Sie direkt auf Version 2.0 aktualisieren. X-x oder später und alle neuen Funktionen sofort aktivieren, AWS Encryption SDK wird nicht in der Lage sein, Chiffretext zu entschlüsseln, der unter älteren Versionen der AWS Encryption SDK.

Note

Die früheste Version des AWS Encryption SDK für .NET ist Version 3.0. X-x. Alle Versionen der AWS Encryption SDK für .NET unterstützen Sie die in 2.0 eingeführten Best Practices für Sicherheit. X-x der AWS Encryption SDK. Sie können sicher auf die neueste Version aktualisieren, ohne Code- oder Datenänderungen vorzunehmen.

AWS Verschlüsselungs-CLI: Verwenden Sie beim Lesen dieses Migrationsleitfadens 1.7. X-x Anleitungen zur Migration für AWS Verschlüsselung CLI 1.8. X-x und benutze 2.0. X-x Anleitungen zur Migration für AWS Verschlüsselung CLI 2.1. X-x. Details hierzu finden Sie unter [Versionen des AWS Verschlüsselung CLI](#).

Neue Sicherheitsfunktionen wurden ursprünglich in veröffentlicht AWS Encryption CLI Version 1.7.X-x und 2.0X-x. Allerdings AWS Verschlüsselung CLI Version 1.8.X-x ersetzt Version 1.7.X-x und AWS Verschlüsselung CLI 2.1.X-x ersetzt 2.0.X-x. Details finden Sie unter [Beratung zur Sicherheit](#) in der [aws-encryption-sdk-cli](#) Repository auf GitHub.

Neuer Benutzer

Wenn für Sie neu ist AWS Encryption SDK, installieren Sie die neueste Version der AWS Encryption SDK für Ihre Programmiersprache. Die Standardwerte aktivieren alle Sicherheitsfunktionen der AWS Encryption SDK, einschließlich Verschlüsselung mit Signierung, Schlüsselableitung und [zentrale Verpflichtung](#). der AWS Encryption SDK

Aktuelle Nutzer

Wir empfehlen Ihnen, Ihre aktuelle Version so schnell wie möglich auf die neueste verfügbare Version zu aktualisieren. Alles 1.X-x Versionen des AWS Encryption SDK sind in der [end-of-support Phase](#), ebenso wie spätere Versionen in einigen Programmiersprachen. Einzelheiten über den Support- und Wartungsstatus der AWS Encryption SDK in Ihrer Programmiersprache finden Sie unter [Support und Wartung](#).

AWS Encryption SDK Versionen 2.0.X-x und später neue Sicherheitsfunktionen zum Schutz Ihrer Daten bereitstellen. Allerdings AWS Encryption SDK Version 2.0.X-x beinhaltet bahnbrechende Änderungen, die nicht abwärtskompatibel sind. Um einen sicheren Übergang zu gewährleisten, migrieren Sie zunächst von Ihrer aktuellen Version auf die neueste 1.X-x in Ihrer Programmiersprache. Wenn dein spätestes 1.X-x Version ist vollständig bereitgestellt und funktioniert erfolgreich. Sie können sicher auf die Versionen 2.0 migrieren.X-x und höher. Das [zweistufige Prozess](#) ist besonders für verteilte Anwendungen von entscheidender Bedeutung.

Weitere Informationen zu finden Sie unter AWS Encryption SDK Sicherheitsfunktionen, die diesen Änderungen zugrunde liegen, siehe [Verbesserte clientseitige Verschlüsselung: explizit KeyIds und wichtige Verpflichtung](#) in der AWS Blog zur Sicherheit.

Auf der Suche nach Hilfe bei der Verwendung des AWS-Verschlüsselungs-SDK for Java mit dem AWS SDK for Java 2.x? Siehe [Voraussetzungen](#).

Themen

- [So migrieren und implementieren Sie das AWS Encryption SDK](#)

- [Aktualisierung der AWS KMS Hauptschlüsselanbieter](#)
- [Wird aktualisiert AWS KMS Schlüsselringe](#)
- [Festlegung Ihrer Verpflichtungspolitik](#)
- [Problembehandlung bei der Migration auf die neuesten Versionen](#)

So migrieren und implementieren Sie das AWS Encryption SDK

Bei der Migration von einem AWS Encryption SDK Version vor 1.7.X- auf Version 2.0.X- oder später müssen Sie sicher zur Verschlüsselung übergehen mit [zentrale Verpflichtung](#). Andernfalls stößt Ihre Anwendung auf Chiffretexte, die sie nicht entschlüsseln kann. Wenn Sie verwenden AWS KMS Masterschlüssel-Provider müssen Sie auf neue Konstruktoren aktualisieren, die Hauptschlüsselanbieter im strikten Modus oder im Ermittlungsmodus erstellen.

Note

Dieses Thema richtet sich an Benutzer, die von früheren Versionen der AWS Encryption SDK auf Version 2.0.X- oder später. Wenn Sie neu in der AWS Encryption SDK können Sie sofort mit der Verwendung der neuesten verfügbaren Version mit den Standardeinstellungen beginnen.

Um eine kritische Situation zu vermeiden, in der Sie verschlüsselten Text, den Sie lesen müssen, nicht entschlüsseln können, empfehlen wir Ihnen, die Migration und Bereitstellung in mehreren unterschiedlichen Phasen durchzuführen. Stellen Sie sicher, dass jede Phase abgeschlossen und vollständig bereitgestellt ist, bevor Sie mit der nächsten Stufe beginnen. Dies ist besonders wichtig für verteilte Anwendungen mit mehreren Hosts.

Phase 1: Aktualisieren Sie Ihre Anwendung auf den neuesten Stand 1.X-x Ausführung

Auf die neueste -Version aktualisieren 1.X-x Version für Ihre Programmiersprache. Testen Sie sorgfältig, stellen Sie Ihre Änderungen bereit und stellen Sie sicher, dass das Update an alle Zielhosts weitergegeben wurde, bevor Sie Phase 2 starten.

⚠ Important

Stellen Sie sicher, dass Ihr neuestes 1.X-xVersion ist Version 1.7.X-xoder später derAWS Encryption SDK.

Die neueste 1.X-xVersionen desAWS Encryption SDKsind abwärtskompatibel mit älteren Versionen derAWS Encryption SDKund vorwärtskompatibel mit den Versionen 2.0.X-xund höher Sie enthalten die neuen Funktionen, die in Version 2.0 verfügbar sind.X-x, enthalten aber sichere Standardwerte, die für diese Migration entwickelt wurden. Sie ermöglichen es Ihnen, IhrAWS KMSMaster-Key-Anbieter, falls erforderlich, und um sie vollständig mit Algorithmus-Suiten bereitzustellen, die Chiffretext mit Schlüsselbindung entschlüsseln können.

- Ersetzen Sie veraltete Elemente, einschließlich Konstruktoren für LegacyAWS KMSMaster-Key-Anbieter. In[Python](#), stellen Sie sicher, dass Sie Verfallswarnungen aktivieren. Codeelemente, die in der letzten Version veraltet sind 1.X-xVersionen wurden aus den Versionen 2.0 entfernt.X-xund höher
- Legen Sie Ihre Verpflichtungsrichtlinie ausdrücklich aufForbidEncryptAllowDecrypt. Obwohl dies der einzig gültige Wert in der letzten 1 ist.X-x-Versionen ist diese Einstellung erforderlich, wenn Sie die in dieser Version eingeführten APIs verwenden. Es verhindert, dass Ihre Anwendung bei der Migration auf Version 2.0 verschlüsselten Chiffretext ohne Schlüsselbindung ablehnt.X-xund höher Details hierzu finden Sie unter [the section called “Festlegung Ihrer Verpflichtungspolitik”](#).
- Wenn SieAWS KMSMaster Key-Anbieter müssen Sie Ihre alten Master-Key-Anbieter auf Master Key-Anbieter aktualisieren, dieStrikter ModusundDiscovery Modus. Dieses Update ist erforderlich fürAWS-Verschlüsselungs-SDK for Java,AWS-Verschlüsselungs-SDK for Python, und derAWSVerschlüsselungs-CLI. Wenn Sie Master Key Provider im Discovery-Modus verwenden, empfehlen wir Ihnen, den Discovery-Filter zu implementieren, der die verwendeten Wrapping-Schlüssel auf bestimmte beschränktAWS-Konten. Dieses Update ist optional, aber es ist ein[Bewährte Methode](#)die wir empfehlen. Details hierzu finden Sie unter [Aktualisierung der AWS KMS Hauptschlüsselanbieter](#).
- Wenn Sie[AWS KMSDiscovery Schlüsselringe](#)empfehlen wir, dass Sie einen Discovery-Filter einbinden, der die bei der Entschlüsselung verwendeten Wrapping-Schlüssel auf bestimmte beschränktAWS-Konten. Dieses Update ist optional, aber es ist ein[Bewährte Methode](#)die wir empfehlen. Details hierzu finden Sie unter [Wird aktualisiertAWS KMSSchlüsselringe](#).

Phase 2: Aktualisieren Sie Ihre Anwendung auf die neueste Version

Nach der Bereitstellung der neuesten Version 1.X-xVersion erfolgreich auf alle Hosts, Sie können ein Upgrade auf Version 2.0 durchführen. X-xund höher Version 2.0.X-xbeinhaltet wichtige Änderungen für alle früheren Versionen desAWS Encryption SDK. Wenn Sie jedoch die in Phase 1 empfohlenen Codeänderungen vornehmen, können Sie Fehler bei der Migration auf die neueste Version vermeiden.

Stellen Sie vor dem Update auf die neueste Version sicher, dass Ihre Verpflichtungsrichtlinie einheitlich aufForbidEncryptAllowDecrypt. Abhängig von Ihrer Datenkonfiguration können Sie dann in Ihrem eigenen Tempo migrierenRequireEncryptAllowDecryptund dann zur StandardeinstellungRequireEncryptRequireDecrypt. Wir empfehlen eine Reihe von Übergangsschritten wie das folgende Muster.

1. Begins withCommiteingestellt aufForbidEncryptAllowDecrypt. DieAWS Encryption SDKkann Nachrichten mit Schlüsselbindung entschlüsseln, aber es wird noch nicht mit Schlüsselbindung verschlüsselt.
2. Wenn Sie bereit sind, aktualisieren Sie Ihre Commit Policy aufRequireEncryptAllowDecrypt. DieAWS Encryption SDKbeginnt Ihre Daten zu verschlüsselnzentrale Verpflichtung. Es kann Chiffretext mit und ohne Schlüsselbindung entschlüsseln.

Bevor Sie Ihre Verpflichtungsrichtlinie auf aktualisierenRequireEncryptAllowDecrypt, vergewissern Sie sich, dass Ihre neueste 1.X-xVersion wird auf allen Hosts bereitgestellt, einschließlich Hosts aller Anwendungen, die den von Ihnen erstellten Chiffretext entschlüsseln. Versionen desAWS Encryption SDKvor Version 1.7.X-xkann Nachrichten, die mit Schlüsselbindung verschlüsselt wurden, nicht entschlüsseln.

Dies ist auch ein guter Zeitpunkt, um Ihrer Anwendung Metriken hinzuzufügen, um zu messen, ob Sie Chiffretext noch ohne wichtige Verpflichtungen verarbeiten. Auf diese Weise können Sie feststellen, wann es sicher ist, Ihre Einstellung für Verpflichtungsrichtlinien aufRequireEncryptRequireDecrypt. Für einige Anwendungen, z. B. solche, die Nachrichten in einer Amazon SQS SQS-Warteschlange verschlüsseln, kann dies bedeuten, dass lange genug gewartet wird, bis der gesamte unter alten Versionen verschlüsselte Chiffretext erneut verschlüsselt oder gelöscht wurde. Für andere Anwendungen, z. B. verschlüsselte S3-Objekte, müssen Sie möglicherweise alle Objekte herunterladen, neu verschlüsseln und erneut hochladen.

3. Wenn Sie sicher sind, dass Sie keine Nachrichten ohne Schlüsselbindung verschlüsselt haben, können Sie Ihre Verpflichtungsrichtlinie aufRequireEncryptRequireDecrypt. Dieser Wert stellt sicher, dass Ihre Daten immer mit Schlüsselbindung ver- und entschlüsselt werden. Diese

Einstellung ist die Standardeinstellung, daher müssen Sie sie nicht explizit festlegen, aber wir empfehlen sie. Eine explizite Einstellung wird [Unterstützt das Debuggings](#) und alle möglichen Rollbacks, die erforderlich sein könnten, wenn Ihre Anwendung auf verschlüsselten Chiffretext stößt, der ohne Schlüsselbindung verschlüsselt wurde.

Aktualisierung der AWS KMS Hauptschlüsselanbieter

Um auf die neueste Version zu migrieren 1. x-Version von AWS Encryption SDK und dann auf Version 2.0. x oder höher, Sie müssen ältere AWS KMS Hauptschlüsselanbieter durch Hauptschlüsselanbieter ersetzen, die explizit im [strikten Modus oder Discovery-Modus](#) erstellt wurden. Ältere Hauptschlüsselanbieter sind in Version 1.7 veraltet. x und wurde in Version 2.0 entfernt. x. Diese Änderung ist für Anwendungen und Skripts erforderlich [AWS-Verschlüsselungs-SDK for Java](#), die die [AWS-Verschlüsselungs-SDK for Python](#), und die [AWS Verschlüsselung verwenden](#)CLI. Die Beispiele in diesem Abschnitt zeigen Ihnen, wie Sie Ihren Code aktualisieren können.

Note

[Schalten Sie in Python Verfallswarnungen ein.](#) Auf diese Weise können Sie die Teile Ihres Codes identifizieren, die Sie aktualisieren müssen.

Wenn Sie einen AWS KMS Hauptschlüssel (keinen Hauptschlüsselanbieter) verwenden, können Sie diesen Schritt überspringen. AWS KMS Hauptschlüssel sind nicht veraltet oder wurden nicht entfernt. Sie verschlüsseln und entschlüsseln nur mit den von Ihnen angegebenen Wrapping-Schlüsseln.

Die Beispiele in diesem Abschnitt konzentrieren sich auf die Elemente Ihres Codes, die Sie ändern müssen. Ein vollständiges Beispiel für den aktualisierten Code finden Sie im Abschnitt Beispiele des GitHub Repositorys für Ihre [Programmiersprache](#). Außerdem wird in diesen Beispielen in der Regel ein Schlüssel ARNs zur Darstellung verwendet AWS KMS keys. Wenn Sie einen Hauptschlüsselanbieter für die Verschlüsselung erstellen, können Sie einen beliebigen gültigen AWS KMS [Schlüsselbezeichner](#) verwenden, um einen AWS KMS key darzustellen. Wenn Sie einen Hauptschlüsselanbieter für die Entschlüsselung erstellen, müssen Sie einen Schlüssel verwenden.
ARN

Erfahren Sie mehr über Migration

Informationen zur Einrichtung Ihrer Verpflichtungsrichtlinie finden Sie für alle AWS Encryption SDK Benutzer unter [the section called “Festlegung Ihrer Verpflichtungspolitik”](#).

Für AWS-Verschlüsselungs-SDK for JavaScript Benutzer AWS-Verschlüsselungs-SDK for C und Benutzer finden Sie weitere Informationen zu einem optionalen Update für Schlüsselanhänger in [Wird aktualisiert AWS KMSSchlüsselringe](#).

Themen

- [Umstellung auf den strikten Modus](#)
- [In den Discovery-Modus migrieren](#)

Umstellung auf den strikten Modus

Nach dem Update auf die neueste Version 1. x-Version von AWS Encryption SDK, ersetzen Sie Ihre alten Hauptschlüsselanbieter durch Hauptschlüsselanbieter im strikten Modus. Im strikten Modus müssen Sie die Wrapping-Schlüssel angeben, die beim Verschlüsseln und Entschlüsseln verwendet werden sollen. Der AWS Encryption SDK verwendet nur die von Ihnen angegebenen Wrapping-Schlüssel. Veraltete Hauptschlüsselanbieter können Daten mit jedem entschlüsseln AWS KMS key , der einen Datenschlüssel verschlüsselt hat, auch AWS KMS keys in verschiedenen AWS-Konten Regionen.

Master-Key-Anbieter im strikten Modus wurden in Version 1.7 eingeführt. AWS Encryption SDK x. Sie ersetzen ältere Master-Key-Anbieter, die in 1.7 veraltet sind. x und in 2.0 entfernt. x. Die Verwendung von Master-Key-Anbietern im strikten Modus ist eine AWS Encryption SDK [bewährte Methode](#).

Der folgende Code erstellt einen Hauptschlüsselanbieter im strikten Modus, den Sie zum Verschlüsseln und Entschlüsseln verwenden können.

Java

Dieses Beispiel stellt Code in einer Anwendung dar, die die Version 1.6.2 oder früher von verwendet. AWS-Verschlüsselungs-SDK for Java

In diesem Code wird die `KmsMasterKeyProvider.builder()` Methode verwendet, um einen AWS KMS Hauptschlüsselanbieter zu instanziierten, der einen AWS KMS key als Umschließungsschlüssel verwendet.

```
// Create a master key provider
```

```
// Replace the example key ARN with a valid one
String awsKmsKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

KmsMasterKeyProvider masterKeyProvider = KmsMasterKeyProvider.builder()
    .withKeysForEncryption(awsKmsKey)
    .build();
```

Dieses Beispiel stellt Code in einer Anwendung dar, die Version 1.7 verwendet. x oder höher von AWS-Verschlüsselungs-SDK for Java . Ein vollständiges Beispiel finden Sie unter [BasicEncryptionExample.java](#).

Die im vorherigen Beispiel verwendeten `Builder.withKeysForEncryption()` Methoden `Builder.build()` und sind in Version 1.7 veraltet. x und wurden aus Version 2.0 entfernt. x.

Um auf einen Master-Key-Anbieter im strikten Modus zu aktualisieren, ersetzt dieser Code Aufrufe veralteter Methoden durch einen Aufruf der neuen `Builder.buildStrict()` Methode. In diesem Beispiel wird eine AWS KMS key als Schlüssel für den Zeilenumbruch angegeben, aber die `Builder.buildStrict()` Methode kann auch eine Liste mit mehreren verwenden. AWS KMS keys

```
// Create a master key provider in strict mode
// Replace the example key ARN with a valid one from your AWS-Konto.
String awsKmsKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

KmsMasterKeyProvider masterKeyProvider = KmsMasterKeyProvider.builder()
    .buildStrict(awsKmsKey);
```

Python

Dieses Beispiel stellt Code in einer Anwendung dar, die Version 1.4.1 von verwendet. AWS-Verschlüsselungs-SDK for Python Dieser Code verwendet `KMSMasterKeyProvider`, was in Version 1.7 veraltet ist. x und wurde aus Version 2.0 entfernt. x. Bei der Entschlüsselung wird jeder verwendet, der einen Datenschlüssel verschlüsselt hat AWS KMS key , unabhängig von dem, was AWS KMS keys Sie angeben.

Beachten Sie, dass dies nicht veraltet `KMSMasterKey` ist oder entfernt wurde. Beim Verschlüsseln und Entschlüsseln werden nur die von Ihnen angegebenen verwendet. AWS KMS key

```
# Create a master key provider
# Replace the example key ARN with a valid one
key_1 = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
key_2 = "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321"

aws_kms_master_key_provider = KMSMasterKeyProvider(
    key_ids=[key_1, key_2]
)
```

Dieses Beispiel stellt Code in einer Anwendung dar, die Version 1.7 verwendet. x von AWS-Verschlüsselungs-SDK for Python. Ein vollständiges Beispiel finden Sie unter [basic_encryption.py](#).

Um auf einen Master-Key-Anbieter im strikten Modus zu aktualisieren, ersetzt dieser Code den Aufruf von `KMSMasterKeyProvider()` durch einen Aufruf von `StrictAwsKmsMasterKeyProvider()`.

```
# Create a master key provider in strict mode
# Replace the example key ARNs with valid values from your AWS-Konto
key_1 = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
key_2 = "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321"

aws_kms_master_key_provider = StrictAwsKmsMasterKeyProvider(
    key_ids=[key_1, key_2]
)
```

AWS Encryption CLI

Dieses Beispiel zeigt, wie mit der AWS CLI Verschlüsselungsversion 1.1.7 oder früher ver- und entschlüsselt wird.

In Version 1.1.7 und früheren Versionen geben Sie beim Verschlüsseln einen oder mehrere Hauptschlüssel (oder Wrapping-Schlüssel) an, z. B. AWS KMS key. Beim Entschlüsseln können Sie keine Umschließungsschlüssel angeben, es sei denn, Sie verwenden einen benutzerdefinierten Hauptschlüsselanbieter. Bei der AWS Verschlüsselung CLI kann ein beliebiger Umschließungsschlüssel verwendet werden, der einen Datenschlüssel verschlüsselt hat.

```

\\ Replace the example key ARN with a valid one
$ keyArn=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

\\ Encrypt your plaintext data
$ aws-encryption-cli --encrypt \
    --input hello.txt \
    --master-keys key=$keyArn \
    --metadata-output ~/metadata \
    --encryption-context purpose=test \
    --output .

\\ Decrypt your ciphertext
$ aws-encryption-cli --decrypt \
    --input hello.txt.encrypted \
    --encryption-context purpose=test \
    --metadata-output ~/metadata \
    --output .

```

Dieses Beispiel zeigt, wie mit der AWS CLI Verschlüsselungsversion 1.7 verschlüsselt und entschlüsselt wird. x oder höher. Vollständige Beispiele finden Sie unter [Beispiele für die AWS CLI für Verschlüsselung](#).

Der `--master-keys` Parameter ist in Version 1.7 veraltet. x und wurde in Version 2.0 entfernt. x. Er wurde durch den `--wrapping-keys` Parameter by ersetzt, der für Befehle zum Verschlüsseln und Entschlüsseln erforderlich ist. Dieser Parameter unterstützt den Strict-Modus und den Discovery-Modus. Der strikte Modus ist eine AWS Encryption SDK bewährte Methode, mit der sichergestellt wird, dass Sie den gewünschten Umbruchschlüssel verwenden.

Um auf den strikten Modus umzusteigen, verwenden Sie das Schlüsselattribut des `--wrapping-keys` Parameters, um beim Verschlüsseln und Entschlüsseln einen Umschließungsschlüssel anzugeben.

```

\\ Replace the example key ARN with a valid value
$ keyArn=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

\\ Encrypt your plaintext data
$ aws-encryption-cli --encrypt \
    --input hello.txt \
    --wrapping-keys key=$keyArn \
    --metadata-output ~/metadata \
    --encryption-context purpose=test \
    --output .

```

```
\\ Decrypt your ciphertext
$ aws-encryption-cli --decrypt \
    --input hello.txt.encrypted \
    --wrapping-keys key=$keyArn \
    --encryption-context purpose=test \
    --metadata-output ~/metadata \
    --output .
```

In den Discovery-Modus migrieren

Ab Version 1.7. x, [es hat sich bewährt AWS Encryption SDK, den strikten Modus für AWS KMS Hauptschlüsselanbieter zu verwenden, d. h., beim Verschlüsseln und Entschlüsseln die Schlüssel zu verpacken](#). Sie müssen beim Verschlüsseln immer Wrapping-Schlüssel angeben. Es gibt jedoch Situationen, in denen die Angabe ARNs des Schlüssels AWS KMS keys für die Entschlüsselung nicht praktikabel ist. Wenn Sie beispielsweise AWS KMS keys beim Verschlüsseln Aliase zur Identifizierung verwenden, verlieren Sie den Vorteil von Aliasen, wenn Sie beim Entschlüsseln den Schlüssel angeben müssen. ARNs Da sich Hauptschlüsselanbieter im Erkennungsmodus wie die ursprünglichen Hauptschlüsselanbieter verhalten, können Sie sie außerdem vorübergehend als Teil Ihrer Migrationsstrategie verwenden und später im strikten Modus auf Hauptschlüsselanbieter umsteigen.

In solchen Fällen können Sie Hauptschlüsselanbieter im Erkennungsmodus verwenden. Bei diesen Hauptschlüsselanbietern können Sie keine Schlüssel angeben, sodass Sie sie nicht zum Verschlüsseln verwenden können. Bei der Entschlüsselung können sie jeden beliebigen Umschließungsschlüssel verwenden, mit dem ein Datenschlüssel verschlüsselt wurde. Im Gegensatz zu älteren Hauptschlüsselanbietern, die sich genauso verhalten, werden sie jedoch explizit im Discovery-Modus erstellt. Wenn Sie Master-Key-Anbieter im Discovery-Modus verwenden, können Sie die Anzahl der Wrapping-Schlüssel, die verwendet werden können, auf bestimmte Schlüssel beschränken AWS-Konten. Dieser Erkennungsfiler ist optional, aber es handelt sich dabei um eine bewährte Methode, die wir empfehlen. Informationen zu AWS Partitionen und Konten finden Sie unter [Amazon Resource Names](#) in der Allgemeine AWS-Referenz.

In den folgenden Beispielen werden ein AWS KMS Master-Key-Provider im Strict-Modus für die Verschlüsselung und ein AWS KMS Master-Key-Provider im Discovery-Modus für die Entschlüsselung erstellt. Der Hauptschlüsselanbieter im Discovery-Modus verwendet einen Discovery-Filter, um die für die Entschlüsselung verwendeten Wrapping-Schlüssel auf die aws Partition und auf ein bestimmtes Beispiel zu beschränken. AWS-Konten Obwohl der Kontofilter in

diesem sehr einfachen Beispiel nicht erforderlich ist, handelt es sich um eine bewährte Methode, die sehr nützlich ist, wenn eine Anwendung Daten verschlüsselt und eine andere Anwendung die Daten entschlüsselt.

Java

Dieses Beispiel stellt Code in einer Anwendung dar, die Version 1.7 verwendet. x oder höher von AWS-Verschlüsselungs-SDK for Java. Ein vollständiges Beispiel finden Sie unter [DiscoveryDecryptionExample.java](#).

In diesem Beispiel wird die Methode verwendet, um einen Hauptschlüsselanbieter im strikten Modus für die Verschlüsselung zu instanziiieren. `Builder.buildStrict()` Um einen Hauptschlüsselanbieter im Discovery-Modus für die Entschlüsselung zu instanziiieren, wird die Methode verwendet. `Builder.buildDiscovery()` Die `Builder.buildDiscovery()` Methode verwendet eine, `DiscoveryFilter` die den Wert AWS Encryption SDK auf AWS KMS keys in der angegebenen AWS Partition und den angegebenen Konten begrenzt.

```
// Create a master key provider in strict mode for encrypting
// Replace the example alias ARN with a valid one from your AWS-Konto.
String awsKmsKey = "arn:aws:kms:us-west-2:111122223333:alias/ExampleAlias";

KmsMasterKeyProvider encryptingKeyProvider = KmsMasterKeyProvider.builder()
    .buildStrict(awsKmsKey);

// Create a master key provider in discovery mode for decrypting
// Replace the example account IDs with valid values.
DiscoveryFilter accounts = new DiscoveryFilter("aws", Arrays.asList("111122223333",
    "444455556666"));

KmsMasterKeyProvider decryptingKeyProvider = KmsMasterKeyProvider.builder()
    .buildDiscovery(accounts);
```

Python

Dieses Beispiel stellt Code in einer Anwendung dar, die Version 1.7 verwendet. x oder höher von AWS-Verschlüsselungs-SDK for Python . Ein vollständiges Beispiel finden Sie unter [discovery_kms_provider.py](#).

In diesem Beispiel wird `StrictAwsKmsMasterKeyProvider`, um einen Hauptschlüsselanbieter im strikten Modus für die Verschlüsselung zu erstellen. Um einen Hauptschlüsselanbieter im Discovery-Modus für die Entschlüsselung zu erstellen, wird

`DiscoveryAwsKmsMasterKeyProvider` verwendet, `DiscoveryFilter` das den Wert `AWS Encryption SDK` auf AWS KMS keys in der angegebenen AWS Partition und den angegebenen Konten begrenzt.

```
# Create a master key provider in strict mode
# Replace the example key ARN and alias ARNs with valid values from your AWS-Konto.
key_1 = "arn:aws:kms:us-west-2:111122223333:alias/ExampleAlias"
key_2 = "arn:aws:kms:us-west-2:444455556666:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d"

aws_kms_master_key_provider = StrictAwsKmsMasterKeyProvider(
    key_ids=[key_1, key_2]
)

# Create a master key provider in discovery mode for decrypting
# Replace the example account IDs with valid values
accounts = DiscoveryFilter(
    partition="aws",
    account_ids=["111122223333", "444455556666"]
)
aws_kms_master_key_provider = DiscoveryAwsKmsMasterKeyProvider(
    discovery_filter=accounts
)
```

AWS Encryption CLI

Dieses Beispiel zeigt, wie mit der Verschlüsselungsversion 1.7 verschlüsselt und entschlüsselt wird AWS . CLI x oder höher. Ab Version 1.7. x, der `--wrapping-keys` Parameter ist beim Verschlüsseln und Entschlüsseln erforderlich. Der `--wrapping-keys` Parameter unterstützt den Strict-Modus und den Discovery-Modus. Vollständige Beispiele finden Sie unter [the section called "Beispiele"](#).

Bei der Verschlüsselung wird in diesem Beispiel ein Umschließungsschlüssel angegeben, der erforderlich ist. Beim Entschlüsseln wird explizit der Erkennungsmodus ausgewählt, indem das `discovery` Attribut des `--wrapping-keys` Parameters mit dem Wert `von verwendet` wird. `true`

Um die Anzahl der Schlüssel, die im Discovery-Modus verwendet werden AWS Encryption SDK können, auf bestimmte Schlüssel zu beschränken AWS-Konten, werden in diesem Beispiel die `discovery-account` Attribute `discovery-partition` und des `--wrapping-keys` Parameters verwendet. Diese optionalen Attribute sind nur gültig, wenn das `discovery` Attribut

auf gesetzt ist `true`. Sie müssen die `discovery-account` Attribute `discovery-partition` und `discovery-account` zusammen verwenden. Keines der Attribute ist für sich allein gültig.

```

\\ Replace the example key ARN with a valid value
$ keyAlias=arn:aws:kms:us-west-2:111122223333:alias/ExampleAlias

\\ Encrypt your plaintext data
$ aws-encryption-cli --encrypt \
    --input hello.txt \
    --wrapping-keys key=$keyAlias \
    --metadata-output ~/metadata \
    --encryption-context purpose=test \
    --output .

\\ Decrypt your ciphertext
\\ Replace the example account IDs with valid values
$ aws-encryption-cli --decrypt \
    --input hello.txt.encrypted \
    --wrapping-keys discovery=true \
    discovery-partition=aws \
    discovery-account=111122223333 \
    discovery-account=444455556666 \
    --encryption-context purpose=test \
    --metadata-output ~/metadata \
    --output .

```

Wird aktualisiert AWS KMSSchlüsselringe

Die AWS KMSSchlüsselringe in der [AWS-Verschlüsselungs-SDK for C](#), das [AWS Encryption SDK für .NET](#), und der [AWS-Verschlüsselungs-SDK for JavaScript](#) Unterstützung [Bewährte Methoden](#) indem Sie beim Ver- und Entschlüsseln Wrapping-Schlüssel angeben können. Wenn Sie eine erstellen [AWS KMS Discovery Schlüsselring](#) tun Sie dies ausdrücklich.

Note

Die früheste Version des AWS Encryption SDK für .NET ist Version 3.0.x. Alle Versionen des AWS Encryption SDK für .NET unterstützen Sie die in 2.0 eingeführten Best Practices für Sicherheit. x der AWS Encryption SDK. Sie können sicher auf die neueste Version aktualisieren, ohne Code- oder Datenänderungen vorzunehmen.

Wenn Sie auf die neueste Version aktualisieren 1.x-Version des AWS Encryption SDK können Sie verwenden eine [Discovery-Filter](#) um die Wrapping-Schlüssel zu begrenzen, die [AWS KMS Discovery Schlüsselring](#) oder [AWS KMS Regional Discovery Schlüsselring](#) wird insbesondere beim Entschlüsseln in diese verwendet AWS-Konten. Das Filtern eines Discovery-Schlüsselrings ist ein AWS Encryption SDK [Bewährte Methode](#).

Die Beispiele in diesem Abschnitt zeigen Ihnen, wie Sie den Discovery-Filter zu einem AWS KMS Schlüsselanhänger für regionale Entdeckungen.

Weitere Informationen über Migration

Für alle AWS Encryption SDK Benutzer, erfahren Sie mehr über das Festlegen Ihrer Verantwortungsrichtlinie [the section called “Festlegung Ihrer Verpflichtungspolitik”](#).

Für AWS-Verschlüsselungs-SDK for Java, AWS-Verschlüsselungs-SDK for Python, und AWS Benutzer von Encryption CLI erfahren mehr über ein erforderliches Update für Master Key-Anbieter unter [the section called “Aktualisierung der AWS KMS Hauptschlüsselanbieter”](#).

Möglicherweise haben Sie Code wie den folgenden in Ihrer Anwendung. Dieses Beispiel erstellt einen AWS KMS regionaler Discovery-Schlüsselring, der nur in der Region US West (Oregon) (us-west-2) -Schlüsselbund verwenden kann. Dieses Beispiel steht für Code in AWS Encryption SDK Versionen vor 1.7.x. Es ist jedoch weiterhin in den Versionen 1.7 gültig.x und später.

C

```
struct aws_cryptosdk_keyring *kms_regional_keyring =
    Aws::Cryptosdk::KmsKeyring::Builder()
        .WithKmsClient(create_kms_client(Aws::Region::US_WEST_2)).BuildDiscovery();
```

JavaScript Browser

```
const clientProvider = getClient(KMS, { credentials })

const discovery = true
const clientProvider = limitRegions(['us-west-2'], getKmsClient)
const keyring = new KmsKeyringBrowser({ clientProvider, discovery })
```

JavaScript Node.js

```
const discovery = true
```

```
const clientProvider = limitRegions(['us-west-2'], getKmsClient)
const keyring = new KmsKeyringNode({ clientProvider, discovery })
```

Beginnend mit Version 1.7.x können Sie einen Discovery-Filter zu jedem hinzufügen AWS KMS Discovery Schlüsselring. Dieser Discovery-Filter begrenzt die AWS KMS keys, die der AWS Encryption SDK für die Entschlüsselung von Personen in der angegebenen Partition und Konten verwendet werden. Bevor Sie diesen Code verwenden, ändern Sie gegebenenfalls die Partition und ersetzen Sie die Beispielkonto-IDs durch gültige.

C

Ein vollständiges Beispiel finden Sie in [kms_discovery.cpp](#).

```
std::shared_ptr<KmsKeyring::DiscoveryFilter> discovery_filter(
    KmsKeyring::DiscoveryFilter::Builder("aws")
        .AddAccount("111122223333")
        .AddAccount("444455556666")
        .Build());

struct aws_cryptosdk_keyring *kms_regional_keyring =
    Aws::Cryptosdk::KmsKeyring::Builder()

        .WithKmsClient(create_kms_client(Aws::Region::US_WEST_2)).BuildDiscovery(discovery_filter))
```

JavaScript Browser

```
const clientProvider = getClient(KMS, { credentials })

const discovery = true
const clientProvider = limitRegions(['us-west-2'], getKmsClient)
const keyring = new KmsKeyringBrowser(clientProvider, {
    discovery,
    discoveryFilter: { accountIDs: ['111122223333', '444455556666'], partition:
    'aws' }
})
```

JavaScript Node.js

Ein vollständiges Beispiel finden Sie in [kms_filtered_discovery.ts](#).

```
const discovery = true
```

```
const clientProvider = limitRegions(['us-west-2'], getKmsClient)
const keyring = new KmsKeyringNode({
  clientProvider,
  discovery,
  discoveryFilter: { accountIDs: ['111122223333', '444455556666'], partition:
    'aws' }
})
```

Festlegung Ihrer Verpflichtungspolitik

[Key Commitment](#) stellt sicher, dass Ihre verschlüsselten Daten immer im gleichen Klartext entschlüsselt werden. Um diese Sicherheitseigenschaft ab Version 1.7 bereitzustellen, verwendet AWS Encryption SDK neue [Algorithmus-Suiten](#) mit hohem Engagement. Um festzustellen, ob Ihre Daten mit Key Commitment verschlüsselt und entschlüsselt werden, verwenden Sie die Konfigurationseinstellung der [Commitment-Richtlinie](#). [Das Verschlüsseln und Entschlüsseln von Daten mit Key Commitment ist eine AWS Encryption SDK bewährte Methode](#).

Die Festlegung einer verbindlichen Richtlinie ist ein wichtiger Bestandteil des zweiten Schritts im Migrationsprozess — der Migration von der letzten Version 1.x Versionen der beiden AWS Encryption SDK Versionen 2.0.x und später. Nachdem Sie Ihre Verpflichtungsrichtlinie festgelegt und geändert haben, sollten Sie Ihre Anwendung gründlich testen, bevor Sie sie in der Produktion einsetzen. Hinweise zur Migration finden Sie unter [So migrieren und implementieren Sie das AWS Encryption SDK](#).

Die Richtlinieneinstellung „Commitment“ hat in den Versionen 2.0 drei gültige Werte, x und später. In der letzten 1.x-Versionen (beginnend mit Version 1.7.x), `ForbidEncryptAllowDecrypt` ist nur gültig.

- `ForbidEncryptAllowDecrypt`— Sie AWS Encryption SDK können nicht mit Schlüsselzusatz verschlüsseln. Es kann Chiffretexte entschlüsseln, die mit oder ohne Schlüsselbindung verschlüsselt wurden.

In der letzten Version 1.x-Versionen, dies ist der einzig gültige Wert. Es stellt sicher, dass Sie erst dann mit Key Commitment verschlüsseln, wenn Sie vollständig bereit sind, mit Key Commitment zu entschlüsseln. Wenn Sie den Wert explizit festlegen, wird verhindert, dass sich Ihre Verpflichtungsrichtlinie automatisch ändert, `require-encrypt-require-decrypt` wenn Sie auf Version 2.0 aktualisieren, x oder später. Stattdessen können Sie [Ihre Verpflichtungspolitik schrittweise migrieren](#).

- `RequireEncryptAllowDecrypt`— Das verschlüsselt AWS Encryption SDK immer mit Schlüsselzusage. Es kann Chiffretexte entschlüsseln, die mit oder ohne Schlüsselbindung verschlüsselt wurden. Dieser Wert wurde in Version 2.0 hinzugefügt. x.
- `RequireEncryptRequireDecrypt`— Der verschlüsselt und entschlüsselt AWS Encryption SDK immer mit Schlüsselbindung. Dieser Wert wurde in Version 2.0 hinzugefügt. x. Dies ist der Standardwert in den Versionen 2.0. x und später.

In der letzten 1. x-Versionen, der einzig gültige Wert der Verpflichtungspolice ist `ForbidEncryptAllowDecrypt`. Nach der Migration auf Version 2.0. x oder später können Sie [Ihre Verpflichtungsrichtlinie schrittweise ändern](#), sobald Sie bereit sind. Aktualisieren Sie Ihre Verpflichtungsrichtlinie `RequireEncryptRequireDecrypt` erst, wenn Sie sicher sind, dass Sie keine Nachrichten ohne Schlüsselbindung verschlüsselt haben.

Diese Beispiele zeigen Ihnen, wie Sie Ihre Verpflichtungsrichtlinie in der neuesten Version festlegen. x-Versionen und in den Versionen 2.0. x und später. Die Technik hängt von Ihrer Programmiersprache ab.

Erfahren Sie mehr über Migration

Informationen zu den AWS erforderlichen Änderungen an den Hauptschlüsselanbietern finden Sie unter [the section called “Aktualisierung der AWS KMS Hauptschlüsselanbieter”](#). AWS-Verschlüsselungs-SDK for Java AWS-Verschlüsselungs-SDK for Python CLI

Informationen zu einem optionalen Update für AWS-Verschlüsselungs-SDK for C Schlüsselanhänger finden Sie unter [Wird aktualisiert AWS KMSSchlüsselringe](#). AWS-Verschlüsselungs-SDK for JavaScript

Wie legen Sie Ihre Verpflichtungsrichtlinie fest

Die Methode, mit der Sie Ihre Verpflichtungspolitik festlegen, unterscheidet sich je nach Sprachimplementierung geringfügig. Diese Beispiele zeigen Ihnen, wie das geht. Bevor Sie Ihre Verpflichtungspolitik ändern, überprüfen Sie den mehrstufigen Ansatz unter [So migrieren und implementieren Sie](#).

C

Ab Version 1.7. x von AWS-Verschlüsselungs-SDK for C, Sie verwenden die `aws_cryptosdk_session_set_commitment_policy` Funktion, um die Verpflichtungsrichtlinie für Ihre Verschlüsselungs- und Entschlüsselungssitzungen festzulegen.

Die von Ihnen festgelegte Commitment-Richtlinie gilt für alle Verschlüsselungs- und Entschlüsselungsvorgänge, die in dieser Sitzung aufgerufen werden.

Die `aws_cryptosdk_session_new_from_cmm` Funktionen `aws_cryptosdk_session_new_from_keyring` und sind in Version 1.7 veraltet. `x` und wurde in Version 2.0 entfernt. `x`. Diese Funktionen werden durch `aws_cryptosdk_session_new_from_cmm_2` Funktionen `aws_cryptosdk_session_new_from_keyring_2` und ersetzt, die eine Sitzung zurückgeben.

Wenn Sie das `aws_cryptosdk_session_new_from_keyring_2` und `aws_cryptosdk_session_new_from_cmm_2` in der letzten Version verwenden 1. `x`-Versionen, Sie müssen die `aws_cryptosdk_session_set_commitment_policy` Funktion mit dem `COMMITMENT_POLICY_FORBID_ENCRYPT_ALLOW_DECRYPT` Commitment-Policy-Wert aufrufen. In den Versionen 2.0. `x` und höher ist der Aufruf dieser Funktion optional und akzeptiert alle gültigen Werte. Die Standard-Commitment-Richtlinie für Versionen 2.0. `x` und höher ist `COMMITMENT_POLICY_REQUIRE_ENCRYPT_REQUIRE_DECRYPT`.

Ein vollständiges Beispiel finden Sie unter [string.cpp](#).

```

/* Load error strings for debugging */
aws_cryptosdk_load_error_strings();

/* Create an AWS KMS keyring */
const char * key_arn = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
struct aws_cryptosdk_keyring *kms_keyring =
    Aws::Cryptosdk::KmsKeyring::Builder().Build(key_arn);

/* Create an encrypt session with a CommitmentPolicy setting */
struct aws_cryptosdk_session *encrypt_session =
    aws_cryptosdk_session_new_from_keyring_2(
        alloc, AWS_CRYPTOSDK_ENCRYPT, kms_keyring);

aws_cryptosdk_keyring_release(kms_keyring);
aws_cryptosdk_session_set_commitment_policy(encrypt_session,
    COMMITMENT_POLICY_FORBID_ENCRYPT_ALLOW_DECRYPT);

...
/* Encrypt your data */

size_t plaintext_consumed_output;
aws_cryptosdk_session_process(encrypt_session,

```

```

        ciphertext_output,
        ciphertext_buf_sz_output,
        ciphertext_len_output,
        plaintext_input,
        plaintext_len_input,
        &plaintext_consumed_output)

...

/* Create a decrypt session with a CommitmentPolicy setting */

struct aws_cryptosdk_keyring *kms_keyring =
    Aws::Cryptosdk::KmsKeyring::Builder().Build(key_arn);
struct aws_cryptosdk_session *decrypt_session =
    *aws_cryptosdk_session_new_from_keyring_2(
        alloc, AWS_CRYPTOSDK_DECRYPT, kms_keyring);
aws_cryptosdk_keyring_release(kms_keyring);
aws_cryptosdk_session_set_commitment_policy(decrypt_session,
        COMMITMENT_POLICY_FORBID_ENCRYPT_ALLOW_DECRYPT);

/* Decrypt your ciphertext */
size_t ciphertext_consumed_output;
aws_cryptosdk_session_process(decrypt_session,
        plaintext_output,
        plaintext_buf_sz_output,
        plaintext_len_output,
        ciphertext_input,
        ciphertext_len_input,
        &ciphertext_consumed_output)

```

C# / .NET

Der `require-encrypt-require-decrypt` Wert ist die Standard-Commitment-Richtlinie in allen Versionen von AWS Encryption SDK for .NET. Sie können es als bewährte Methode explizit festlegen, es ist jedoch nicht erforderlich. Wenn Sie jedoch das AWS Encryption SDK für verwenden. NET Um Chiffretext zu entschlüsseln, der mit einer Implementierung von AWS Encryption SDK Without Key Commitment in einer anderen Sprache verschlüsselt wurde, müssen Sie den Wert der Commitment-Richtlinie auf oder ändern. `REQUIRE_ENCRYPT_ALLOW_DECRYPT` `FORBID_ENCRYPT_ALLOW_DECRYPT` Andernfalls schlagen Versuche, den Chiffretext zu entschlüsseln, fehl.

In der Form. AWS Encryption SDK NET, Sie legen die Verpflichtungsrichtlinie für eine Instanz von fest AWS Encryption SDK. Instanzieren Sie ein `AwsEncryptionSdkConfig` Objekt mit einem

CommitmentPolicy Parameter und verwenden Sie das Konfigurationsobjekt, um die Instanz zu erstellen. AWS Encryption SDK Rufen Sie dann die Decrypt() Methoden Encrypt() und der konfigurierten AWS Encryption SDK Instanz auf.

In diesem Beispiel wird die Commitment-Richtlinie auf festgelegtrequire-encrypt-allow-decrypt.

```
// Instantiate the material providers
var materialProviders =

    AwsCryptographicMaterialProvidersFactory.CreateDefaultAwsCryptographicMaterialProviders();

// Configure the commitment policy on the AWS Encryption SDK instance
var config = new AwsEncryptionSdkConfig
{
    CommitmentPolicy = CommitmentPolicy.REQUIRE_ENCRYPT_ALLOW_DECRYPT
};
var encryptionSdk = AwsEncryptionSdkFactory.CreateAwsEncryptionSdk(config);

string keyArn = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

var encryptionContext = new Dictionary<string, string>()
{
    {"purpose", "test"}encryptionSdk
};

var createKeyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = keyArn
};
var keyring = materialProviders.CreateAwsKmsKeyring(createKeyringInput);

// Encrypt your plaintext data
var encryptInput = new EncryptInput
{
    Plaintext = plaintext,
    Keyring = keyring,
    EncryptionContext = encryptionContext
};
var encryptOutput = encryptionSdk.Encrypt(encryptInput);
```

```
// Decrypt your ciphertext
var decryptInput = new DecryptInput
{
    Ciphertext = ciphertext,
    Keyring = keyring
};
var decryptOutput = encryptionSdk.Decrypt(decryptInput);
```

AWS Encryption CLI

Verwenden Sie den `--commitment-policy` ParameterCLI, um eine Verpflichtungsrichtlinie in der AWS Verschlüsselung festzulegen. Dieser Parameter wurde in Version 1.8 eingeführt. x.

In der letzten 1. X-Version, wenn Sie den `--wrapping-keys` Parameter in einem `--encrypt` `--decrypt` OR-Befehl verwenden, ist ein `--commitment-policy` Parameter mit dem `forbid-encrypt-allow-decrypt` Wert erforderlich. Andernfalls ist der `--commitment-policy` Parameter ungültig.

In den Versionen 2.1. x und höher ist der `--commitment-policy` Parameter optional und hat standardmäßig den `require-encrypt-require-decrypt` Wert, der keinen Chiffretext verschlüsselt oder entschlüsselt, der ohne Schlüsselzuweisung verschlüsselt wurde. Wir empfehlen jedoch, dass Sie die Commitment-Richtlinie explizit in allen Verschlüsseln und Entschlüsseln festlegen, um die Wartung und Fehlerbehebung zu erleichtern.

In diesem Beispiel wird die Commitment-Richtlinie festgelegt. Außerdem wird der `--wrapping-keys` Parameter verwendet, der den `--master-keys` Parameter ab Version 1.8 ersetzt. x. Details hierzu finden Sie unter [the section called “Aktualisierung der AWS KMS Hauptschlüsselanbieter”](#). Vollständige Beispiele finden Sie unter [Beispiele für dieAWSCLI für Verschlüsselung](#).

```
\\ To run this example, replace the fictitious key ARN with a valid value.
$ keyArn=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

\\ Encrypt your plaintext data - no change to algorithm suite used
$ aws-encryption-cli --encrypt \
    --input hello.txt \
    --wrapping-keys key=$keyArn \
    --commitment-policy forbid-encrypt-allow-decrypt \
    --metadata-output ~/metadata \
    --encryption-context purpose=test \
    --output .
```



```

\\ Decrypt your ciphertext - supports key commitment on 1.7 and later
$ aws-encryption-cli --decrypt \
    --input hello.txt.encrypted \
    --wrapping-keys key=$keyArn \
    --commitment-policy forbid-encrypt-allow-decrypt \
    --encryption-context purpose=test \
    --metadata-output ~/metadata \
    --output .

```

Java

Ab Version 1.7. x von AWS-Verschlüsselungs-SDK for Java, Sie legen die Commitment-Richtlinie für Ihre Instanz von `festAwsCrypto`, dem Objekt, das den AWS Encryption SDK Client darstellt. Diese Commitment-Richtlinieneinstellung gilt für alle Verschlüsselungs- und Entschlüsselungsvorgänge, die auf diesem Client aufgerufen werden.

Der `AwsCrypto()` Konstruktor ist in der neuesten Version 1 veraltet. x-Versionen von AWS-Verschlüsselungs-SDK for Java und wurden in Version 2.0 entfernt. x. Es wird durch eine neue Builder Klasse, eine `Builder.withCommitmentPolicy()` Methode und den `CommitmentPolicy` Aufzählungstyp ersetzt.

In der letzten 1. x-Versionen, die Builder Klasse benötigt die `Builder.withCommitmentPolicy()` Methode und das `CommitmentPolicy.ForbidEncryptAllowDecrypt` Argument. Ab Version 2.0. x, die `Builder.withCommitmentPolicy()` Methode ist optional; der Standardwert ist `CommitmentPolicy.RequireEncryptRequireDecrypt`.

Ein vollständiges Beispiel finden Sie unter [SetCommitmentPolicyExample.java](#).

```

// Instantiate the client
final AwsCrypto crypto = AwsCrypto.builder()
    .withCommitmentPolicy(CommitmentPolicy.ForbidEncryptAllowDecrypt)
    .build();

// Create a master key provider in strict mode
String awsKmsKey = "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

KmsMasterKeyProvider masterKeyProvider = KmsMasterKeyProvider.builder()
    .buildStrict(awsKmsKey);

// Encrypt your plaintext data

```

```

CryptoResult<byte[], KmsMasterKey> encryptResult = crypto.encryptData(
    masterKeyProvider,
    sourcePlaintext,
    encryptionContext);
byte[] ciphertext = encryptResult.getResult();

// Decrypt your ciphertext
CryptoResult<byte[], KmsMasterKey> decryptResult = crypto.decryptData(
    masterKeyProvider,
    ciphertext);
byte[] decrypted = decryptResult.getResult();

```

JavaScript

Ab Version 1.7. x von AWS-Verschlüsselungs-SDK for JavaScript, Sie können die Commitment-Richtlinie festlegen, wenn Sie die neue `buildClient` Funktion aufrufen, die einen AWS Encryption SDK Client instanziiert. Die `buildClient` Funktion verwendet einen Aufzählungswert, der Ihre Commitment-Richtlinie darstellt. Sie gibt aktualisierte `decrypt` Funktionen `encrypt` und Funktionen zurück, die Ihre Verpflichtungsrichtlinie beim Verschlüsseln und Entschlüsseln durchsetzen.

In der letzten Version 1. x-Versionen, die `buildClient` Funktion benötigt das `CommitmentPolicy.FORBID_ENCRYPT_ALLOW_DECRYPT` Argument. Ab Version 2.0. x, das Argument der Commitment-Richtlinie ist optional und der Standardwert ist `CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT`.

Der Code für Node.js und der Browser sind für diesen Zweck identisch, außer dass der Browser eine Anweisung benötigt, um Anmeldeinformationen festzulegen.

Im folgenden Beispiel werden Daten mit einem AWS KMS Schlüsselbund verschlüsselt. Die neue `buildClient` Funktion setzt die Commitment-Richtlinie auf `FORBID_ENCRYPT_ALLOW_DECRYPT`, den Standardwert in der letzten Version 1. x Versionen. Das Upgrade `encrypt` und die `decrypt` Funktionen, die `buildClient` zurückgegeben werden, setzen die von Ihnen festgelegte Verpflichtungsrichtlinie durch.

```

import { buildClient } from '@aws-crypto/client-node'
const { encrypt, decrypt } =
  buildClient(CommitmentPolicy.FORBID_ENCRYPT_ALLOW_DECRYPT)

// Create an AWS KMS keyring
const generatorKeyId = 'arn:aws:kms:us-west-2:111122223333:alias/ExampleAlias'

```

```

const keyIds = ['arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab']
const keyring = new KmsKeyringNode({ generatorKeyId, keyIds })

// Encrypt your plaintext data
const { ciphertext } = await encrypt(keyring, plaintext, { encryptionContext:
  context })

// Decrypt your ciphertext
const { decrypted, messageHeader } = await decrypt(keyring, ciphertext)

```

Python

Ab Version 1.7. x von AWS-Verschlüsselungs-SDK for Python, Sie legen die Commitment-Richtlinie für Ihre Instanz von `EncryptionSDKClient`, einem neuen Objekt, das den AWS Encryption SDK Client darstellt. Die von Ihnen festgelegte Commitment-Richtlinie gilt für alle `decrypt` Aufrufe `encrypt` und Aufrufe, die diese Instanz des Clients verwenden.

In der letzten Version 1. X-Versionen, der `EncryptionSDKClient` Konstruktor benötigt den `CommitmentPolicy.FORBID_ENCRYPT_ALLOW_DECRYPT` Aufzählungswert. Ab Version 2.0. x, das Argument der Commitment-Richtlinie ist optional und der Standardwert ist `CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT`.

In diesem Beispiel wird der neue `EncryptionSDKClient` Konstruktor verwendet und die Commitment-Richtlinie auf 1.7 gesetzt. x Standardwert. Der Konstruktor instanziiert einen Client, der den darstellt. AWS Encryption SDK Wenn Sie die `stream` Methoden `encryptdecrypt`, oder auf diesem Client aufrufen, setzen sie die von Ihnen festgelegte Commitment-Richtlinie durch. In diesem Beispiel wird auch der neue Konstruktor für die `StrictAwsKmsMasterKeyProvider` Klasse verwendet, der festlegt, AWS KMS keys wann verschlüsselt und entschlüsselt werden soll.

[Ein vollständiges Beispiel finden Sie unter `set_commitment.py`.](#)

```

# Instantiate the client
client =
  aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.FORBID_ENCRYPT_ALLOW_DECRYPT)

// Create a master key provider in strict mode
aws_kms_key = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
aws_kms_strict_master_key_provider = StrictAwsKmsMasterKeyProvider(
  key_ids=[aws_kms_key]
)

```

```
# Encrypt your plaintext data
ciphertext, encrypt_header = client.encrypt(
    source=source_plaintext,
    encryption_context=encryption_context,
    master_key_provider=aws_kms_strict_master_key_provider
)

# Decrypt your ciphertext
decrypted, decrypt_header = client.decrypt(
    source=ciphertext,
    master_key_provider=aws_kms_strict_master_key_provider
)
```

Problembehandlung bei der Migration auf die neuesten Versionen

Bevor Sie Ihre Anwendung auf Version 2.0 aktualisieren, aktualisieren Sie später die AWS Encryption SDK, aktualisieren Sie auf die neueste Version 1.X der AWS Encryption SDK und setzen Sie es vollständig ein. Auf diese Weise können Sie die meisten Fehler vermeiden, die beim Update auf Version 2.0 auftreten können. Und später. Eine ausführliche Anleitung, einschließlich Beispielen, finden Sie unter [Migrieren Sie Ihre AWS Encryption SDK](#).

Important

Stellen Sie sicher, dass Ihre neueste 1.X-Version ist Version 1.7. Und später die AWS Encryption SDK.

Note

AWS-Verschlüsselung CLI: Verweise auf Version 1.7 in diesem Handbuch. Und der AWS Encryption SDK gelten für Version 1.8. Und der AWS-Verschlüsselungs-CLI. Verweise auf Version 2.0 in diesem Handbuch. Und der AWS Encryption SDK. Bewerben Sie sich auf 2.1. Und der AWS-Verschlüsselungs-CLI.

Neue Sicherheitsfunktionen wurden ursprünglich in veröffentlichten AWS-Verschlüsselung CLI Versionen 1.7. Und 2.0. X. Allerdings AWS-Verschlüsselung CLI Version 1.8. X ersetzt Version 1.7. Und AWS-Verschlüsselung CLI 2.1. X ersetzt 2.0. X. Details dazu finden Sie unter [Sicherheits-Beratung](#) in der [aws-encryption-sdk-cli](#) Repository auf GitHub.

Dieses Thema soll Ihnen helfen, die am häufigsten auftretenden Fehler zu erkennen und zu beheben.

Themen

- [Veraltete oder entfernte Objekte](#)
- [Konfigurationskonflikt: Commitment Policy- und Algorithmus](#)
- [Konfigurationskonflikt: Verpflichtungspolitik und Chiffretext](#)
- [Validierung der wichtigsten Verpflichtungen fehlgeschlagen](#)
- [Andere Verschlüsselungsfehler](#)
- [Andere Fehler bei der Entschlüsselung](#)
- [Überlegungen zum Rollback](#)

Veraltete oder entfernte Objekte

Version 2.0.X enthält mehrere bahnbrechende Änderungen, einschließlich des Entfernens von Legacy-Konstruktor, Methoden, Funktionen und Klassen, die in Version 1.7 veraltet waren. Um Compilerfehler, Importfehler, Syntaxfehler und Fehler „Symbol nicht gefunden“ (abhängig von Ihrer Programmiersprache) zu vermeiden, aktualisieren Sie zuerst auf die neueste Version 1.X des AWS Encryption SDK für Ihre Programmiersprache. (Dies muss Version 1.7 sein oder später.) Während der Verwendung der neuesten Version 1.X können Sie mit der Verwendung der Ersatzelemente beginnen, bevor die ursprünglichen Symbole entfernt werden.

Wenn Sie auf Version 2.0 aktualisieren müssen oder später sofort, [konsultiere das Changelog](#) für Ihre Programmiersprache, und ersetzen Sie die Legacy-Symbole durch die vom Changelog empfohlenen Symbole.

Konfigurationskonflikt: Commitment Policy- und Algorithmus

Wenn Sie eine Algorithmus-Suite angeben, die mit Ihrem [Commit](#) schlägt der Aufruf von `encrypt` fehl mit einem Konflikt bei der Konfiguration Fehler.

Um diese Art von Fehler zu vermeiden, geben Sie keine Algorithmus-Suite an. Standardmäßig ist AWS Encryption SDK wählt den sichersten Algorithmus, der mit Ihrer Verpflichtungsrichtlinie kompatibel ist. Wenn Sie jedoch eine Algorithmus-Suite angeben müssen, z. B. eine ohne Signierung, stellen Sie sicher, dass Sie eine Algorithmus-Suite auswählen, die mit Ihrer Verpflichtungsrichtlinie kompatibel ist.

Richtlinien für Verpflichtung	Kompatible Algorithmen Suites
ForbidEncryptAllowDecrypt	Jede Algorithmus-Suite ohne wichtige Verpflichtung, wie zum Beispiel: AES_256_GCM_IV12_TAG16_HKDF_SHA384_ECDSA_P384 (03 78) (mit Unterschrift) AES_256_GCM_IV12_TAG16_HKDF_SHA256 (01 78) (ohne Unterschrift)
RequireEncryptAllowDecrypt RequireEncryptRequireDecrypt	Jede Algorithmus-Suite mit wichtiger Verpflichtung, wie zum Beispiel: AES_256_GCM_HKDF_SHA512_COMMIT_KEY_ECDSA_P384 (05 78) (mit Unterschrift) AES_256_GCM_HKDF_SHA512_COMMIT_KEY (04 78) (ohne Unterschrift)

Wenn dieser Fehler auftritt, wenn Sie keine Algorithmus-Suite angegeben haben, wurde die widersprüchliche Algorithmus-Suite möglicherweise von Ihrem [Kryptografischer Materialmanager](#) (CMM). Das Standard-CMM wählt keine widersprüchliche Algorithmus-Suite aus, ein benutzerdefiniertes CMM jedoch möglicherweise. Informationen dazu finden Sie in der Dokumentation zu Ihrem benutzerdefinierten CMM.

Konfigurationskonflikt: Verpflichtungspolitik und Chiffretext

Die `RequireEncryptRequireDecrypt` [Committer](#) erlaubt nicht AWS Encryption SDK um eine Nachricht zu entschlüsseln, die ohne verschlüsselt wurde [wichtige Verpflichtung](#). Wenn du das fragst AWS Encryption SDK um eine Nachricht ohne Schlüsselbindung zu entschlüsseln, wird ein Konflikt bei der Konfiguration Fehler.

Um diesen Fehler zu vermeiden, müssen Sie vor dem Setzen der `RequireEncryptRequireDecrypt` Commitment-Richtlinie, stellen Sie sicher, dass alle Chiffretexte, die ohne Schlüsselbindung verschlüsselt wurden, mit Schlüsselbindung entschlüsselt und neu verschlüsselt oder von einer anderen Anwendung verarbeitet werden. Wenn dieser Fehler

auftritt, können Sie einen Fehler für den widersprüchlichen Chiffretext zurückgeben oder Ihre Verpflichtungsrichtlinie vorübergehend in `ändernRequireEncryptAllowDecrypt`.

Wenn dieser Fehler auftritt, weil Sie auf Version 2.0 aktualisiert haben. Xoder höher aus einer Version vor 1.7. Xohne vorher auf die neueste Version zu aktualisieren 1.XVersion (Version 1.7.Xoder später), überlegen Sie [Rollback](#) bis zum neuesten 1.XVersion und Bereitstellung dieser Version auf allen Hosts vor dem Upgrade auf Version 2.0.Xoder später. Weitere Informationen dazu finden Sie unter [So migrieren und implementieren Sie dasAWS Encryption SDK](#).

Validierung der wichtigsten Verpflichtungen fehlgeschlagen

Wenn Sie Nachrichten entschlüsseln, die mit Schlüsselbindung verschlüsselt sind, erhalten Sie möglicherweise eine Validierung der wichtigsten Verpflichtungen fehlgeschlagen. Dies zeigt an, dass der Entschlüsselungsaufwurf fehlgeschlagen ist, weil ein Datenschlüssel in einem [verschlüsselte Mitteilung](#) ist nicht identisch mit dem eindeutigen Datenschlüssel für die Nachricht. Durch die Validierung des Datenschlüssels während der Entschlüsselung [wichtige Verpflichtung](#) schützt Sie vor der Entschlüsselung einer Nachricht, die zu mehr als einem Klartext führen kann.

Dieser Fehler weist darauf hin, dass die verschlüsselte Nachricht, die Sie zu entschlüsseln versuchten, nicht vom AWS Encryption SDK. Es kann sich um eine manuell erstellte Nachricht oder das Ergebnis einer Datenbeschädigung handeln. Wenn dieser Fehler auftritt, kann Ihre Anwendung die Nachricht zurückweisen und fortfahren oder die Verarbeitung neuer Nachrichten beenden.

Andere Verschlüsselungsfehler

Die Verschlüsselung kann aus mehreren Gründen fehlschlagen. Sie können keinen [AWS KMS Discovery Schlüsselring](#) oder ein [Master Key-Anbieter im Discovery-Modus](#) um eine Nachricht zu verschlüsseln.

Stellen Sie sicher, dass Sie einen Schlüsselbund- oder Hauptschlüsselanbieter mit Umbruchschlüsseln angeben, die Sie haben [Berechtigung zur Nutzung](#) für Verschlüsselung. Für Hilfe bei Berechtigungen auf AWS KMS keys, siehe [Anzeigen einer Schlüsselrichtlinie](#) und [Bestimmen des Zugriffs auf AWS KMS key](#) in der AWS Key Management Service-Entwicklerhandbuch.

Andere Fehler bei der Entschlüsselung

Wenn Ihr Versuch, eine verschlüsselte Nachricht zu entschlüsseln, fehlschlägt, bedeutet dies, dass die AWS Encryption SDK konnte (oder würde) keinen der verschlüsselten Datenschlüssel in der Nachricht entschlüsseln.

Wenn Sie einen Schlüsselbund- oder Hauptschlüsselanbieter verwendet haben, der umlaufende Schlüssel angibt, AWS Encryption SDK verwendet nur die von Ihnen angegebenen Wrapping-Schlüssel. Stellen Sie sicher, dass Sie die beabsichtigten Wickelschlüssel verwenden, die Sie verwenden und dass Sie `Decrypt`-Berechtigung auf mindestens einen der Wrapping-Schlüssel. Wenn Sie AWS KMS keys, als Fallback können Sie versuchen, die Nachricht mit einem [AWS KMS Discovery Schlüsselanhänger](#) oder ein [Master Key-Anbieter im Discovery-Modus](#). Wenn der Vorgang erfolgreich ist, stellen Sie vor der Rückgabe des Klartextes sicher, dass der Schlüssel, der zum Entschlüsseln der Nachricht verwendet wird, vertrauenswürdig ist.

Überlegungen zum Rollback

Wenn Ihre Anwendung Daten nicht ver- oder entschlüsseln kann, können Sie das Problem normalerweise lösen, indem Sie die Codesymbole, Schlüsselringe, Hauptschlüsselanbieter oder [Commitment-Politik](#). In einigen Fällen entscheiden Sie sich jedoch möglicherweise dafür, dass es am besten ist, Ihre Anwendung auf eine frühere Version des AWS Encryption SDK.

Wenn Sie einen Rollback durchführen müssen, tun Sie dies mit Vorsicht. Versionen des AWS Encryption SDK vor 1.7.X verschlüsselter Chiffretext kann nicht entschlüsselt werden mit [wichtige Verpflichtung](#).

- Rollback vom neuesten 1.X Version auf eine frühere Version des AWS Encryption SDK ist im Allgemeinen sicher. Möglicherweise müssen Sie Änderungen, die Sie an Ihrem Code vorgenommen haben, rückgängig machen, um Symbole und Objekte zu verwenden, die in früheren Versionen nicht unterstützt wurden.
- Sobald Sie mit der Verschlüsselung mit Key Commitment begonnen haben (setzen Sie Ihre Commitment-Richtlinie auf `RequireEncryptAllowDecrypt`) in Version 2.0.X oder höher können Sie auf Version 1.7 zurücksetzen, aber nicht zu einer früheren Version. Versionen des AWS Encryption SDK vor 1.7.X verschlüsselter Chiffretext kann nicht entschlüsselt werden mit [wichtige Verpflichtung](#).

Wenn Sie versehentlich die Verschlüsselung mit Schlüsselübergabe aktivieren, bevor alle Hosts mit Schlüsselbindung entschlüsseln können, ist es möglicherweise besser, mit dem Rollout fortzufahren, anstatt ein Rollback durchzuführen. Wenn Nachrichten vorübergehend sind oder sicher verworfen werden können, sollten Sie einen Rollback mit Nachrichtenverlust in Betracht ziehen. Wenn ein Rollback erforderlich ist, sollten Sie erwägen, ein Tool zu schreiben, das alle Nachrichten entschlüsselt und neu verschlüsselt.

Häufig gestellte Fragen

- [Wie unterscheidet sich das AWS Encryption SDK von den AWS-SDKs?](#)
- [Wie ist das AWS Encryption SDK anders als der Amazon S3 Encryption Client?](#)
- [Welche kryptographischen Algorithmen werden vom AWS Encryption SDK unterstützt, und welcher ist der Standard?](#)
- [Wie wird der Initialisierungsvektor \(IV\) generiert und wo wird er gespeichert?](#)
- [Wie werden die einzelnen Datenschlüssel generiert, verschlüsselt und entschlüsselt?](#)
- [Wie kann ich nachverfolgen, welche Datenschlüssel zum Verschlüsseln meiner Daten verwendet wurden?](#)
- [Wie speichert das AWS Encryption SDK verschlüsselte Datenschlüssel mit ihren verschlüsselten Daten?](#)
- [Wie viel Overhead verursacht das AWS Encryption SDK-Nachrichtenformat für meine verschlüsselten Daten?](#)
- [Kann ich meinen eigenen Masterschlüsselanbieter verwenden?](#)
- [Kann ich Daten unter mehr als einem Umhüllungsschlüssel verschlüsseln?](#)
- [Welche Datentypen kann ich mit dem AWS Encryption SDK verschlüsseln?](#)
- [Wie verschlüsselt und entschlüsselt das AWS Encryption SDK E/A-Streams \(Eingabe/Ausgabe\)?](#)

Wie unterscheidet sich das AWS Encryption SDK von den AWS-SDKs?

Die [AWS-SDKs](#) Bereitstellung von Bibliotheken für die Interaktion mit Amazon Web Services (AWS), einschließlich [AWS Key Management Service \(AWS KMS\)](#) enthalten. Einige der Sprachimplementierungen des [AWS Encryption SDK](#), wie das [AWS Encryption SDK für .NET](#), benötigen immer die [AWS SDK](#) in derselben Programmiersprache. Andere Sprachimplementierungen erfordern die entsprechende [AWS SDK](#) nur wenn Sie verwenden [AWS KMS](#) Schlüssel in Ihren Schlüsselringen oder Hauptschlüsselanbietern. Weitere Informationen finden Sie im Thema Ihrer Programmiersprache unter [AWS Encryption SDK-Programmiersprachen](#) aus.

Sie können das [AWS SDK](#) zum Interagieren [AWS KMS](#), einschließlich der Verschlüsselung und Entschlüsselung kleiner Datenmengen (bis zu 4.096 Byte mit einem symmetrischen Verschlüsselungsschlüssel) und Generieren von Datenschlüsseln für die clientseitige Verschlüsselung. Wenn Sie jedoch einen Datenschlüssel generieren, müssen Sie den gesamten

Verschlüsselungs- und Entschlüsselungsprozess verwalten, einschließlich der Verschlüsselung Ihrer Daten mit dem Datenschlüssel außerhalb von AWS KMS, indem Sie den Klartext-Datenschlüssel sicher verwerfen, den verschlüsselten Datenschlüssel speichern und dann den Datenschlüssel entschlüsseln und Ihre Daten entschlüsseln. Die AWS Encryption SDK wickelt diesen Prozess für Sie ab.

Die AWS Encryption SDK stellt eine Bibliothek bereit, die Daten mithilfe von Industriestandards und Best Practices verschlüsselt und entschlüsselt. Es generiert den Datenschlüssel, verschlüsselt ihn unter den von Ihnen angegebenen Umbruchschlüsseln und gibt eine verschlüsselte Nachricht, einem tragbaren Datenobjekt, das die verschlüsselten Daten und die verschlüsselten Datenschlüssel enthält, die Sie für die Entschlüsselung benötigen. Wenn es Zeit zum Entschlüsseln ist, geben Sie die verschlüsselte Nachricht und mindestens einen der Umschlagschlüssel (optional) und die AWS Encryption SDK gibt Ihre Klartext-Daten zurück.

Sie können AWS KMS keys als Schlüsselpackung im AWS Encryption SDK, aber es ist nicht erforderlich. Sie können Verschlüsselungsschlüssel verwenden, die Sie generieren, und die von Ihrem Schlüsselmanager oder dem lokalen Hardwaresicherheitsmodul. Sie können das AWS Encryption SDK auch wenn Sie kein AWS Konto.

Wie ist das AWS Encryption SDK anders als der Amazon S3 Encryption Client?

Die [Amazon S3 Encryption Client](#) im AWS SDKs unterstützt Verschlüsselung und Entschlüsselung für Daten, die Sie im Amazon Simple Storage Service (Amazon S3) speichern. Diese Kunden sind eng mit Amazon S3 gekoppelt und ausschließlich für die Verwendung mit den dort gespeicherten Daten vorgesehen.

Das AWS Encryption SDK unterstützt Verschlüsselung und Entschlüsselung für Daten, die Sie an einem beliebigen Ort speichern können. Die AWS Encryption SDK und der Amazon S3 Encryption Client ist nicht kompatibel, da er Ciphertexte mit unterschiedlichen Datenformaten erzeugt.

Welche kryptographischen Algorithmen werden vom AWS Encryption SDK unterstützt, und welcher ist der Standard?

Die AWS Encryption SDK verwenden Sie den symmetrischen Algorithmus Advanced Encryption Standard (GCM) im Galois/Counter Mode (GCM), als AES-GCM bezeichnet, zur Verschlüsselung Ihrer Daten. Sie können aus mehreren symmetrischen und asymmetrischen Algorithmen wählen, um die Datenschlüssel zu verschlüsseln, die Ihre Daten verschlüsseln.

Für AES-GCM ist die Standard-Algorithmus-Suite AES-GCM mit einem 256-Bit-Schlüssel, Schlüsselableitung (HKDF), [digitale Signaturen](#), und [Schlüsselverpflichtung](#) aus AWS

Encryption SDK unterstützt außerdem 192- und 128-Bit-Verschlüsselungsschlüssel und Verschlüsselungsalgorithmen ohne digitale Signaturen und Schlüsselverpflichtungen.

Die Länge des Initialisierungsvektors (IV) ist immer 12 Bytes; die Länge des Authentifizierungstags ist immer 16 Bytes. Standardmäßig verwendet das SDK den Datenschlüssel als Eingabe für den HMAC-basierten extract-and-expand Die Schlüsselableitungsfunktion (HKDF), um den AES-GCM-Verschlüsselungsschlüssel abzuleiten, und fügt außerdem eine ECDSA-Signatur (Elliptic Curve Digital Signature Algorithm) hinzu.

Weitere Informationen zum Auswählen des Algorithmus finden Sie unter [Unterstützte Algorithmen-Pakete](#).

Weitere Informationen zu den unterstützten Algorithmen finden Sie unter [Algorithmen – Referenz](#).

Wie wird der Initialisierungsvektor (IV) generiert und wo wird er gespeichert?

Die AWS Encryption SDK verwenden Sie eine deterministische Methode, um für jeden Frame einen anderen IV-Wert zu konstruieren. Dieses Verfahren garantiert, dass IVs niemals innerhalb einer Nachricht wiederholt werden. (Vor Version 1.3.0 des AWS-Verschlüsselungs-SDK for Java und die AWS-Verschlüsselungs-SDK for Python, der AWS Encryption SDKs hat zufällig einen eindeutigen IV-Wert für jeden Frame generiert.)

Das IV wird in der verschlüsselten Nachricht gespeichert, das die AWS Encryption SDK gibt zurück. Weitere Informationen hierzu finden Sie unter [AWS Encryption SDK Referenz zum Nachrichtenformat](#).

Wie werden die einzelnen Datenschlüssel generiert, verschlüsselt und entschlüsselt?

Die Methode hängt von dem verwendeten Schlüsselbund oder Masterschlüssel-Anbieter ab.

Die AWS KMS Schlüsselringe und Master Key Provider im AWS Encryption SDK verwenden Sie das AWS KMS [GenerateDataKey](#) API-Operation, um jeden Datenschlüssel zu generieren und unter seinem Umschlagschlüssel zu verschlüsseln. Um Kopien des Datenschlüssels unter zusätzlichen KMS-Schlüsseln zu verschlüsseln, verwenden Sie die AWS KMS [Encrypt](#) verwenden. Um die Datenschlüssel zu entschlüsseln, verwenden Sie die AWS KMS [Decrypt](#) verwenden. Details dazu finden Sie unter [AWS KMS Schlüsselring](#) im AWS Encryption SDK Spezifikation in GitHub aus.

Andere Schlüsselringe generieren den Datenschlüssel, verschlüsseln und entschlüsseln mit Best Practice-Methoden für jede Programmiersprache. Einzelheiten finden Sie in der Spezifikation des Schlüsselbunds oder Hauptschlüsselanbieters im [Framework](#) der AWS Encryption SDK Spezifikation in GitHub aus.

Wie kann ich nachverfolgen, welche Datenschlüssel zum Verschlüsseln meiner Daten verwendet wurden?

Die AWS Encryption SDK erledigt dies für Sie. Wenn Sie Daten verschlüsseln, verschlüsselt das SDK den Datenschlüssel und speichert den verschlüsselten Schlüssel zusammen mit den verschlüsselten Daten in der [verschlüsselten Nachricht](#), die es zurückgibt. Wenn Sie Daten entschlüsseln, extrahiert das AWS Encryption SDK den verschlüsselten Datenschlüssel aus der verschlüsselten Nachricht, entschlüsselt ihn und verwendet ihn dann zur Entschlüsselung der Daten.

Wie speichert das AWS Encryption SDK verschlüsselte Datenschlüssel mit ihren verschlüsselten Daten?

Die Verschlüsselungsoperationen im AWS Encryption SDK geben eine [verschlüsselte Nachricht](#) zurück, eine einzelne Datenstruktur, die alle verschlüsselten Daten und ihre verschlüsselten Datenschlüssel enthält. Das Nachrichtenformat besteht aus mindestens zwei Teilen: einem Header und einem Text. Der Nachrichten-Header enthält die verschlüsselten Datenschlüssel sowie Informationen darüber, wie der Nachrichtentext gebildet wird. Der Nachrichtentext enthält die verschlüsselten Daten. Wenn die Algorithmus-Suite ein [digitale Signatur](#) enthält das NachrichtenformatFußzeileDas enthält die Signatur. Weitere Informationen finden Sie unter [AWS Encryption SDK Referenz zum Nachrichtenformat](#).

Wie viel Overhead verursacht das AWS Encryption SDK-Nachrichtenformat für meine verschlüsselten Daten?

Die Höhe des durch das AWS Encryption SDK verursachten Overheads hängt von mehreren Faktoren ab, unter anderem:

- Der Größe der Klartextdaten
- Welche der unterstützten Algorithmen verwendet werden
- Ob zusätzliche authentifizierte Daten (AAD) bereitgestellt werden, und von der Länge dieser AAD
- Die Anzahl und Art der Umhüllungsschlüssel oder Masterschlüssel
- Der Framegröße (wenn [Daten mit Frame](#) verwendet werden)

Wenn Sie das AWS Encryption SDK mit seiner Standardkonfiguration (eins AWS KMS key Als Umhüllungsschlüssel (oder Masterschlüssel), kein AAD, nicht gerahmte Daten und ein Verschlüsselungsalgorithmus mit Signatur) beträgt der Overhead ca. 600 Bytes. Im Allgemeinen können Sie davon ausgehen, dass das AWS Encryption SDK einen Overhead von 1 KB

oder weniger verursacht, wobei die bereitgestellten AAD nicht berücksichtigt sind. Weitere Informationen finden Sie unter [AWS Encryption SDK Referenz zum Nachrichtenformat](#).

Kann ich meinen eigenen Masterschlüsselanbieter verwenden?

Ja. Die Implementierungsdetails variieren abhängig davon, welche der [unterstützten Programmiersprachen](#) Sie verwenden. In allen unterstützten Sprachen können Sie jedoch benutzerdefinierte definieren [Manager für kryptografische Materialien \(CMMs\)](#), Hauptschlüsselanbieter, Schlüsselringe, Hauptschlüssel und Wrapping Keys.

Kann ich Daten unter mehr als einem Umhüllungsschlüssel verschlüsseln?

Ja. Sie können den Datenschlüssel mit zusätzlichen Umhüllungsschlüsseln (oder Masterschlüsseln) verschlüsseln, um die Redundanz zu erhöhen, wenn sich der Schlüssel in einer anderen Region befindet oder für die Entschlüsselung nicht verfügbar ist.

Um Daten unter mehreren Umhüllungsschlüsseln zu verschlüsseln, legen Sie einen Schlüsselbund- oder Masterschlüsselanbieter mit mehreren Umhüllungsschlüsseln an. Wenn Sie mit Schlüsselbunden arbeiten, können Sie einen [einzelnen Schlüsselbund mit mehreren Umhüllungsschlüsseln](#) oder einen [Multi-Schlüsselbund](#) erstellen.

Wenn Sie Daten mit mehreren Umbruchschlüsseln verschlüsseln, wird der AWS Encryption SDK verwendet einen Umschlagschlüssel, um einen Klartext-Datenschlüssel zu erzeugen. Der Datenschlüssel ist eindeutig und mathematisch nichts mit dem Umschlagschlüssel zu tun. Der Vorgang gibt den Klartext-Datenschlüssel und eine mit dem Umhüllungsschlüssel verschlüsselte Kopie des Datenschlüssels zurück. Dann verschlüsselt die Verschlüsselungsmethode den Datenschlüssel mit den anderen Umhüllungsschlüsseln. Das resultierende [Verschlüsselte Nachricht](#) Enthält die verschlüsselten Daten und einen verschlüsselten Datenschlüssel für jeden Umhüllungsschlüssel.

Die verschlüsselte Nachricht kann mit einem beliebigen der bei der Verschlüsselungsoperation verwendeten Umhüllungsschlüssel entschlüsselt werden. Die AWS Encryption SDK verwenden Sie einen Umhüllungsschlüssel, um einen verschlüsselten Datenschlüssel zu entschlüsseln. Dann verwendet es den Klartext-Datenschlüssel, um die Daten zu entschlüsseln.

Welche Datentypen kann ich mit dem AWS Encryption SDK verschlüsseln?

Die meisten Programmiersprachen-Implementierungen des AWS Encryption SDK kann Rohbytes (Byte-Arrays), E/A-Streams (Byte-Streams) und Zeichenfolgen verschlüsseln. Die AWS Encryption SDK für .NET unterstützt keine E/A-Streams. Wir stellen Beispielcode für jede der [unterstützten Programmiersprachen](#) zur Verfügung.

Wie verschlüsselt und entschlüsselt das AWS Encryption SDK E/A-Streams (Eingabe/Ausgabe)?

Das AWS Encryption SDK erzeugt einen verschlüsselnden oder entschlüsselnden Stream, der einen zugrunde liegenden E/A-Stream umhüllt. Der verschlüsselnde oder entschlüsselnde Stream führt bei einem Lese- oder Schreibaufwurf eine kryptographische Operation durch. Beispielsweise kann er Klartextdaten aus dem zugrundeliegenden Stream lesen und verschlüsseln, bevor er das Ergebnis zurückgibt. Oder er kann Verschlüsselungstext aus einem zugrundeliegenden Stream lesen und entschlüsseln, bevor er das Ergebnis zurückgibt. Wir stellen Beispielcode für das Verschlüsseln und Entschlüsseln von Streams für jeden der [unterstützte Programmiersprachen](#) das unterstützt Streaming.

Die AWS Encryption SDK für .NET unterstützt keine E/A-Streams.

AWS Encryption SDK Referenz

Die Informationen auf dieser Seite stellen eine Referenz für die Erstellung Ihrer eigenen, mit AWS Encryption SDK kompatiblen Verschlüsselungsbibliothek dar. Falls Sie keine eigene kompatible Verschlüsselungsbibliothek erstellen, benötigen Sie diese Informationen wahrscheinlich nicht.

Informationen zur Verwendung von AWS Encryption SDK in einer der unterstützten Programmiersprachen finden Sie unter [Programmiersprachen](#).

Die Spezifikation, die die Elemente einer ordnungsgemäßen AWS Encryption SDK Implementierung definiert, finden Sie in der [AWS Encryption SDK Spezifikation](#) unter GitHub.

Der AWS Encryption SDK verwendet die [unterstützten Algorithmen](#), um eine einzelne Datenstruktur oder Nachricht zurückzugeben, die verschlüsselte Daten und die entsprechenden verschlüsselten Datenschlüssel enthält. Die folgenden Themen erläutern die Algorithmen und die Datenstruktur. Verwenden Sie diese Informationen, um Bibliotheken zu erstellen, die Chiffretexte lesen und schreiben können, die damit kompatibel sind. SDK

Themen

- [AWS Encryption SDK Referenz zum Nachrichtenformat](#)
- [AWS Encryption SDK Beispiele für Nachrichtenformate](#)
- [Hauptteil der Referenz für zusätzliche authentifizierte Daten \(\) für AAD AWS Encryption SDK](#)
- [AWS Encryption SDK Referenz zu Algorithmen](#)
- [AWS Encryption SDK Referenz auf den Initialisierungsvektor](#)
- [AWS KMS Technische Details zum hierarchischen Schlüsselbund](#)

AWS Encryption SDK Referenz zum Nachrichtenformat

Die Informationen auf dieser Seite stellen eine Referenz für die Erstellung Ihrer eigenen, mit AWS Encryption SDK kompatiblen Verschlüsselungsbibliothek dar. Falls Sie keine eigene kompatible Verschlüsselungsbibliothek erstellen, benötigen Sie diese Informationen wahrscheinlich nicht.

Informationen zur Verwendung von AWS Encryption SDK in einer der unterstützten Programmiersprachen finden Sie unter [Programmiersprachen](#).

Die Spezifikation, die die Elemente einer ordnungsgemäßen AWS Encryption SDK Implementierung definiert, finden Sie in der [AWS Encryption SDK Spezifikation](#) unter GitHub.

Die Verschlüsselungsoperationen AWS Encryption SDK geben eine einzelne Datenstruktur oder eine [verschlüsselte Nachricht](#) zurück, die die verschlüsselten Daten (Chiffretext) und alle verschlüsselten Datenschlüssel enthält. Um diese Datenstruktur zu verstehen, oder zum Erstellen von Bibliotheken, die sie lesen und schreiben können, müssen Sie das Nachrichtenformat verstehen.

Das Nachrichtenformat besteht aus mindestens zwei Teilen: einem Header und einem Text. In einigen Fällen enthält das Nachrichtenformat einen dritten Teil, einen Footer. Das Nachrichtenformat definiert eine sortierte Reihenfolge von Bytes in Netzwerk-Bytereihenfolge, auch als Big-Endian-Format bezeichnet. Das Nachrichtenformat beginnt mit dem Header, gefolgt vom Text, gefolgt vom Footer (falls vorhanden).

Die [Algorithmus-Suiten](#), die von der unterstützt werden, AWS Encryption SDK verwenden eine von zwei Versionen im Nachrichtenformat. Algorithmus-Suiten ohne [Schlüsselbindung](#) verwenden das Nachrichtenformat Version 1. Algorithmus-Suiten mit Key Commitment verwenden das Nachrichtenformat Version 2.

Themen

- [Header-Struktur](#)
- [Textstruktur](#)
- [Footer-Struktur](#)

Header-Struktur

Der Nachrichten-Header enthält den verschlüsselten Datenschlüssel sowie Informationen darüber, wie der Nachrichtentext gebildet wird. In der folgenden Tabelle werden die Felder beschrieben, die den Header in den Nachrichtenformat-Versionen 1 und 2 bilden. Die Byte werden in der angegebenen Reihenfolge angehängt.

Der Wert Nicht vorhanden gibt an, dass das Feld in dieser Version des Nachrichtenformats nicht vorhanden ist. Fettgedruckter Text weist auf Werte hin, die in jeder Version unterschiedlich sind.

Note

Möglicherweise müssen Sie horizontal oder vertikal scrollen, um alle Daten in dieser Tabelle anzuzeigen.

Header-Struktur

Feld	Nachrichtenformat, Version 1	Nachrichtenformat Version 2
	Länge (Bytes)	Länge (Bytes)
Version	1	1
Type	1	Nicht vorhanden
Algorithm ID	2	2
Message ID	16	32
AAD Length	2	2
	Wenn der Verschlüsselungskontext leer ist, ist der Wert des AAD 2-Byte-Längenfeldes 0.	Wenn der Verschlüsselungskontext leer ist, ist der Wert des AAD 2-Byte-Längenfeldes 0.
AAD	Variable. Die Länge dieses Felds wird in den vorherigen 2 Byte angezeigt (Feld AAD Länge). Wenn der Verschlüsselungskontext leer ist, gibt es kein AAD Feld in der Kopfzeile.	Variable. Die Länge dieses Felds wird in den vorherigen 2 Byte angezeigt (Feld AAD Länge). Wenn der Verschlüsselungskontext leer ist, gibt es kein AAD Feld in der Kopfzeile.
Encrypted Data Key Count	2	2
Encrypted Data Key(s)	Variable. Abhängig von der Anzahl der verschlüsselten	Variable. Abhängig von der Anzahl der verschlüsselten

Feld	Nachrichtenformat, Version 1	Nachrichtenformat Version 2
	Länge (Bytes)	Länge (Bytes)
	Datenschlüssel und ihrer jeweiligen Länge.	Datenschlüssel und ihrer jeweiligen Länge.
Content Type	1	1
Reserved	4	Nicht vorhanden
IV Length	1	Nicht anwesend
Frame Length	4	4
Algorithm Suite Data	Nicht anwesend	Variabel. Wird durch den Algorithmus bestimmt, der die Nachricht generiert hat.
Header Authentication	Variable. Wird durch den Algorithmus bestimmt, der die Nachricht generiert hat.	Variable. Wird durch den Algorithmus bestimmt, der die Nachricht generiert hat.

Ausführung

Die Version dieses Nachrichtenformats. Die Version ist entweder 1 oder 2 als Byte 01 oder 02 in hexadezimaler Schreibweise codiert

Geben Sie ein

Der Typ dieses Nachrichtenformats. Der Typ gibt die Art der Struktur an. Der einzige unterstützte Typ wird als vom Kunden authentifizierte verschlüsselte Daten beschrieben. Sein Typwert ist 128, kodiert als Byte 80 im Hexadezimalformat.

Dieses Feld ist im Nachrichtenformat Version 2 nicht vorhanden.

Algorithmus-ID

Eine ID für den verwendeten Algorithmus. Dies ist ein 2-Byte-Wert, interpretiert als vorzeichenlose 16-Bit-Ganzzahl. Weitere Informationen die Algorithmen finden Sie unter [AWS Encryption SDK Referenz zu Algorithmen](#).

Nachrichten-ID

Ein zufällig generierter Wert, der die Nachricht identifiziert. Die Nachrichten-ID:

- Identifiziert die verschlüsselte Nachricht eindeutig.
- Bindet den Nachrichten-Header schwach an den Nachrichtentext.
- Stellt einen Mechanismus zur sicheren Wiederverwendung eines Datenschlüssels für mehrere verschlüsselte Nachrichten bereit.
- Schützt vor versehentlicher Wiederverwendung eines Datenschlüssels oder der Abnutzung von Schlüsseln im AWS Encryption SDK.

Dieser Wert beträgt 128 Bit im Nachrichtenformat Version 1 und 256 Bit in Version 2.

AADLänge

Die Länge der zusätzlichen authentifizierten Daten (AAD). Es handelt sich um einen 2-Byte-Wert, der als 16-Bit-Ganzzahl ohne Vorzeichen interpretiert wird und die Anzahl der Byte angibt, die den enthalten. AAD

Wenn der [Verschlüsselungskontext](#) leer ist, ist der Wert des Felds AAD Länge 0.

AAD

Die zusätzlich authentifizierten Daten (Additional Authenticated Data, ADD). Das AAD ist eine Kodierung des [Verschlüsselungskontextes](#), ein Array von Schlüssel-Wert-Paaren, wobei jeder Schlüssel und Wert eine Zeichenfolge von UTF-8 codierten Zeichen ist. Der Verschlüsselungskontext wird in eine Bytefolge umgewandelt und für den Wert verwendet. AAD Wenn der Verschlüsselungskontext leer ist, gibt es kein AAD Feld in der Kopfzeile.

Wenn die [Algorithmen mit Signatur](#) verwendet werden, muss der Verschlüsselungskontext das Schlüssel-Wert-Paar {'aws-crypto-public-key', Qtxt} enthalten. Qtxt steht für den elliptischen Kurvenpunkt Q, der gemäß [Version SEC 1 2.0](#) komprimiert und dann Base64-kodiert wurde. Der Verschlüsselungskontext kann zusätzliche Werte enthalten, aber die maximale Länge des Konstruierten AAD beträgt $2^{16} - 1$ Byte.

In der folgenden Tabelle werden die Felder beschrieben, die den bilden. AAD Schlüssel-Wert-Paare werden nach Schlüsseln in aufsteigender Reihenfolge gemäß dem UTF-8-stelligen Code sortiert. Die Byte werden in der angegebenen Reihenfolge angehängt.

AADStruktur

Feld	Länge (Bytes)
Key-Value Pair Count	2
Key Length	2
Key	Variable. Gleich dem Wert, der in den vorherigen 2 Bytes angegeben ist (Schlüssel­länge).
Value Length	2
Value	Variable. Gleich dem Wert, der in den vorherigen 2 Bytes angegeben ist (Wert­länge).

Anzahl der Schlüssel-Wert-Paare

Die Anzahl der Schlüssel-Wert-Paare in der AAD. Es handelt sich um einen 2-Byte-Wert, der als 16-Bit-Ganzzahl ohne Vorzeichen interpretiert wird und die Anzahl der Schlüssel-Wert-Paare in der angibt. AAD Die maximale Anzahl von Schlüssel-Wert-Paaren in der ist $2^{16} - 1$. AAD

Wenn kein Verschlüsselungskontext vorhanden ist oder der Verschlüsselungskontext leer ist, ist dieses Feld in der Struktur nicht vorhanden. AAD

Länge des Schlüssels

Die Länge des Schlüssels für das Schlüssel-Wert-Paar. Es handelt sich um einen 2-Byte-Wert, interpretiert als vorzeichenlose 16-Bit-Ganzzahl, die die Anzahl der Bytes angibt, die den Schlüssel enthalten.

Schlüssel

Der Schlüssel für das Schlüssel-Wert-Paar. Es ist eine Sequenz von UTF-8 codierten Bytes.

Wert: Länge

Die Länge des Werts für das Schlüssel-Wert-Paar. Es handelt sich um einen 2-Byte-Wert, interpretiert als vorzeichenlose 16-Bit-Ganzzahl, die die Anzahl der Bytes angibt, die den Wert enthalten.

Wert

Der Wert für das Schlüssel-Wert-Paar. Es ist eine Sequenz von UTF -8 codierten Bytes.

Anzahl verschlüsselter Datenschlüssel

Die Anzahl der verschlüsselten Datenschlüssel. Es handelt sich um einen 2-Byte-Wert, interpretiert als vorzeichenlose 16-Bit-Ganzzahl, die die Anzahl der verschlüsselten Datenschlüssel angibt. Die maximale Anzahl verschlüsselter Datenschlüssel in jeder Nachricht beträgt 65.535 ($2^{16} - 1$).

Verschlüsselte Datenschlüssel

Eine Folge von verschlüsselten Datenschlüsseln. Die Länge der Folge wird durch die Anzahl der verschlüsselten Datenschlüssel und ihre jeweilige Länge bestimmt. Die Folge enthält mindestens einen verschlüsselten Datenschlüssel.

In der folgenden Tabelle sind die Felder beschrieben, die die verschlüsselten Datenschlüssel bilden. Die Byte werden in der angegebenen Reihenfolge angehängt.

Struktur der verschlüsselten Datenschlüssel

Feld	Länge (Bytes)
Key Provider ID Length	2
Key Provider ID	Variable. Gleich dem Wert, der in den vorherigen 2 Bytes angegeben ist (Länge der Schlüsselanbieter-ID).
Key Provider Information Length	2
Key Provider Information	Variable. Gleich dem Wert, der in den vorherigen 2 Bytes angegeben ist (Länge der Schlüsselanbieterinformation).
Encrypted Data Key Length	2

Feld	Länge (Bytes)
Encrypted Data Key	Variable. Gleich dem Wert, der in den vorherigen 2 Bytes angegeben ist (Länge des verschlüsselten Datenschlüssels).

Länge der Schlüsselanbieter-ID

Die Länge der Schlüsselanbieter-ID. Es handelt sich um einen 2-Byte-Wert, interpretiert als vorzeichenlose 16-Bit-Ganzzahl, die die Anzahl der Bytes angibt, die die Schlüsselanbieter-ID enthalten.

ID des Schlüsselanbieters

Die Schlüsselanbieter-ID. Wird verwendet, um den Anbieter des verschlüsselten Datenschlüssels anzugeben, und ist auf Erweiterbarkeit ausgelegt.

Länge der Informationen zum Schlüsselanbieter

Die Länge der Schlüsselanbieterinformation. Es handelt sich um einen 2-Byte-Wert, interpretiert als vorzeichenlose 16-Bit-Ganzzahl, die die Anzahl der Bytes angibt, die die Schlüsselanbieterinformation enthalten.

Informationen zu den wichtigsten Anbietern

Die Schlüsselanbieterinformation. Wird durch den Schlüsselanbieter bestimmt.

Wenn der Hauptschlüsselanbieter AWS KMS ist oder Sie einen AWS KMS Schlüsselbund verwenden, enthält dieser Wert den Amazon-Ressourcennamen (ARN) von AWS KMS key

Länge des verschlüsselten Datenschlüssels

Die Länge des verschlüsselten Datenschlüssels. Es handelt sich um einen 2-Byte-Wert, interpretiert als vorzeichenlose 16-Bit-Ganzzahl, die die Anzahl der Bytes angibt, die den verschlüsselten Datenschlüssel enthalten.

Verschlüsselter Datenschlüssel

Der verschlüsselte Datenschlüssel. Dies ist der vom Schlüsselanbieter verschlüsselte Datenverschlüsselungsschlüssel.

Art des Inhalts

Der Typ der verschlüsselten Daten, entweder ohne Frame oder Frame.

Note

Verwenden Sie nach Möglichkeit gerahmte Daten. Das AWS Encryption SDK unterstützt Daten ohne Frames nur für die Verwendung in älteren Versionen. Einige Sprachimplementierungen von AWS Encryption SDK können immer noch nicht gerahmten Chiffretext generieren. Alle unterstützten Sprachimplementierungen können gerahmten und ungerahmten Chiffretext entschlüsseln.

Frame-Daten werden in gleich lange Teile aufgeteilt; jeder Teil wird separat verschlüsselt. Inhalt mit Frame ist Typ 2, kodiert als Byte 02 im Hexadezimalformat.

Daten ohne Frames werden nicht aufgeteilt, sondern sind ein einziger verschlüsselter Blob. Inhalt ohne Frame ist Typ 1, kodiert als Byte 01 im Hexadezimalformat.

Reserviert

Eine reservierte Folge aus 4 Bytes. Der Wert muss 0 sein. Es ist kodiert als die Bytes 00 00 00 00 im Hexadezimalformat (d. h. eine 4-Byte-Folge einer 32-Bit-Ganzzahl mit dem Wert 0).

Dieses Feld ist im Nachrichtenformat Version 2 nicht vorhanden.

IV Länge

Die Länge des Initialisierungsvektors (IV). Es handelt sich um einen 1-Byte-Wert, interpretiert als vorzeichenlose 8-Bit-Ganzzahl, die die Anzahl der Bytes angibt, die den IV enthalten. Dieser Wert wird durch den IV-Byte-Wert des [Algorithmus](#) bestimmt, der die Nachricht generiert hat.

Dieses Feld ist in Version 2 des Nachrichtenformats nicht vorhanden, das nur Algorithmus-Suites unterstützt, die deterministische IV-Werte im Nachrichtenkopf verwenden.

Länge des Frames

Die Länge jedes Frames mit gerahmten Daten. Es handelt sich um einen 4-Byte-Wert, der als 32-Bit-Ganzzahl ohne Vorzeichen interpretiert wird und die Anzahl der Bytes in jedem Frame angibt. Wenn die Daten nicht gerahmt sind, d. h. wenn der Wert des Content Type Felds 1 ist, muss dieser Wert 0 sein.

Note

Verwenden Sie nach Möglichkeit gerahmte Daten. Das AWS Encryption SDK unterstützt Daten ohne Frames nur für die Verwendung in älteren Versionen. Einige

Sprachimplementierungen von AWS Encryption SDK können immer noch nicht gerahmten Chiffretext generieren. Alle unterstützten Sprachimplementierungen können gerahmten und ungerahmten Chiffretext entschlüsseln.

Daten der Algorithm Suite

Zusätzliche Daten, die der [Algorithmus](#) benötigt, der die Nachricht generiert hat. Die Länge und der Inhalt werden durch den Algorithmus bestimmt. Ihre Länge könnte 0 sein.

Dieses Feld ist im Nachrichtenformat Version 1 nicht vorhanden.

Header-Authentifizierung

Die Header-Authentifizierung wird durch den [Algorithmus](#) bestimmt, der die Meldung generiert hat. Die Header-Authentifizierung wird unter Verwendung des kompletten Headers berechnet. Sie besteht aus einem IV und einem Authentifizierungs-Tag. Die Byte werden in der angegebenen Reihenfolge angehängt.

Struktur der Header-Authentifizierung

Feld	Länge in Version 1.0 (Byte)	Länge in Version 2.0 (Byte)
IV	Variable. Wird durch den IV-Byte-Wert des Algorithmus bestimmt, der die Nachricht generiert hat.	N/A
Authentication Tag	Variable. Wird durch den Authentifizierungs-Tag-Byte-Wert des Algorithmus bestimmt, die die Nachricht generiert hat.	Variable. Wird durch den Authentifizierungs-Tag-Byte-Wert des Algorithmus bestimmt, die die Nachricht generiert hat.

IV

Der Initialisierungsvektor (IV) zur Berechnung des Header-Authentifizierungs-Tags.

Dieses Feld ist im Header der Nachrichtenformatversion 2 nicht vorhanden. Nachrichtenformat Version 2 unterstützt nur Algorithmus-Suites, die deterministische IV-Werte im Nachrichtenkopf verwenden.

Authentifizierungs-Tag

Die Authentifizierungswert für den Header. Es wird verwendet, um den gesamten Inhalt des Headers zu authentifizieren.

Textstruktur

Der Nachrichtentext enthält die verschlüsselten Daten, den sogenannten Verschlüsselungstext. Die Struktur des Textes hängt vom Inhaltstyp ab (mit oder ohne Frame). Die folgenden Abschnitte beschreiben das Format des Nachrichtentexts für jeden Inhaltstyp. Die Struktur des Nachrichtentexts ist in den Nachrichtenformat-Versionen 1 und 2 dieselbe.

Themen

- [Daten ohne Frame](#)
- [Daten mit Frame](#)

Daten ohne Frame

[Daten ohne Frames werden in einem einzigen Blob mit einer eindeutigen ID und einem eindeutigen Nachrichtentext verschlüsselt. AAD](#)

Note

Verwenden Sie nach Möglichkeit gerahmte Daten. Das AWS Encryption SDK unterstützt Daten ohne Frames nur für die Verwendung in älteren Versionen. Einige Sprachimplementierungen von AWS Encryption SDK können immer noch nicht gerahmten Chiffretext generieren. Alle unterstützten Sprachimplementierungen können gerahmten und ungerahmten Chiffretext entschlüsseln.

In der folgenden Tabelle sind die Felder beschrieben, die Daten ohne Frame bilden. Die Byte werden in der angegebenen Reihenfolge angehängt.

Struktur von Text ohne Frame

Feld	Länge, in Bytes
IV	Variable. Gleich dem im IV Length -Byte des Headers angegebenen Wert.
Encrypted Content Length	8
Encrypted Content	Variable. Gleich dem Wert, der in den vorherigen 8 Bytes angegeben ist (Länge des verschlüsselten Inhalts).
Authentication Tag	Variable. Wird durch die verwendete Algorithmus-Implementierung bestimmt.

IV

Der Initialisierungsvektor (IV) für die Verwendung mit dem [Verschlüsselungsalgorithmus](#).

Länge des verschlüsselten Inhalts

Die Länge des verschlüsselten Inhalts oder Verschlüsselungstext. Es handelt sich um einen 8-Byte-Wert, interpretiert als vorzeichenlose 64-Bit-Ganzzahl, die die Anzahl der Bytes angibt, die den verschlüsselten Inhalt enthalten.

Technisch ist der maximal zulässige Wert $2^{63} - 1$ oder 8 Exbibytes (8 EiB). In der Praxis ist der maximale Wert jedoch $2^{36} - 32$ oder 64 Gibibyte (64 GiB), aufgrund von Einschränkungen durch die [implementierten Algorithmen](#).

Note

Die Java-Implementierung schränkt diesen Wert aufgrund von Einschränkungen in der Sprache SDK weiter auf $2^{31} - 1$ oder 2 Gibibyte (2 GiB) ein.

Verschlüsselter Inhalt

Der verschlüsselte Inhalt (Verschlüsselungstext), wie vom [Verschlüsselungsalgorithmus](#) zurückgegeben.

Authentifizierungs-Tag

Die Authentifizierungswert für den Text. Er wird verwendet, um den Nachrichtentext zu authentifizieren.

Daten mit Frame

Bei Daten mit Frame werden die Klartextdaten in gleichlange Teile unterteilt, die als Frames bezeichnet werden. Das AWS Encryption SDK verschlüsselt jeden Frame separat mit einer eindeutigen ID und einem eindeutigen [Hauptteil AAD](#).

Note

Verwenden Sie nach Möglichkeit gerahmte Daten. Das AWS Encryption SDK unterstützt Daten ohne Frames nur für die Verwendung in älteren Versionen. Einige Sprachimplementierungen von AWS Encryption SDK können immer noch nicht gerahmten Chiffretext generieren. Alle unterstützten Sprachimplementierungen können gerahmten und ungerahmten Chiffretext entschlüsseln.

Die [Frame-Länge](#), d. h. die Länge des [verschlüsselten Inhalts](#) im Frame, kann für jede Nachricht unterschiedlich sein. Die maximale Anzahl von Bytes in einem Frame ist $2^{32} - 1$. Die maximale Anzahl von Frames in einer Nachricht ist $2^{32} - 1$.

Es gibt zwei Arten von Frames: reguläre und abschließende. Jede Nachricht muss aus einem abschließenden Frame bestehen oder einen enthalten.

Alle regulären Frames in einer Nachricht haben die gleiche Frame-Länge. Der abschließende Frame kann eine andere Frame-Länge haben.

Die Zusammensetzung der Frames in Daten mit Frame variiert je nach Länge des verschlüsselten Inhalts.

- Entspricht der Framelänge — Wenn die Länge des verschlüsselten Inhalts mit der Framelänge der regulären Frames übereinstimmt, kann die Nachricht aus einem regulären Frame bestehen, der die Daten enthält, gefolgt von einem letzten Frame mit der Länge Null (0). Oder die Nachricht kann nur aus einem abschließenden Frame bestehen, der die Daten enthält. In diesem Fall hat der abschließende Frame die gleiche Frame-Länge wie die regulären Frames.

- **Vielfaches der Framelänge** — Wenn die Länge des verschlüsselten Inhalts ein exaktes Vielfaches der Framelänge der regulären Frames ist, kann die Nachricht in einem regulären Frame enden, der die Daten enthält, gefolgt von einem letzten Frame mit einer Länge von Null (0). Oder die Nachricht kann in einem abschließenden Frame enden, der die Daten enthält. In diesem Fall hat der abschließende Frame die gleiche Frame-Länge wie die regulären Frames.
- **Kein Vielfaches der Framelänge** — Wenn die Länge des verschlüsselten Inhalts kein exaktes Vielfaches der Framelänge der regulären Frames ist, enthält der letzte Frame die verbleibenden Daten. Die Frame-Länge des abschließenden Frames ist kleiner als die Frame-Länge der regulären Frames.
- **Weniger als die Framelänge** — Wenn die Länge des verschlüsselten Inhalts kleiner als die Framelänge der regulären Frames ist, besteht die Nachricht aus einem letzten Frame, der alle Daten enthält. Die Frame-Länge des abschließenden Frames ist kleiner als die Frame-Länge der regulären Frames.

In den folgenden Tabellen sind die Felder beschrieben, die die Frames bilden. Die Byte werden in der angegebenen Reihenfolge angehängt.

Textstruktur mit Frame, regulärer Frame

Feld	Länge, in Bytes
Sequence Number	4
IV	Variable. Gleich dem im IV Length -Byte des Headers angegebenen Wert.
Encrypted Content	Variable. Gleich dem im Frame Length des Headers angegebenen Wert.
Authentication Tag	Variable. Wird durch den verwendeten Algorithmus bestimmt, wie im Algorithm ID des Headers spezifiziert.

Sequenznummer

Die Frame-Folgennummer. Dies ist ein inkrementeller Zähler für den Frame. Dies ist ein 4-Byte-Wert, interpretiert als vorzeichenlose 32-Bit-Ganzzahl.

Daten mit Frame müssen mit der Folgennummer 1 beginnen. Nachfolgende Frames müssen sich in der richtigen Reihenfolge befinden ein Inkrement von 1 gegenüber dem vorherigen Frame enthalten. Andernfalls wird der Entschlüsselungsprozess angehalten und eine Fehlermeldung ausgegeben.

IV

Der Initialisierungsvektor (IV) für den Frame. Das SDK verwendet eine deterministische Methode, um für jeden Frame in der Nachricht eine andere IV zu erstellen. Seine Länge wird durch die verwendete [Algorithmus-Folge](#) bestimmt.

Verschlüsselter Inhalt

Der verschlüsselte Inhalt (Verschlüsselungstext) für den Frame, wie vom [Verschlüsselungsalgorithmus](#) zurückgegeben.

Authentifizierungs-Tag

Die Authentifizierungswert für den Frame. Er wird verwendet, um den gesamten Frame zu authentifizieren.

Textstruktur mit Frame, abschließender Frame

Feld	Länge, in Bytes
Sequence Number End	4
Sequence Number	4
IV	Variable. Gleich dem im IV Length -Byte des Headers angegebenen Wert.
Encrypted Content Length	4
Encrypted Content	Variable. Gleich dem Wert, der in den vorherigen 4 Bytes angegeben ist (Länge des verschlüsselten Inhalts).
Authentication Tag	Variable. Wird durch den verwendeten Algorithmus bestimmt, wie im Algorithm ID des Headers spezifiziert.

Ende der Sequenznummer

Ein Indikator für den abschließenden Frame. Der Wert wird als die 4 Bytes FF FF FF FF im Hexadezimalformat kodiert.

Sequenznummer

Die Frame-Folgenummer. Dies ist ein inkrementeller Zähler für den Frame. Dies ist ein 4-Byte-Wert, interpretiert als vorzeichenlose 32-Bit-Ganzzahl.

Daten mit Frame müssen mit der Folgenummer 1 beginnen. Nachfolgende Frames müssen sich in der richtigen Reihenfolge befinden ein Inkrement von 1 gegenüber dem vorherigen Frame enthalten. Andernfalls wird der Entschlüsselungsprozess angehalten und eine Fehlermeldung ausgegeben.

IV

Der Initialisierungsvektor (IV) für den Frame. Das SDK verwendet eine deterministische Methode, um für jeden Frame in der Nachricht eine andere IV zu erstellen. Die Länge der IV-Länge wird durch die [Algorithmus-Folge angegeben](#).

Länge des verschlüsselten Inhalts

Die Länge des verschlüsselten Inhalts. Es handelt sich um einen 4-Byte-Wert, interpretiert als vorzeichenlose 32-Bit-Ganzzahl, die die Anzahl der Bytes angibt, die den verschlüsselten Inhalt für den Frame enthalten.

Verschlüsselter Inhalt

Der verschlüsselte Inhalt (Verschlüsselungstext) für den Frame, wie vom [Verschlüsselungsalgorithmus](#) zurückgegeben.

Authentifizierungs-Tag

Die Authentifizierungswert für den Frame. Er wird verwendet, um den gesamten Frame zu authentifizieren.

Footer-Struktur

Wenn die [Algorithmen mit Signatur](#) verwendet werden, enthält das Nachrichtenformat einen Footer. Die Fußzeile der Nachricht enthält eine [digitale Signatur](#), die anhand des Nachrichtenkopfs und des Nachrichtentexts berechnet wird. In der folgenden Tabelle sind die Felder des Footers beschrieben. Die Byte werden in der angegebenen Reihenfolge angehängt. Die Struktur der Nachrichtenfußzeile ist in den Nachrichtenformat-Versionen 1 und 2 dieselbe.

Footer-Struktur

Feld	Länge, in Bytes
Signature Length	2
Signature	Variable. Gleich dem Wert, der in den vorherigen 2 Bytes angegeben ist (Signaturlänge).

Länge der Signatur

Die Länge der Signatur. Es handelt sich um einen 2-Byte-Wert, interpretiert als vorzeichenlose 16-Bit-Ganzzahl, die die Anzahl der Bytes angibt, die die Signatur enthalten.

Signatur

Die Signatur

AWS Encryption SDK Beispiele für Nachrichtenformate

Die Informationen auf dieser Seite stellen eine Referenz für die Erstellung Ihrer eigenen, mit AWS Encryption SDK kompatiblen Verschlüsselungsbibliothek dar. Falls Sie keine eigene kompatible Verschlüsselungsbibliothek erstellen, benötigen Sie diese Informationen wahrscheinlich nicht.

Informationen zur Verwendung von AWS Encryption SDK in einer der unterstützten Programmiersprachen finden Sie unter [Programmiersprachen](#).

Die Spezifikation, die die Elemente einer ordnungsgemäßen AWS Encryption SDK Implementierung definiert, finden Sie in der [AWS Encryption SDK Spezifikation](#) unter GitHub.

Die folgenden Themen zeigen Beispiele für das AWS Encryption SDK Nachrichtenformat. Jedes Beispiel zeigt die Rohbytes im Hexadezimalformat, gefolgt von einer Beschreibung, wofür diese Bytes stehen.

Themen

- [Gerahmte Daten \(Nachrichtenformat, Version 1\)](#)
- [Frame-Daten \(Nachrichtenformat, Version 2\)](#)

- [Daten ohne Frames \(Nachrichtenformat, Version 1\)](#)

Gerahmte Daten (Nachrichtenformat, Version 1)

Das folgende Beispiel zeigt das Nachrichtenformat für Frame-Daten im [Nachrichtenformat Version 1](#).

```
+-----+
| Header |
+-----+
01          Version (1.0)
80          Type (128, customer authenticated encrypted
  data)
0378       Algorithm ID (see Algorithmen - Referenz)
6E7C0FBD 4DF4A999 717C22A2 DDFE1A27 Message ID (random 128-bit value)
008E       AAD Length (142)
0004       AAD Key-Value Pair Count (4)
0005       AAD Key-Value Pair 1, Key Length (5)
30746869 73 AAD Key-Value Pair 1, Key ("0This")
0002       AAD Key-Value Pair 1, Value Length (2)
6973       AAD Key-Value Pair 1, Value ("is")
0003       AAD Key-Value Pair 2, Key Length (3)
31616E     AAD Key-Value Pair 2, Key ("1an")
000A       AAD Key-Value Pair 2, Value Length (10)
656E6372 79774690 6F6E AAD Key-Value Pair 2, Value ("encryption")
0008       AAD Key-Value Pair 3, Key Length (8)
32636F6E 74657874 AAD Key-Value Pair 3, Key ("2context")
0007       AAD Key-Value Pair 3, Value Length (7)
6578616D 706C65 AAD Key-Value Pair 3, Value ("example")
0015       AAD Key-Value Pair 4, Key Length (21)
6177732D 63727970 746F2D70 75626C69 AAD Key-Value Pair 4, Key ("aws-crypto-
public-key")
632D6B65 79
0044       AAD Key-Value Pair 4, Value Length (68)
416A4173 7569326F 7430364C 4B77715A AAD Key-Value Pair 4, Value
  ("AjAsui2ot06LKwqZXDJnU/Aqc2vD+00kp0Z1cc8Tg2qd7rs5aLTg71vfUEW/86+/5w==")
58444A6E 552F4171 63327644 2B304F6B
704F5A31 63633854 67327164 37727335
614C5467 376C7666 5545572F 38362B2F
35773D3D
0002       EncryptedDataKeyCount (2)
0007       Encrypted Data Key 1, Key Provider ID Length
  (7)
```


6177732D 6B6D73	Encrypted Data Key 1, Key Provider ID ("aws-
kms")	
004B	Encrypted Data Key 1, Key Provider
Information Length (75)	
61726E3A 6177733A 6B6D733A 75732D77	Encrypted Data Key 1, Key Provider
Information ("arn:aws:kms:us-west-2:111122223333:key/715c0818-5825-4245-	
a755-138a6d9a11e6")	
6573742D 323A3131 31313232 32323333	
33333A6B 65792F37 31356330 3831382D	
35383235 2D343234 352D6137 35352D31	
33386136 64396131 316536	
00A7	Encrypted Data Key 1, Encrypted Data Key
Length (167)	
01010200 7857A1C1 F7370545 4ECA7C83	Encrypted Data Key 1, Encrypted Data Key
956C4702 23DCE8D7 16C59679 973E3CED	
02A4EF29 7F000000 7E307C06 092A8648	
86F70D01 0706A06F 306D0201 00306806	
092A8648 86F70D01 0701301E 06096086	
48016503 04012E30 11040C3F F02C897B	
7A12EB19 8BF2D802 0110803B 24003D1F	
A5474FBC 392360B5 CB9997E0 6A17DE4C	
A6BD7332 6BF86DAB 60D8CCB8 8295DBE9	
4707E356 ADA3735A 7C52D778 B3135A47	
9F224BF9 E67E87	
0007	Encrypted Data Key 2, Key Provider ID Length
(7)	
6177732D 6B6D73	Encrypted Data Key 2, Key Provider ID ("aws-
kms")	
004E	Encrypted Data Key 2, Key Provider
Information Length (78)	
61726E3A 6177733A 6B6D733A 63612D63	Encrypted Data Key 2, Key Provider
Information ("arn:aws:kms:ca-central-1:111122223333:key/9b13ca4b-afcc-46a8-aa47-	
be3435b423ff")	
656E7472 616C2D31 3A313131 31323232	
32333333 333A6B65 792F3962 31336361	
34622D61 6663632D 34366138 2D616134	
372D6265 33343335 62343233 6666	
00A7	Encrypted Data Key 2, Encrypted Data Key
Length (167)	
01010200 78FAFFFB D6DE06AF AC72F79B	Encrypted Data Key 2, Encrypted Data Key
0E57BD87 3F60F4E6 FD196144 5A002C94	
AF787150 69000000 7E307C06 092A8648	
86F70D01 0706A06F 306D0201 00306806	
092A8648 86F70D01 0701301E 06096086	

```

48016503 04012E30 11040C36 CD985E12
D218B674 5BBC6102 0110803B 0320E3CD
E470AA27 DEAB660B 3E0CE8E0 8B1A89E4
57DCC69B AAB1294F 21202C01 9A50D323
72EBAAFD E24E3ED8 7168E0FA DB40508F
556FBD58 9E621C
02
00000000
0C
00000100
4ECBD5C0 9899CA65 923D2347
0B896144 0CA27950 CA571201 4DA58029
+-----+
| Body |
+-----+
00000001
6BD3FE9C ADBC213 5B89E8F1
1F6471E0 A51AF310 10FA9EF6 F0C76EDF
F5AFA33C 7D2E8C6C 9C5D5175 A212AF8E
FBD9A0C3 C6E3FB59 C125DBF2 89AC7939
BDEE43A8 0F00F49E ACBB8B2 1C785089
A90DB923 699A1495 C3B31B50 0A48A830
201E3AD9 1EA6DA14 7F6496DB 6BC104A4
DEB7F372 375ECB28 9BF84B6D 2863889F
CB80A167 9C361C4B 5EC07438 7A4822B4
A7D9D2CC 5150D414 AF75F509 FCE118BD
6D1E798B AEBA4CDB AD009E5F 1A571B77
0041BC78 3E5F2F41 8AF157FD 461E959A
BB732F27 D83DC36D CC9EBC05 00D87803
57F2BB80 066971C2 DEEA062F 4F36255D
E866C042 E1382369 12E9926B BA40E2FC
A820055F FB47E428 41876F14 3B6261D9
5262DB34 59F5D37E 76E46522 E8213640
04EE3CC5 379732B5 F56751FA 8E5F26AD
00000002
F1140984 FF25F943 959BE514
216C7C6A 2234F395 F0D2D9B9 304670BF
A1042608 8A8BCB3F B58CF384 D72EC004
A41455B4 9A78BAC9 36E54E68 2709B7BD
A884C1E1 705FF696 E540D297 446A8285
23DFEE28 E74B225A 732F2C0C 27C6BDA2
7597C901 65EF3502 546575D4 6D5EBF22
1FF787AB 2E38FD77 125D129C 43D44B96
778D7CEE 3C36625F FF3A985C 76F7D320

```

Content Type (2, framed data)
Reserved
IV Length (12)
Frame Length (256)
IV
Authentication Tag

Frame 1, Sequence Number (1)
Frame 1, IV
Frame 1, Encrypted Content

Frame 1, Authentication Tag
Frame 2, Sequence Number (2)
Frame 2, IV
Frame 2, Encrypted Content

ED70B1F3 79729B47 E7D9B5FC 02FCE9F5	
C8760D55 7779520A 81D54F9B EC45219D	
95941F7E 5CBAEAC8 CEC13B62 1464757D	
AC65B6EF 08262D74 44670624 A3657F7F	
2A57F1FD E7060503 AC37E197 2F297A84	
DF1172C2 FA63CF54 E6E2B9B6 A86F582B	
3B16F868 1BBC5E4D 0B6919B3 08D5ABCF	
FECDC4A4 8577F08B 99D766A1 E5545670	
A61F0A3B A3E45A84 4D151493 63ECA38F	
FFFFFFFF	Frame 2, Authentication Tag
00000003	Final Frame, Sequence Number End
35F74F11 25410F01 DD9E04BF	Final Frame, Sequence Number (3)
0000008E	Final Frame, IV
F7A53D37 2F467237 6FBD0B57 D1DFE830	Final Frame, Encrypted Content Length (142)
B965AD1F A910AA5F 5EFFFFFF4 BC7D431C	Final Frame, Encrypted Content
BA9FA7C4 B25AF82E 64A04E3A A0915526	
88859500 7096FABB 3ACAD32A 75CFED0C	
4A4E52A3 8E41484D 270B7A0F ED61810C	
3A043180 DF25E5C5 3676E449 0986557F	
C051AD55 A437F6BC 139E9E55 6199FD60	
6ADC017D BA41CDA4 C9F17A83 3823F9EC	
B66B6A5A 80FDB433 8A48D6A4 21CB	
811234FD 8D589683 51F6F39A 040B3E3B	
+-----+	
Footer	
+-----+	
0066	Final Frame, Authentication Tag
30640230 085C1D3C 63424E15 B2244448	Signature Length (102)
639AED00 F7624854 F8CF2203 D7198A28	Signature
758B309F 5EFD9D5D 2E07AD0B 467B8317	
5208B133 02301DF7 2DFC877A 66838028	
3C6A7D5E 4F8B894E 83D98E7C E350F424	
7E06808D 0FE79002 E24422B9 98A0D130	
A13762FF 844D	

Frame-Daten (Nachrichtenformat, Version 2)

Das folgende Beispiel zeigt das Nachrichtenformat für Frame-Daten im [Nachrichtenformat Version 2](#).

+-----+	
Header	
+-----+	
02	Version (2.0)

0578	Algorithm ID (see Algorithms reference)
122747eb 21dfe39b 38631c61 7fad7340	
cc621a30 32a11cc3 216d0204 fd148459	Message ID (random 256-bit value)
008e	AAD Length (142)
0004	AAD Key-Value Pair Count (4)
0005	AAD Key-Value Pair 1, Key Length (5)
30546869 73	AAD Key-Value Pair 1, Key ("0This")
0002	AAD Key-Value Pair 1, Value Length (2)
6973	AAD Key-Value Pair 1, Value ("is")
0003	AAD Key-Value Pair 2, Key Length (3)
31616e	AAD Key-Value Pair 2, Key ("1an")
000a	AAD Key-Value Pair 2, Value Length (10)
656e6372 79707469 6f6e	AAD Key-Value Pair 2, Value ("encryption")
0008	AAD Key-Value Pair 3, Key Length (8)
32636f6e 74657874	AAD Key-Value Pair 3, Key ("2context")
0007	AAD Key-Value Pair 3, Value Length (7)
6578616d 706c65	AAD Key-Value Pair 3, Value ("example")
0015	AAD Key-Value Pair 4, Key Length (21)
6177732d 63727970 746f2d70 75626c69	AAD Key-Value Pair 4, Key ("aws-crypto-
public-key")	
632d6b65 79	
0044	AAD Key-Value Pair 4, Value Length (68)
41746733 72703845 41345161 36706669	AAD Key-Value Pair 4, Value
("QXRnM3JwOEVBnFFhNnBmaTk3MULtNTk3NHp0MnlZWE5vSmtwRHFPc0dIYkVaVDRqME50MLFkRStmbTFVY01WdThnPT0=	
39373149 53353937 347a4e32 7959584e	
6f4a6b70 44714f73 47486245 5a54346a	
304e4e32 5164452b 666d3155 634d5675	
38673d3d	
0001	Encrypted Data Key Count (1)
0007	Encrypted Data Key 1, Key Provider ID Length
(7)	
6177732d 6b6d73	Encrypted Data Key 1, Key Provider ID ("aws-
kms")	
004b	Encrypted Data Key 1, Key Provider
Information Length (75)	
61726e3a 6177733a 6b6d733a 75732d77	Encrypted Data Key 1, Key
Provider Information ("arn:aws:kms:us-west-2:658956600833:key/b3537ef1-	
d8dc-4780-9f5a-55776cbb2f7f")	
6573742d 323a3635 38393536 36303038	
33333a6b 65792f62 33353337 6566312d	
64386463 2d343738 302d3966 35612d35	
35373736 63626232 663766	
00a7	Encrypted Data Key 1, Encrypted Data Key
Length (167)	

01010100 7840f38c 275e3109 7416c107	Encrypted Data Key 1, Encrypted Data Key
29515057 1964ada3 ef1c21e9 4c8ba0bd	
bc9d0fb4 14000000 7e307c06 092a8648	
86f70d01 0706a06f 306d0201 00306806	
092a8648 86f70d01 0701301e 06096086	
48016503 04012e30 11040c39 32d75294	
06063803 f8460802 0110803b 2a46bc23	
413196d2 903bf1d7 3ed98fc8 a94ac6ed	
e00ee216 74ec1349 12777577 7fa052a5	
ba62e9e4 f2ac8df6 bcb1758f 2ce0fb21	
cc9ee5c9 7203bb	
02	Content Type (2, framed data)
00001000	Frame Length (4096)
05cd035b 29d5499d 4587570b 87502afe	Algorithm Suite Data (key commitment)
634f7b2c c3df2aa9 88a10105 4a2c7687	
76cb339f 2536741f 59a1c202 4f2594ab	Authentication Tag
+-----+	
Body	
+-----+	
ffffffff	Final Frame, Sequence Number End
00000001	Final Frame, Sequence Number (1)
00000000 00000000 00000001	Final Frame, IV
00000009	Final Frame, Encrypted Content Length (9)
fa6e39c6 02927399 3e	Final Frame, Encrypted Content
f683a564 405d68db eeb0656c d57c9eb0	Final Frame, Authentication Tag
+-----+	
Footer	
+-----+	
0067	Signature Length (103)
30650230 2a1647ad 98867925 c1712e8f	Signature
ade70b3f 2a2bc3b8 50eb91ef 56cfdd18	
967d91d8 42d92baf 357bba48 f636c7a0	
869cade2 023100aa ae12d08f 8a0afe85	
e5054803 110c9ed8 11b2e08a c4a052a9	
074217ea 3b01b660 534ac921 bf091d12	
3657e2b0 9368bd	

Daten ohne Frames (Nachrichtenformat, Version 1)

Das folgende Beispiel zeigt das Nachrichtenformat für Daten ohne Frame.

Note

Verwenden Sie nach Möglichkeit gerahmte Daten. Das AWS Encryption SDK unterstützt Daten ohne Frames nur für die Verwendung in älteren Versionen. Einige Sprachimplementierungen von AWS Encryption SDK können immer noch nicht gerahmten Chiffretext generieren. Alle unterstützten Sprachimplementierungen können gerahmten und ungerahmten Chiffretext entschlüsseln.

```
+-----+
| Header |
+-----+
01          Version (1.0)
80          Type (128, customer authenticated encrypted
  data)
0378       Algorithm ID (see Algorithmen - Referenz)
B8929B01 753D4A45 C0217F39 404F70FF Message ID (random 128-bit value)
008E       AAD Length (142)
0004       AAD Key-Value Pair Count (4)
0005       AAD Key-Value Pair 1, Key Length (5)
30746869 73  AAD Key-Value Pair 1, Key ("0This")
0002       AAD Key-Value Pair 1, Value Length (2)
6973       AAD Key-Value Pair 1, Value ("is")
0003       AAD Key-Value Pair 2, Key Length (3)
31616E     AAD Key-Value Pair 2, Key ("1an")
000A       AAD Key-Value Pair 2, Value Length (10)
656E6372 79774690 6F6E AAD Key-Value Pair 2, Value ("encryption")
0008       AAD Key-Value Pair 3, Key Length (8)
32636F6E 74657874 AAD Key-Value Pair 3, Key ("2context")
0007       AAD Key-Value Pair 3, Value Length (7)
6578616D 706C65 AAD Key-Value Pair 3, Value ("example")
0015       AAD Key-Value Pair 4, Key Length (21)
6177732D 63727970 746F2D70 75626C69 AAD Key-Value Pair 4, Key ("aws-crypto-
public-key")
632D6B65 79
0044       AAD Key-Value Pair 4, Value Length (68)
41734738 67473949 6E4C5075 3136594B AAD Key-Value Pair 4, Value
("AsG8gG9InLPu16YK1qXT0D+nykG8YqHAhqcj8aXfD2e5B4gtVE73dZkyClA+rAM0Q==")
6C715854 4F442B6E 796B4738 59714841
68716563 6A386158 66443265 35423467
74564537 33645A6B 79436C41 2B72414D
4F513D3D
```

0002	Encrypted Data Key Count (2)
0007	Encrypted Data Key 1, Key Provider ID Length
(7)	
6177732D 6B6D73	Encrypted Data Key 1, Key Provider ID ("aws-kms")
004B	Encrypted Data Key 1, Key Provider
Information Length (75)	
61726E3A 6177733A 6B6D733A 75732D77	Encrypted Data Key 1, Key Provider
Information ("arn:aws:kms:us-west-2:111122223333:key/715c0818-5825-4245-a755-138a6d9a11e6")	
6573742D 323A3131 31313232 32323333	
33333A6B 65792F37 31356330 3831382D	
35383235 2D343234 352D6137 35352D31	
33386136 64396131 316536	
00A7	Encrypted Data Key 1, Encrypted Data Key
Length (167)	
01010200 7857A1C1 F7370545 4ECA7C83	Encrypted Data Key 1, Encrypted Data Key
956C4702 23DCE8D7 16C59679 973E3CED	
02A4EF29 7F000000 7E307C06 092A8648	
86F70D01 0706A06F 306D0201 00306806	
092A8648 86F70D01 0701301E 06096086	
48016503 04012E30 11040C28 4116449A	
0F2A0383 659EF802 0110803B B23A8133	
3A33605C 48840656 C38BCB1F 9CCE7369	
E9A33EBE 33F46461 0591FECA 947262F3	
418E1151 21311A75 E575ECC5 61A286E0	
3E2DEBD5 CB005D	
0007	Encrypted Data Key 2, Key Provider ID Length
(7)	
6177732D 6B6D73	Encrypted Data Key 2, Key Provider ID ("aws-kms")
004E	Encrypted Data Key 2, Key Provider
Information Length (78)	
61726E3A 6177733A 6B6D733A 63612D63	Encrypted Data Key 2, Key Provider
Information ("arn:aws:kms:ca-central-1:111122223333:key/9b13ca4b-afcc-46a8-aa47-be3435b423ff")	
656E7472 616C2D31 3A313131 31323232	
32333333 333A6B65 792F3962 31336361	
34622D61 6663632D 34366138 2D616134	
372D6265 33343335 62343233 6666	
00A7	Encrypted Data Key 2, Encrypted Data Key
Length (167)	
01010200 78FAFFFB D6DE06AF AC72F79B	Encrypted Data Key 2, Encrypted Data Key
0E57BD87 3F60F4E6 FD196144 5A002C94	

```

AF787150 69000000 7E307C06 092A8648
86F70D01 0706A06F 306D0201 00306806
092A8648 86F70D01 0701301E 06096086
48016503 04012E30 11040CB2 A820D0CC
76616EF2 A6B30D02 0110803B 8073D0F1
FDD01BD9 B0979082 099FDBFC F7B13548
3CC686D7 F3CF7C7A CCC52639 122A1495
71F18A46 80E2C43F A34C0E58 11D05114
2A363C2A E11397
01
00000000
0C
00000000
734C1BBE 032F7025 84CDA9D0
2C82BB23 4CBF4AAB 8F5C6002 622E886C
+-----+
| Body |
+-----+
D39DD3E5 915E0201 77A4AB11
00000000 0000028E
E8B6F955 B5F22FE4 FD890224 4E1D5155
5871BA4C 93F78436 1085E4F8 D61ECE28
59455BD8 D76479DF C28D2E0B BDB3D5D3
E4159DFE C8A944B6 685643FC EA24122B
6766ECD5 E3F54653 DF205D30 0081D2D8
55FCDA5B 9F5318BC F4265B06 2FE7C741
C7D75BCC 10F05EA5 0E2F2F40 47A60344
ECE10AA7 559AF633 9DE2C21B 12AC8087
95FE9C58 C65329D1 377C4CD7 EA103EC1
31E4F48A 9B1CC047 EE5A0719 704211E5
B48A2068 8060DF60 B492A737 21B0DB21
C9B21A10 371E6179 78FAFB0B BAAEC3F4
9D86E334 701E1442 EA5DA288 64485077
54C0C231 AD43571A B9071925 609A4E59
B8178484 7EB73A4F AAE46B26 F5B374B8
12B0000C 8429F504 936B2492 AAF47E94
A5BA804F 7F190927 5D2DF651 B59D4C2F
A15D0551 DAEB44AF 2060D0D5 CB1DA4E6
5E2034DB 4D19E7CD EEA6CF7E 549C86AC
46B2C979 AB84EE12 202FD6DF E7E3C09F
C2394012 AF20A97E 369BCBDA 62459D3E
C6FFB914 FEFD4DE5 88F5AFE1 98488557
1BABBAE4 BE55325E 4FB7E602 C1C04BEE
F3CB6B86 71666C06 6BF74E1B 0F881F31

```

Content Type (1, nonframed data)
Reserved
IV Length (12)
Frame Length (0, nonframed data)
IV
Authentication Tag

IV
Encrypted Content Length (654)
Encrypted Content


```

B731839B CF711F6A 84CA95F5 958D3B44
E3862DF6 338E02B5 C345CFF8 A31D54F3
6920AA76 0BF8E903 552C5A04 917CCD11
D4E5DF5C 491EE86B 20C33FE1 5D21F0AD
6932E67C C64B3A26 B8988B25 CFA33E2B
63490741 3AB79D60 D8AEFBE9 2F48E25A
978A019C FE49EE0A 0E96BF0D D6074DDB
66DFF333 0E10226F 0A1B219C BE54E4C2
2C15100C 6A2AA3F1 88251874 FDC94F6B
9247EF61 3E7B7E0D 29F3AD89 FA14A29C
76E08E9B 9ADCDF8C C886D4FD A69F6CB4
E24FDE26 3044C856 BF08F051 1ADAD329
C4A46A1E B5AB72FE 096041F1 F3F3571B
2EAFD9CB B9EB8B83 AE05885A 8F2D2793
1E3305D9 0C9E2294 E8AD7E3B 8E4DEC96
6276C5F1 A3B7E51E 422D365D E4C0259C
50715406 822D1682 80B0F2E5 5C94
65B2E942 24BEEA6E A513F918 CCEC1DE3      Authentication Tag
+-----+
| Footer |
+-----+
0067      Signature Length (103)
30650230 7229DDF5 B86A5B64 54E4D627      Signature
CBE194F1 1CC0F8CF D27B7F8B F50658C0
BE84B355 3CED1721 A0BE2A1B 8E3F449E
1BEB8281 023100B2 0CB323EF 58A4ACE3
1559963B 889F72C3 B15D1700 5FB26E61
331F3614 BC407CEE B86A66FA CBF74D9E
34CB7E4B 363A38

```

Hauptteil der Referenz für zusätzliche authentifizierte Daten () für AAD AWS Encryption SDK

Die Informationen auf dieser Seite stellen eine Referenz für die Erstellung Ihrer eigenen, mit AWS Encryption SDK kompatiblen Verschlüsselungsbibliothek dar. Falls Sie keine eigene kompatible Verschlüsselungsbibliothek erstellen, benötigen Sie diese Informationen wahrscheinlich nicht.

Informationen zur Verwendung von AWS Encryption SDK in einer der unterstützten Programmiersprachen finden Sie unter [Programmiersprachen](#).

Die Spezifikation, die die Elemente einer ordnungsgemäßen AWS Encryption SDK Implementierung definiert, finden Sie in der [AWS Encryption SDK Spezifikation](#) unter GitHub.

Für jeden kryptografischen Vorgang müssen Sie dem [GCMAlgorithmus AES](#) - zusätzliche authentifizierte Daten (AAD) zur Verfügung stellen. Dies gilt für [Textdaten](#) mit und ohne Frame. Weitere Informationen zu AAD und zu seiner Verwendung im Galois/Counter Mode (GCM) finden Sie unter [Empfehlungen für Blockchiffriermodi: Galois/Counter Mode \(\) und. GCM GMAC](#)

In der folgenden Tabelle werden die Felder beschrieben, die den Hauptteil bilden. AAD Die Byte werden in der angegebenen Reihenfolge angehängt.

AADStruktur des Körpers

Feld	Länge, in Bytes
Message ID	16
Body AAD Content	Variable. Siehe AADHauptinhalt in der folgenden Liste.
Sequence Number	4
Content Length	8

Nachrichten-ID

Der gleiche [Message ID](#)-Wert, wie der im Nachrichten-Header festgelegte Wert.

AADInhalt des Hauptteils

Ein mit UTF -8 codierter Wert, der durch die Art der verwendeten Körperdaten bestimmt wird.

Für [Daten ohne Frame](#) verwenden Sie den Wert `AWSKMSEncryptionClient Single Block`.

Für reguläre Frames in [Daten mit Frame](#) verwenden Sie den Wert `AWSKMSEncryptionClient Frame`.

Für den abschließenden Frame in [Daten mit Frame](#) verwenden Sie den Wert `AWSKMSEncryptionClient Final Frame`.

Sequenznummer

Ein 4-Byte-Wert, interpretiert als vorzeichenlose 32-Bit-Ganzzahl.

Für [Daten mit Frame](#) ist dies die Frame-Folgenummer.

Für [Daten ohne Frame](#) verwenden Sie den Wert 1, kodiert als die 4 Bytes 00 00 00 01 im Hexadezimalformat.

Länge des Inhalts

Die Länge der Klartextdaten in Bytes, die dem Algorithmus zur Verschlüsselung zur Verfügung gestellt werden. Dies ist ein 8-Byte-Wert, interpretiert als vorzeichenlose 64-Bit-Ganzzahl.

AWS Encryption SDK Referenz zu Algorithmen

Die Informationen auf dieser Seite stellen eine Referenz für die Erstellung Ihrer eigenen, mit AWS Encryption SDK kompatiblen Verschlüsselungsbibliothek dar. Falls Sie keine eigene kompatible Verschlüsselungsbibliothek erstellen, benötigen Sie diese Informationen wahrscheinlich nicht.

Informationen zur Verwendung von AWS Encryption SDK in einer der unterstützten Programmiersprachen finden Sie unter [Programmiersprachen](#).

Die Spezifikation, die die Elemente einer ordnungsgemäßen AWS Encryption SDK Implementierung definiert, finden Sie in der [AWS Encryption SDK Spezifikation](#) unter GitHub.

Wenn Sie Ihre eigene Bibliothek erstellen, die Chiffretexte lesen und schreiben kann, die mit dem kompatibel sind AWS Encryption SDK, müssen Sie verstehen, wie die unterstützten Algorithmus-Suites zur Verschlüsselung von Rohdaten AWS Encryption SDK implementiert.

Die AWS Encryption SDK unterstützt die folgenden Algorithmus-Suiten. Alle AES GCM Algorithmus-Suiten haben einen [12-Byte-Initialisierungsvektor](#) und ein AES GCM 16-Byte-Authentifizierungs-Tag. Die Standard-Algorithmus-Suite variiert je nach AWS Encryption SDK Version und ausgewählter Key Commitment Policy. Einzelheiten finden Sie unter [Commitment-Richtlinie und Algorithmus-Suite](#).

AWS Encryption SDK Algorithmus-Suiten

Algorithmus-ID	Version im Nachrichtenformat	Verschlüsselungsalgorithmus	Länge des Datenschlüssels (Bits)	Schlüsselableitungsalgorithmus	Signaturalgorithmus	Algorithmus für Schlüsselzusagen	Datenlänge der Algorithm Suite (Byte)
05 78	0x02	AES-GCM	256	HKDFmit SHA -512	ECDSAmith P-384 und -384 SHA	HKDFmit -512 SHA	32 (zentrale Verpflichtung)
04 78	0x02	AES-GCM	256	HKDFmit SHA -512	None	HKDFmit SHA -512	32 (zentrale Verpflichtung)
03 78	0x01	AES-GCM	256	HKDFmit SHA -384	ECDSAmith P-384 und -384 SHA	None	N/A
03 46	0x01	AES-GCM	192	HKDFmit -384 SHA	ECDSAmith P-384 und -384 SHA	None	N/A
02 14	0x01	AES-GCM	128	HKDFmit -256 SHA	ECDSAmith P-256 und -256 SHA	None	N/A
01 78	0x01	AES-GCM	256	HKDFmit -256 SHA	Keine	None	N/A
01 46	0x01	AES-GCM	192	HKDFmit SHA -256	Keine	None	N/A

Algorithmus-ID	Version im Nachrichtenformat	Verschlüsselungsalgorithmus	Länge des Datenschlüssels (Bits)	Schlüsselableitungsalgorithmus	Signaturalgorithmus	Algorithmus für Schlüsselzusagen	Datenlänge der Algorithm Suite (Byte)
01 14	0x01	AES-GCM	128	HKDFmit SHA -256	Keine	None	N/A
00 78	0x01	AES-GCM	256	Keine	Keine	None	N/A
00 46	0x01	AES-GCM	192	Keine	Keine	None	N/A
00 14	0x01	AES-GCM	128	Keine	Keine	None	N/A

Algorithmus-ID

Ein 2-Byte-Hexadezimalwert, der eine Algorithmusimplementierung eindeutig identifiziert. Dieser Wert wird im [Nachrichtenkopf des Chiffretextes](#) gespeichert.

Version im Nachrichtenformat

Die Version des Nachrichtenformats. Algorithmus-Suites mit Key Commitment verwenden das Nachrichtenformat Version 2 (0x02). Algorithmus-Suites ohne Schlüsselzusage verwenden das Nachrichtenformat Version 1 (0x01).

Datenlänge der Algorithmus-Suite

Die Länge der für die Algorithmus-Suite spezifischen Daten in Byte. Dieses Feld wird nur im Nachrichtenformat Version 2 (0x02) unterstützt. Im Nachrichtenformat Version 2 (0x02) werden diese Daten im `Algorithm suite data` Feld des Nachrichtenkopfs angezeigt. Algorithmus-Suites, die [Key Commitment](#) unterstützen, verwenden 32 Byte für die Key-Commitment-Zeichenfolge. Weitere Informationen finden Sie in dieser Liste unter Key Commitment-Algorithmus.

Länge des Datenschlüssels

Die Länge des [Datenschlüssels](#) in Bits. Der AWS Encryption SDK unterstützt 256-Bit-, 192-Bit- und 128-Bit-Schlüssel. [Der Datenschlüssel wird durch einen Schlüsselbund oder Hauptschlüssel generiert.](#)

In einigen Implementierungen wird dieser Datenschlüssel als Eingabe für eine HMAC basierte extract-and-expand Schlüsselableitungsfunktion () verwendet. HKDF Die Ausgabe von HKDF wird als Datenverschlüsselungsschlüssel im Verschlüsselungsalgorithmus verwendet. Weitere Informationen finden Sie in dieser Liste unter Algorithmus zur Schlüsselableitung.

Verschlüsselungsalgorithmus

Der Name und der Modus des verwendeten Verschlüsselungsalgorithmus. Algorithmus-Suiten AWS Encryption SDK verwenden den Advanced Encryption Standard (AES) - Verschlüsselungsalgorithmus mit Galois/Counter-Modus (). GCM

Algorithmus für Schlüsselzusagen

Der Algorithmus, der zur Berechnung der Key-Commitment-Zeichenfolge verwendet wurde. Die Ausgabe wird im `Algorithm suite data` Feld des Nachrichtenkopfs gespeichert und dient zur Validierung des Datenschlüssels für Key Commitment.

Eine technische Erläuterung des Hinzufügens von Key Commitment zu einer Algorithmus-Suite finden Sie unter [Key Committing AEADs](#) in Cryptology Archive ePrint .

Schlüsselableitungsalgorithmus

Die HMAC basierte extract-and-expand Schlüsselableitungsfunktion (HKDF), die zur Ableitung des Datenverschlüsselungsschlüssels verwendet wird. [Die AWS Encryption SDK verwendet die in HKDF RFC 5869 definierte.](#)

Algorithmus-Suiten ohne Schlüsselbindung (Algorithmus-ID 01xx —03xx)

- Die verwendete Hash-Funktion ist entweder SHA -384 oder SHA -256, abhängig von der Algorithmus-Suite.
- Für den Extraktionsschritt:
 - Es wird kein Salt verwendet. Gemäß dem ist RFC das Salz auf eine Folge von Nullen gesetzt. Die Länge der Zeichenfolge entspricht der Länge der Ausgabe der Hash-Funktion, die 48 Byte für SHA -384 und 32 Byte für SHA -256 beträgt.
 - Das Eingabematerial ist der Datenschlüssel aus dem Schlüsselbund oder dem Hauptschlüsselanbieter.

- Für den Expansionsschritt:
 - Der pseudozufällige Eingabeschlüssel ist die Ausgabe aus dem Extraktionsschritt.
 - Die Eingabeinformationen sind eine Verkettung der Algorithmus-ID und der Nachrichten-ID (in dieser Reihenfolge).
 - Die Länge des Ausgabe-Keying-Materials entspricht der Länge des Datenschlüssels. Diese Ausgabe wird als Datenverschlüsselungsschlüssel im Verschlüsselungsalgorithmus verwendet.

Algorithmus-Suiten mit Schlüsselbindung (Algorithmus-ID 04xx und 05xx)

- Die verwendete Hash-Funktion ist SHA -512.
- Für den Extraktionsschritt:
 - Der Salt ist ein kryptografischer 256-Bit-Zufallswert. Im [Nachrichtenformat Version 2](#) (0x02) wird dieser Wert im Feld gespeichert. MessageID
 - Bei der ersten Eingabe handelt es sich um den Datenschlüssel aus dem Schlüsselbund oder dem Hauptschlüsselanbieter.
- Für den Expansionsschritt:
 - Der pseudozufällige Eingabeschlüssel ist die Ausgabe aus dem Extraktionsschritt.
 - Die Schlüsselbezeichnung besteht aus den UTF -8-kodierten Bytes der DERIVEKEY Zeichenfolge in Big-Endian-Byte-Reihenfolge.
 - Die Eingabeinformationen sind eine Verkettung der Algorithmus-ID und der Schlüsselbezeichnung (in dieser Reihenfolge).
 - Die Länge des Ausgabe-Keying-Materials entspricht der Länge des Datenschlüssels. Diese Ausgabe wird als Datenverschlüsselungsschlüssel im Verschlüsselungsalgorithmus verwendet.

Version im Nachrichtenformat

Die Version des Nachrichtenformats, das mit der Algorithmensuite verwendet wird. Details hierzu finden Sie unter [Nachrichtenformat – Referenz](#).

Signatur-Algorithmus

Der Signaturalgorithmus, der verwendet wird, um eine [digitale Signatur](#) über dem Chiffretext-Header und dem Hauptteil zu generieren. Der AWS Encryption SDK verwendet den Algorithmus für digitale Signaturen mit elliptischen Kurven (ECDSA) mit den folgenden Besonderheiten:

- Die verwendete elliptische Kurve die P-384- oder P-256-Kurve, wie durch die Algorithmus-ID angegeben. Diese Kurven sind im [Digital Signature Standard \(DSS\) \(FIPSPUB186-4\)](#) definiert.

- Die verwendete Hash-Funktion ist SHA -384 (mit der P-384-Kurve) oder SHA -256 (mit der P-256-Kurve).

AWS Encryption SDK Referenz auf den Initialisierungsvektor

Die Informationen auf dieser Seite stellen eine Referenz für die Erstellung Ihrer eigenen, mit AWS Encryption SDK kompatiblen Verschlüsselungsbibliothek dar. Falls Sie keine eigene kompatible Verschlüsselungsbibliothek erstellen, benötigen Sie diese Informationen wahrscheinlich nicht.

Informationen zur Verwendung von AWS Encryption SDK in einer der unterstützten Programmiersprachen finden Sie unter [Programmiersprachen](#).

Die Spezifikation, die die Elemente einer ordnungsgemäßen AWS Encryption SDK Implementierung definiert, finden Sie in der [AWS Encryption SDK Spezifikation](#) unter GitHub.

Der AWS Encryption SDK stellt die [Initialisierungsvektoren](#) (IVs) bereit, die von allen unterstützten [Algorithmus-Suiten](#) benötigt werden. Der SDK verwendet Frame-Sequenznummern, um eine IV zu erstellen, sodass keine zwei Frames in derselben Nachricht dieselbe IV haben können.

Jeder 96-Bit- (12-Byte-)IV besteht aus zwei Big-Endian-Byte-Arrays, die in der folgenden Reihenfolge verkettet sind:

- 64 Bits: 0 (für eine zukünftige Nutzung reserviert)
- 32 Bits: Frame-Folgenummer. Für das Header-Authentifizierungs-Tag besteht dieser Wert aus lauter Nullen.

Vor der Einführung der [Zwischenspeicherung von Datenschlüsseln](#) verwendeten sie AWS Encryption SDK immer einen neuen Datenschlüssel, um jede Nachricht zu verschlüsseln, und alle IVs wurden nach dem Zufallsprinzip generiert. Zufällig generierte Schlüssel IVs waren kryptografisch sicher, da Datenschlüssel nie wiederverwendet wurden. Als das Zwischenspeichern von Datenschlüsseln SDK eingeführt wurde, bei dem Datenschlüssel bewusst wiederverwendet werden, haben wir die Art der Generierung geändert. SDK IVs

Die Verwendung von deterministischen IVs Methoden, die sich innerhalb einer Nachricht nicht wiederholen können, erhöht die Anzahl der Aufrufe, die sicher unter einem einzigen Datenschlüssel ausgeführt werden können, erheblich. Darüber hinaus verwenden im Cache gespeichert

Datenschlüssel immer einen Algorithmus mit einer [Schlüsselableitungsfunktion](#). Die Verwendung einer deterministischen IV mit einer Funktion zur Ableitung von Pseudozufallsschlüsseln zur Ableitung von Verschlüsselungsschlüsseln aus einem Datenschlüssel ermöglicht die Verschlüsselung von 2^{32} Nachrichten, ohne die kryptografischen Grenzen AWS Encryption SDK zu überschreiten.

AWS KMS Technische Details zum hierarchischen Schlüsselbund

Der [AWS KMS hierarchische Schlüsselbund](#) verwendet einen eindeutigen Datenschlüssel, um jede Nachricht zu verschlüsseln, und verschlüsselt jeden Datenschlüssel mit einem eindeutigen Umschließungsschlüssel, der von einem aktiven Zweigschlüssel abgeleitet wird. Er verwendet eine [Schlüsselableitung](#) im Zählermodus mit einer Pseudozufallsfunktion mit HMAC SHA -256, um den 32-Byte-Wrapping-Schlüssel mit den folgenden Eingaben abzuleiten.

- Ein 16-Byte-Zufallssalz
- Der aktive Zweigschlüssel
- Der mit [UTF-8 kodierte](#) Wert für die Schlüsselanbieter-ID "" aws-kms-hierarchy

Der hierarchische Schlüsselbund verwendet den abgeleiteten Umschließungsschlüssel, um eine Kopie des Klartext-Datenschlüssels mithilfe von AES - GCM -256 mit einem 16-Byte-Authentifizierungs-Tag und den folgenden Eingaben zu verschlüsseln.

- Der abgeleitete Umschließungsschlüssel wird als Chiffrierschlüssel mit dem Buchstaben - verwendet AES GCM
- Der Datenschlüssel wird als -Nachricht verwendet AES GCM
- Ein zufälliger 12-Byte-Initialisierungsvektor (IV) wird als AES - GCM IV verwendet
- Zusätzliche authentifizierte Daten (AAD), die die folgenden serialisierten Werte enthalten.

Wert	Länge in Byte	Interpretiert als
"aws-kms-hierarchy"	17	UTF-8 codiert
Die Kennung des Zweigschlüssels	Variable	UTF-8 codiert
Die Version mit dem Zweigschlüssel	16	UTF-8 codiert

Wert	Länge in Byte	Interpretiert als
Verschlüsselungskontext	Variable	UTF-8 kodierte Schlüssel-Wert-Paare

Dokumentenverlauf für das AWS Encryption SDK Developer Guide

In diesem Thema werden wichtige Aktualisierungen im AWS Encryption SDK -Entwicklerhandbuch beschrieben.

Themen

- [Neueste Aktualisierungen](#)
- [Frühere Aktualisierungen](#)

Neueste Aktualisierungen

Die folgende Tabelle beschreibt signifikante Änderungen an dieser Dokumentation seit November 2017. Neben den hier aufgelisteten größeren Änderungen aktualisieren wir die Dokumentation regelmäßig überarbeitet, um Beschreibungen und Beispiele zu verbessern und Ihre Rückmeldungen zu berücksichtigen. Wenn Sie über wichtige Änderungen benachrichtigt werden möchten, abonnieren Sie den RSS-Feed.

Änderung	Beschreibung	Datum
Allgemeine Verfügbarkeit	Dokumentation für den AWS KMS ECDH-Schlüsselbund und den Raw ECDH-Schlüsselbund hinzugefügt.	17. Juni 2024
AWS-Verschlüsselungs-SDK for Java Version 3.x	Integriert die Bibliothek AWS-Verschlüsselungs-SDK for Java mit den Materialanbietern. Integriert die Unterstützung für Schlüsselringe und den erforderlichen Verschlüsselungskontext CMM.	6. Dezember 2023
AWS Encryption SDK für.NET-Version 4.x	Integriert die Unterstützung für den AWS KMS hierarchischen	12. Oktober 2023

	Schlüsselbund, den erforderlichen Verschlüsselungskontext (CMM) und asymmetrische RSA-Schlüsselringe. AWS KMS	
Allgemeine Verfügbarkeit	Wir stellen die Unterstützung für AWS Encryption SDK für .NET vor.	17. Mai 2022
Änderung der Dokumentation	Ersetzen Sie den AWS Key Management Service Begriff Customer Master Key (CMK) durch AWS KMS key einen KMS-Schlüssel.	30. August 2021
Allgemeine Verfügbarkeit	Unterstützung für hinzugefügt AWS Key Management Service. (AWS KMS) Schlüssel für mehrere Regionen. Schlüssel mit mehreren Regionen sind unterschiedliche AWS KMS Schlüssel AWS-Regionen , die synonym verwendet werden können, da sie dieselbe Schlüssel-ID und dasselbe Schlüsselmaterial haben.	8. Juni 2021
Allgemeine Verfügbarkeit	Die Dokumentation zum verbesserten Entschlüsselungsprozess von Nachrichten wurde hinzugefügt und aktualisiert.	11. Mai 2021

[Allgemeine Verfügbarkeit](#)

Die Dokumentation für die allgemein verfügbare Version von AWS Encryption CLI Version 1.8 wurde hinzugefügt und aktualisiert. x als Ersatz für AWS Encryption CLI Version 1.7. x und AWS Encryption CLI 2.1. x als Ersatz für AWS Encryption CLI 2.0. x.

27. Oktober 2020

[Allgemeine Verfügbarkeit](#)

Die Dokumentation für die allgemein verfügbare Version der AWS Encryption SDK Versionen 1.7 wurde hinzugefügt und aktualisiert. x und 2.0. x, einschließlich eines [Best-Practices-Leitfadens](#), eines [Migrationsleitfadens](#), aktualisierter [Konzepte](#), aktualisierter [Themen zu Programmiersprachen](#), einer aktualisierten [Referenz zu Algorithm Suites](#), einer aktualisierten [Referenz zum Nachrichtenformat](#) und einem neuen [Beispiel für ein Nachrichtenformat](#).

24. September 2020

[Allgemeine Verfügbarkeit](#)

Dokumentation zur allgemeinen Verfügbarkeitsversion des [AWS-Verschlüsselungs-SDK for JavaScript](#) wurde hinzugefügt und aktualisiert.

1. Oktober 2019

Vorschauversion	Dokumentation der öffentlichen Beta-Version des AWS-Verschlüsselungs-SDK for JavaScript wurde hinzugefügt und aktualisiert.	21. Juni 2019
Allgemeine Verfügbarkeit	Dokumentation zur allgemeinen Verfügbarkeitsversion des AWS-Verschlüsselungs-SDK for C wurde hinzugefügt und aktualisiert.	16. Mai 2019
Vorschauversion	Dokumentation der Vorversion des AWS-Verschlüsselungs-SDK for C wurde hinzugefügt.	5. Februar 2019
Neue Veröffentlichung	Dokumentation der Befehlszeilenschnittstelle für das AWS Encryption SDK hinzugefügt.	20. November 2017

Frühere Aktualisierungen

In der folgenden Tabelle werden wichtige Änderungen am AWS Encryption SDK Developer Guide vor November 2017 beschrieben.

Änderung	Beschreibung	Datum
Neue Version	Zusätzliches Datenschlüssel-Caching Kapitel für die neue Funktion. Das Thema the section called “Initialisierungsvektor – Referenz” wurde hinzugefügt, die erklärt, dass das SDK von der Generierung zufälliger	31. Juli 2017

Änderung	Beschreibung	Datum
	<p>IVs zur Konstruktion deterministischer IVs übergegangen ist.</p> <p>Das the section called “Konzepte” Thema zur Erläuterung von Konzepten , einschließlich des neuen Managers für kryptografische Materialien, wurde hinzugefügt.</p>	
Aktualisierung	<p>Erweiterung der Nachricht enformat – Referenz- Dokumentation in einen neuen AWS Encryption SDK Referenz-Abschnitt.</p> <p>Es wurde ein Abschnitt über die AWS Encryption SDK Unterstützte Algorithmen-Pakete hinzugefügt.</p>	21. März 2017
Neue Version	<p>Das unterstützt AWS Encryption SDK jetzt zusätzlich zu die Python Programmiersprache Java.</p>	21. März 2017
Erstversion	<p>Erste Version der AWS Encryption SDK und dieser Dokumentation.</p>	22. März 2016

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.