aws

Guía para desarrolladores

AWS Encryption SDK

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Encryption SDK: Guía para desarrolladores

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Las marcas comerciales y la imagen comercial de Amazon no se pueden utilizar en relación con ningún producto o servicio que no sea de Amazon, de ninguna manera que pueda causar confusión entre los clientes y que menosprecie o desacredite a Amazon. Todas las demás marcas registradas que no son propiedad de Amazon son propiedad de sus respectivos propietarios, que pueden o no estar afiliados, conectados o patrocinados por Amazon.

Table of Contents

¿Qué es el AWS Encryption SDK? 1
Desarrollado en repositorios de código abierto 2
Compatibilidad con bibliotecas y servicios de cifrado 3
Compatibilidad y mantenimiento 4
Más información 4
Envío de comentarios 5
Conceptos 6
Cifrado de sobre 7
Clave de datos 8
Clave de encapsulamiento 9
Conjuntos de claves y proveedores de claves maestras 10
Contexto de cifrado 11
Mensaje cifrado 13
Conjunto de algoritmos 14
Administrador de materiales criptográficos 14
Cifrado simétrico y asimétrico 15
Compromiso clave 15
Política de compromiso 17
Firmas digitales 18
Funcionamiento del SDK 19
Cómo cifra los datos AWS Encryption SDK 20
Cómo AWS Encryption SDK descifra un mensaje cifrado 20
Conjuntos de algoritmos admitidos 21
Recomendado: AES-GCM con derivación de clave, firma y compromiso clave 22
Otros conjuntos de algoritmos admitidos 23
Interactuando con AWS KMS 25
Prácticas recomendadas 27
Configuración del SDK 32
Selección de un lenguaje de programación 32
Seleccionar las claves de encapsulamiento 33
Uso de varias regiones AWS KMS keys 34
Elección de un conjunto de algoritmos 56
Limitar las claves de datos cifrados 67
Crear un filtro de detección 74

Requerir contextos de cifrado 77
Establecer una política de compromiso 84
Trabajo con datos de streaming 85
Claves de datos de almacenamiento en caché 85
Almacenes de claves 86
La terminología y los conceptos del almacén de claves 86
Implementación de permisos de privilegio mínimo 87
Crear un almacén de claves 88
Configurar las acciones del almacén de claves 89
Configure las acciones de su almacén de claves 90
Crea claves de rama 95
Rote la clave de rama activa 99
Conjuntos de claves 101
Cómo funcionan los conjuntos de claves 101
Compatibilidad de conjuntos de claves 103
Diferentes requisitos para los conjuntos de claves de cifrado 104
Conjuntos de claves compatibles y proveedores de claves maestras 104
AWS KMS llaveros 106
AWS KMS Permisos necesarios para los llaveros 108
Identificarse AWS KMS keys en un AWS KMS llavero 109
Crear un anillo de claves AWS KMS 110
Uso de un anillo de claves de detección AWS KMS 125
Uso de un anillo de claves de detección AWS KMS regional 132
AWS KMS Llaveros jerárquicos 141
Funcionamiento 143
Requisitos previos 145
Permisos necesarios 146
Elige un caché 146
Crear un conjunto de claves jerárquico 160
AWS KMS Llaveros ECDH 168
Permisos necesarios para los AWS KMS llaveros ECDH 169
Crear un conjunto de claves ECDH AWS KMS 170
Creación de un conjunto de claves AWS KMS de detección del ECDH 177
Conjunto de claves de AES sin formato 183
Conjuntos de claves de RSA sin formato 190
Llaveros ECDH sin procesar 200

Creación de un conjunto de claves ECDH sin procesar 201
Conjuntos de claves múltiples 219
Lenguajes de programación 229
C 229
Instalación 230
Uso del SDK de C 231
Ejemplos 236
.NET 244
Instalar y compilar 245
Debugging 246
Ejemplos 246
Go 255
Requisitos previos 256
Instalación 256
Java 257
Requisitos previos 257
Instalación 259
Ejemplos 260
JavaScript 273
Compatibilidad 274
Instalación 276
Módulos 277
Ejemplos 280
Python 288
Requisitos previos 289
Instalación 289
Ejemplos 291
Rust 298
Requisitos previos 299
Instalación 299
Ejemplos 300
Interfaz de línea de comandos 302
Instalación de la CLI de 304
Cómo utilizar la CLI 307
Ejemplos 321
Referencia de sintaxis y parámetros 346

Versiones ... 360
Almacenamiento en caché de claves de datos ... 364
Cómo utilizar el almacenamiento en caché de claves de datos ... 365
Uso del almacenamiento en caché de claves de datos: Step-by-step ... 366
Ejemplo de almacenamiento en caché de claves de datos: cifrado de una cadena ... 374
Configuración de los umbrales de seguridad de la caché ... 390
Información detallada sobre el almacenamiento en caché de claves de datos ... 392
Funcionamiento del almacenamiento en caché de claves de datos ... 392
Creación de una caché de materiales criptográficos ... 396
Creación de un administrador de materiales criptográficos de almacenamiento en caché ... 397
¿Qué es una entrada en la caché de claves de datos? ... 398
Contexto de cifrado: cómo seleccionar las entradas de la caché ... 399
¿Usa mi aplicación claves de datos almacenadas en caché? ... 399
Ejemplo de almacenamiento en caché de claves de datos ... 400
Resultados de la caché local ... 401
Código de ejemplo ... 402
CloudFormation plantilla ... 414
Versiones del AWS Encryption SDK ... 429
C ... 430
C#/.NET ... 431
Interfaz de línea de comandos (CLI) ... 431
Java ... 434
Go ... 436
JavaScript ... 437
Python ... 438
Rust ... 440
Detalles de la versión ... 440
Versiones anteriores a 1.7.x ... 441
Versión 1.7.x ... 441
Versión 2.0.x ... 444
Versión 2.2.x ... 445
Versión 2.3.x ... 446
Migrando su AWS Encryption SDK ... 448
Cómo migrar e implementar ... 450
Etapa 1: actualice su aplicación a la última versión 1.x ... 450
Etapa 2: actualice la aplicación a la versión más reciente ... 451

Actualización de los proveedores de claves AWS KMS maestras 452
Migración al modo estricto 453
Migración al modo de detección 457
Actualización de los AWS KMS llaveros 461
Establecer su política de compromiso 463
¿Cómo establecer su política de compromiso? 465
Solución de problemas de migración a las versiones más recientes 476
Objetos obsoletos o eliminados 477
Conflicto de configuración: conjunto de políticas y algoritmos de compromiso 477
Conflicto de configuración: política de compromiso y texto cifrado 478
Falla en la validación del compromiso clave 479
Otras fallas de cifrado 479
Otras fallas de descifrado 479
Consideraciones de restauración 480
Preguntas frecuentes 481
¿En qué se AWS Encryption SDK diferencia del AWS SDKs? 481
¿En qué se AWS Encryption SDK diferencia del cliente de cifrado Amazon S3? 482
¿Qué algoritmos criptográficos admite y cuál es el AWS Encryption SDK predeterminado? 482
¿Cómo es el vector de inicialización (IV) generado y dónde se almacena? 483
¿Cómo se genera, cifra y descifra cada clave de datos? 483
¿Cómo se realizar el seguimiento de las claves de datos que se han utilizado para cifrar los datos? 484
¿Cómo se AWS Encryption SDK almacenan las claves de datos cifrados con sus datos cifrados? 484
¿Cuánta sobrecarga añade el formato del AWS Encryption SDK mensaje a mis datos cifrados? 484
¿Puedo utilizar mi propio proveedor de claves maestras? 485
¿Puedo cifrar datos con más de una clave de encapsulamiento? 485
¿Qué tipos de datos puedo cifrar con el? AWS Encryption SDK 485
¿Cómo se AWS Encryption SDK cifran y descifran las transmisiones input/output (E/S)? 486
Referencia 487
Referencia de formato de mensajes 487
Estructura del encabezado 488
Estructura del cuerpo 496
Estructura del pie de página 502
Ejemplos de formato de mensajes 502

Datos con trama (formato de mensaje versión 1) 503
Datos con trama (formato de mensaje versión 2) 507
Datos sin trama (formato de mensaje versión 1) 509
Referencia de AAD del cuerpo 513
Referencia de algoritmos 514
Referencia de vectores de inicialización 519
AWS KMS Detalles técnicos del llavero jerárquico 520
Historial de revisión 522
Actualizaciones recientes 522
Actualizaciones anteriores 525
..... dxxvii

¿Qué es el AWS Encryption SDK?

AWS Encryption SDK Se trata de una biblioteca de cifrado del lado del cliente diseñada para que todos puedan cifrar y descifrar datos fácilmente utilizando los estándares y las mejores prácticas del sector. Permite centrarse en la funcionalidad central de la aplicación, en lugar de en la mejor manera de cifrar y descifrar los datos. AWS Encryption SDK Se proporciona de forma gratuita bajo la licencia Apache 2.0.

AWS Encryption SDK Responde a preguntas como las siguientes para usted:

- ¿Qué algoritmo de cifrado debo usar?
- ¿Cómo o en qué modo debo usar ese algoritmo?
- ¿Cómo se genera la clave de cifrado?
- ¿Cómo se protege la clave de cifrado y dónde debo almacenarla?
- ¿Cómo hago que los datos cifrados sean portables?
- ¿Cómo me aseguro de que el destinatario previsto pueda leer los datos cifrados?
- ¿Cómo me aseguro de que los datos cifrados no sufran ninguna modificación desde que se escriben hasta que se leen?
- ¿Cómo utilizo las claves de datos que se AWS KMS devuelven?

Con ellas AWS Encryption SDK, se define un proveedor de claves maestras o un conjunto de claves que determina qué claves de empaquetado se utilizan para proteger los datos. A continuación, se cifran y descifran los datos mediante los sencillos métodos proporcionados por el. AWS Encryption SDK El AWS Encryption SDK hace el resto.

Sin ellas AWS Encryption SDK, podría dedicar más esfuerzo a crear una solución de cifrado que a la funcionalidad principal de su aplicación. AWS Encryption SDK Responde a estas preguntas proporcionando lo siguiente.

Una implementación predeterminada que cumple las prácticas recomendadas en criptografía

> De forma predeterminada, AWS Encryption SDK genera una clave de datos única para cada objeto de datos que cifra. Esto es conforme con la práctica recomendada en criptografía que consiste en usar claves de datos únicas para cada operación de cifrado.

AWS Encryption SDK Cifra los datos mediante un algoritmo de clave simétrica, autenticado y seguro. Para obtener más información, consulte the section called “Conjuntos de algoritmos admitidos”.

Un marco para proteger las claves de datos mediante claves de encapsulamiento

AWS Encryption SDK Protege las claves de datos que cifran sus datos al cifrarlos bajo una o más claves de empaquetado. Al proporcionar un marco para cifrar las claves de datos con más de una clave de empaquetado, AWS Encryption SDK ayuda a que los datos cifrados sean portátiles.

Por ejemplo, cifra los datos con una entrada AWS KMS y una AWS KMS key clave de tu HSM local. Puede usar cualquiera de las claves de encapsulamiento para descifrar los datos, en caso de que alguna no esté disponible o la persona que llama no tenga permiso para usar ambas claves.

Un mensaje con formato que almacene las claves de datos cifradas con los datos cifrados

AWS Encryption SDK Almacena los datos cifrados y la clave de datos cifrados juntos en un mensaje cifrado que utiliza un formato de datos definido. Esto significa que no necesita realizar un seguimiento ni proteger las claves de datos que cifran sus datos porque lo AWS Encryption SDK hacen por usted.

Algunas implementaciones lingüísticas AWS Encryption SDK requieren un AWS SDK, pero AWS Encryption SDK no lo requieren ni dependen de ningún AWS servicio. Cuenta de AWS Cuenta de AWS Solo necesitará uno si decide usarlo AWS KMS keyspara proteger sus datos.

Desarrollado en repositorios de código abierto

AWS Encryption SDK Está desarrollado en repositorios de código abierto en. GitHub Puede usar estos repositorios para ver el código, leer y enviar los problemas y encontrar información específica sobre la implementación de su lenguaje.

- SDK de cifrado de AWS para C — aws-encryption-sdk-c
- AWS Encryption SDK para .NET: directorio.NET del `aws-encryption-sdk` repositorio.
- AWS CLI de cifrado — aws-encryption-sdk-cli
- SDK de cifrado de AWS para Java — aws-encryption-sdk-java
- SDK de cifrado de AWS para JavaScript — aws-encryption-sdk-javascript
- SDK de cifrado de AWS para Python — aws-encryption-sdk-python

- AWS Encryption SDK para Rust: directorio Rust del `aws-encryption-sdk` repositorio.
- AWS Encryption SDK para Go: directorio Go del `aws-encryption-sdk` repositorio

Compatibilidad con bibliotecas y servicios de cifrado

AWS Encryption SDK Es compatible con varios lenguajes de programación. Todas las implementaciones de lenguaje son interoperables. Puede cifrar con una implementación de lenguaje y descifrar con otra. La interoperabilidad puede estar sujeta a restricciones de lenguaje. Si es así, estas restricciones se describen en el tema que trata de la implementación del lenguaje. Además, al cifrar y descifrar, debe usar conjuntos de claves compatibles o claves maestras y proveedores de claves maestras. Para obtener más información, consulte the section called “Compatibilidad de conjuntos de claves”.

Sin embargo, AWS Encryption SDK no puede interoperar con otras bibliotecas. Dado que cada biblioteca devuelve datos cifrados en un formato diferente, no se puede cifrar con una biblioteca y luego descifrar con otra.

Cliente de cifrado de DynamoDB y cifrado del cliente de Amazon S3

> AWS Encryption SDK No pueden descifrar los datos cifrados por el cliente de cifrado de DynamoDB o el cifrado del lado del cliente de Amazon S3. Estas bibliotecas no pueden descifrar el mensaje cifrado que devuelven. AWS Encryption SDK

AWS Key Management Service (AWS KMS)

> AWS Encryption SDK Pueden usar AWS KMS keysclaves de datos para proteger sus datos, incluidas las claves KMS de varias regiones. Por ejemplo, puede configurar el AWS Encryption SDK para cifrar sus datos AWS KMS keys en uno o más de sus archivos. Cuenta de AWS Sin embargo, debe usar el AWS Encryption SDK para descifrar esos datos.
>
> AWS Encryption SDK No se puede descifrar el texto cifrado que devuelven las operaciones de AWS KMS cifrado o. ReEncrypt Del mismo modo, la operación de AWS KMSdescifrado no puede descifrar el mensaje cifrado que devuelve. AWS Encryption SDK
>
> Solo AWS Encryption SDK admite claves KMS de cifrado simétrico. No se puede utilizar una clave KMS asimétrica para el cifrado o la firma en el AWS Encryption SDK. El AWS Encryption SDK genera sus propias claves de firma ECDSA para conjuntos de algoritmos que firmen mensajes.

Compatibilidad y mantenimiento

AWS Encryption SDK Utiliza la misma política de mantenimiento que utilizan el AWS SDK y las herramientas, incluidas las fases de control de versiones y ciclo de vida. Como práctica recomendada, le recomendamos que utilice la última versión disponible del lenguaje de programación y que la AWS Encryption SDK actualice a medida que se publiquen nuevas versiones. Cuando una versión requiera cambios importantes, como la actualización desde AWS Encryption SDK versiones anteriores a la 1.7. x a las versiones 2.0. x y versiones posteriores, proporcionamos instrucciones detalladas para ayudarle.

Cada implementación del lenguaje de programación AWS Encryption SDK se desarrolla en un GitHub repositorio de código abierto independiente. Es probable que el ciclo de vida y la fase de soporte de cada versión varíen de un repositorio a otro. Por ejemplo, una versión determinada de AWS Encryption SDK puede estar en la fase de disponibilidad general (soporte total) en un lenguaje de programación, pero la end-of-support fase en un lenguaje de programación diferente. Le recomendamos que utilice una versión totalmente compatible siempre que sea posible y evite las versiones que ya no lo sean.

Para conocer la fase del ciclo de vida de AWS Encryption SDK las versiones de su lenguaje de programación, consulte el `SUPPORT_POLICY.rst` archivo de cada AWS Encryption SDK repositorio.

- SDK de cifrado de AWS para C — Support_policy.rst
- AWS Encryption SDK para .NET — Support_policy.rst
- AWS CLI de cifrado: Support_policy.rst
- SDK de cifrado de AWS para Java — Support_Policy.rst
- SDK de cifrado de AWS para JavaScript — Support_policy.rst
- SDK de cifrado de AWS para Python — Support_policy.rst

Para obtener más información, consulte la política de mantenimiento Versiones del AWS Encryption SDKAWS SDKs y las herramientas en la Guía de referencia de herramientas y herramientas. AWS SDKs

Más información

Para obtener más información sobre el cifrado AWS Encryption SDK y el cifrado del lado del cliente, pruebe estas fuentes.

- Para obtener ayuda con los términos y conceptos que se utilizan en este SDK, consulte Conceptos en el AWS Encryption SDK.
- Para ver las pautas de prácticas recomendadas, consulte Mejores prácticas para AWS Encryption SDK.
- Para obtener información sobre cómo funciona este SDK, consulte Funcionamiento del SDK.
- Para ver ejemplos que muestran cómo configurar las opciones de AWS Encryption SDK, consulte. Configuración del AWS Encryption SDK
- Para obtener información técnica detallada, consulte la Referencia.
- Para ver las especificaciones técnicas del AWS Encryption SDK, consulte la AWS Encryption SDK especificación en GitHub.
- Para obtener respuestas a sus preguntas sobre el uso del AWS Encryption SDK, lea y publique en el foro de debate sobre AWS Crypto Tools.

Para obtener información sobre las implementaciones del AWS Encryption SDK en diferentes lenguajes de programación.

- C: Consulte SDK de cifrado de AWS para C la documentación de AWS Encryption SDK C y el aws-encryption-sdk-crepositorio en GitHub él.
- C#/.NET: Consulte AWS Encryption SDK para.NET y el aws-encryption-sdk-netdirectorio del `aws-encryption-sdk` repositorio en. GitHub
- Interfaz de línea de comandos: consulteAWS Encryption SDK interfaz de línea de comandos, lea los documentos de la CLI de AWS cifrado y el aws-encryption-sdk-clirepositorio en GitHub.
- Java: Consulte SDK de cifrado de AWS para Java AWS Encryption SDK Javadoc y el aws-encryption-sdk-javarepositorio activados. GitHub

  JavaScript: Consulte the section called “JavaScript” y el aws-encryption-sdk-javascriptrepositorio está activado. GitHub
- Python: consulte SDK de cifrado de AWS para Python la documentación de AWS Encryption SDK Python y el aws-encryption-sdk-pythonrepositorio en él GitHub.

Envío de comentarios

Agradecemos sus comentarios. Si tiene una pregunta o comentario, o un problema del que informar, utilice los siguientes recursos.

- Si descubre una posible vulnerabilidad de seguridad en el AWS Encryption SDK, notifíquelo al equipo AWS de seguridad. No cree una GitHub emisión pública.
- Para enviar comentarios al respecto AWS Encryption SDK, archiva un problema en el GitHub repositorio del lenguaje de programación que estés utilizando.
- Para proporcionar comentarios sobre esta documentación, use el enlace Comentarios de esta página. También puedes archivar un problema o contribuir al aws-encryption-sdk-docsrepositorio de código abierto en GitHub el que se encuentra esta documentación.

Conceptos en el AWS Encryption SDK

En esta sección se presentan los conceptos utilizados en el y AWS Encryption SDK se proporciona un glosario y una referencia. Está diseñado para ayudarte a entender cómo AWS Encryption SDK funciona y los términos que utilizamos para describirlo.

¿Necesita ayuda?

- Descubra cómo AWS Encryption SDK utiliza el cifrado de sobres para proteger sus datos.
- Obtenga información sobre los elementos del cifrado de sobre: las claves de datos que protegen sus datos y las claves de encapsulamiento que protegen sus claves de datos.
- Obtenga información sobre los conjuntos de claves y los proveedores de claves maestras que determinan qué claves de encapsulamiento debe utilizar.
- Obtenga información sobre el contexto de cifrado que agrega integridad a su proceso de cifrado. Es opcional, pero es una práctica recomendada.
- Obtenga información sobre el mensaje cifrado que devuelven los métodos de cifrado.
- Entonces estará listo para usarlo AWS Encryption SDK en su lenguaje de programación preferido.

Temas

- Cifrado de sobre
- Clave de datos
- Clave de encapsulamiento
- Conjuntos de claves y proveedores de claves maestras
- Contexto de cifrado
- Mensaje cifrado
- Conjunto de algoritmos

- Administrador de materiales criptográficos
- Cifrado simétrico y asimétrico
- Compromiso clave
- Política de compromiso
- Firmas digitales

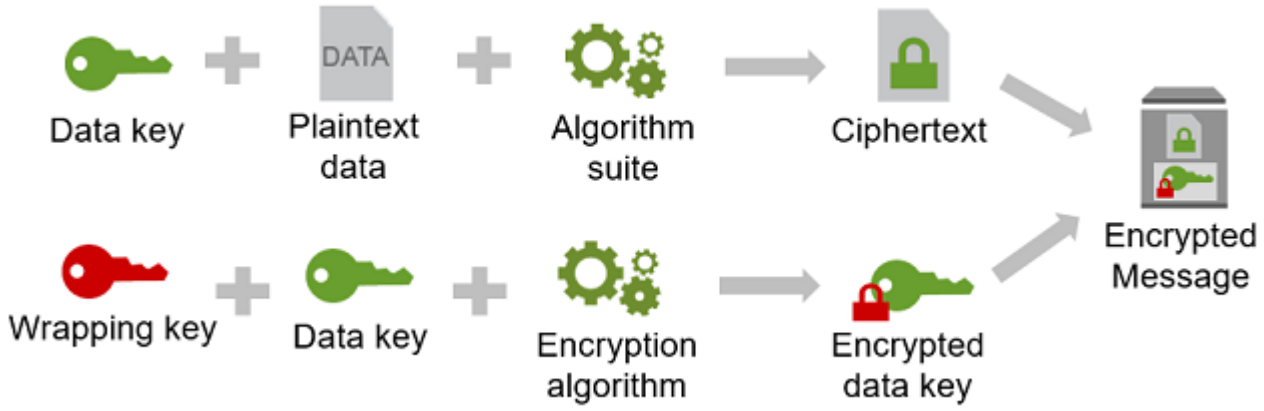
Cifrado de sobre

La seguridad de los datos cifrados depende en parte de la protección de la clave de datos que permite descifrarlos. Una práctica recomendada aceptada para proteger la clave de datos consiste en cifrarla. Para ello, necesita otra clave de cifrado, conocida como clave de cifrado clave o clave de encapsulamiento. Esta práctica de utilizar una clave de encapsulamiento para cifrar las claves de datos se denomina cifrado de sobre.

Protección de las claves de datos

AWS Encryption SDK Encripta cada mensaje con una clave de datos única. Luego, cifra la clave de datos con la clave de encapsulamiento que especifique. Almacena las claves de datos cifradas con los datos cifrados en el mensaje cifrado que devuelve.

Para especificar la clave de encapsulamiento, utilice un conjunto de claves o proveedor de claves maestras.

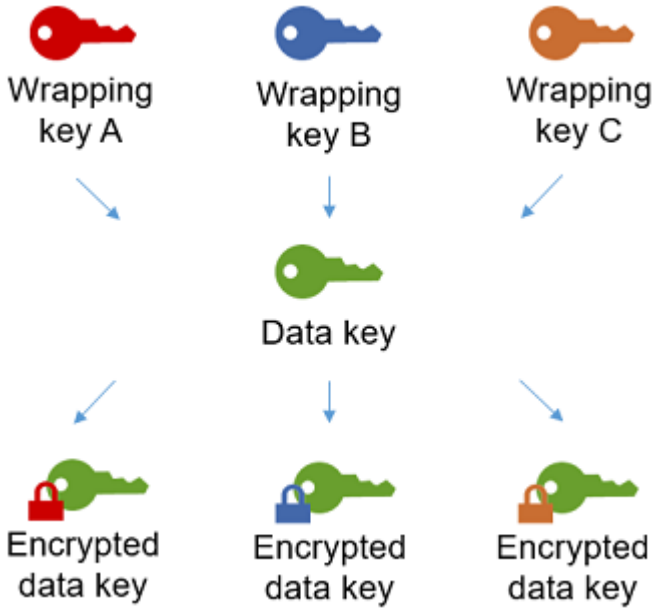


Cifrado de los mismos datos con varias claves múltiples

Puede cifrar la clave de datos con varias claves de encapsulamiento. Es posible que desee proporcionar diferentes claves de encapsulamiento para distintos usuarios, o claves de encapsulamiento de diferentes tipos o en diferentes ubicaciones. Cada una de las claves de

encapsulamiento cifra la misma clave de datos. AWS Encryption SDK Almacena todas las claves de datos cifrados con los datos cifrados del mensaje cifrado.

Para descifrar los datos, debe proporcionar al menos una clave de encapsulamiento que pueda descifrar una de las claves de datos cifrados.



Combinación de los puntos fuertes de varios algoritmos

Para cifrar los datos, de forma predeterminada, AWS Encryption SDK utiliza un sofisticado conjunto de algoritmos con cifrado simétrico AES-GCM, una función de derivación de claves (HKDF) y firma. Para cifrar la clave de datos, puede especificar un algoritmo de cifrado simétrico o asimétrico adecuado a su clave de encapsulamiento.

En general, los algoritmos de cifrado de clave simétrica son más rápidos y producen textos cifrados más pequeños que el cifrado de clave pública o asimétrico. Sin embargo, los algoritmos de clave pública proporcionan una separación inherente entre las funciones y facilitan la administración de las claves. Para combinar las fortalezas de cada uno, puede cifrar datos sin procesar mediante el cifrado de clave simétrica y, a continuación, cifrar la clave de datos con el cifrado de clave pública.

Clave de datos

Una clave de datos es una clave de cifrado que AWS Encryption SDK utiliza para cifrar sus datos. Cada clave de datos es una matriz de bytes que es conforme a los requisitos para claves

criptográficas. A menos que utilices el almacenamiento en caché de claves de datos, AWS Encryption SDK utiliza una clave de datos única para cifrar cada mensaje.

No es necesario especificar, generar, implementar, extender, proteger ni usar claves de datos. El AWS Encryption SDK se encarga de la tarea automáticamente cuando llama a las operaciones de cifrado y descifrado.

Para proteger sus claves de datos, las AWS Encryption SDK cifra en una o más claves de cifrado, conocidas como claves de empaquetado o claves maestras. Una vez que AWS Encryption SDK utiliza las claves de datos de texto sin formato para cifrar los datos, las elimina de la memoria lo antes posible. A continuación, almacena las claves de datos cifradas con los datos cifrados en el mensaje cifrado que devuelven las operaciones de cifrado. Para obtener más información, consulte the section called “Funcionamiento del SDK”.

> **Tip**
>
> En el AWS Encryption SDK, distinguimos las claves de datos de las claves de cifrado de datos. Varios de los conjuntos de algoritmos compatibles, incluido el conjunto predeterminado, utilizan una función de derivación de clave que impide que la clave de datos alcance sus límites criptográficos. La función de derivación de clave toma la clave de datos como entrada y devuelve una clave de cifrado de datos que es la que se utiliza realmente para cifrar los datos. Por este motivo, a menudo decimos que los datos se cifran "bajo" una clave de datos, en lugar de "por" una clave de datos.

Cada clave de datos cifrados incluye metadatos, incluido el identificador de la clave de encapsulamiento que la cifró. Estos metadatos facilitan la identificación de AWS Encryption SDK las claves de empaquetado válidas al descifrar.

Clave de encapsulamiento

Una clave de encapsulamiento es una clave de cifrado de claves que el AWS Encryption SDK utiliza para cifrar la clave de datos que cifra sus datos. Cada clave de datos de texto no cifrado se puede cifrar en una o varias claves de encapsulamiento. Usted determina qué claves de encapsulamiento se utilizan para proteger sus datos al configurar un conjunto de claves o un proveedor de claves maestras.

> **Note**
>
> La clave de encapsulamiento hace referencia a las claves de un conjunto de claves o de un proveedor de claves maestras. La clave maestra suele estar asociada a la clase `MasterKey` de la que se crea una instancia cuando utiliza un proveedor de claves maestras.

AWS Encryption SDK Admite varias claves de empaquetado de uso común, como las AWS Key Management Service (AWS KMS) simétricas AWS KMS keys(incluidas las claves KMS multirregionales), las claves AES-GCM (modo de Standard/Galois contador de cifrado avanzado) sin procesar y las claves RSA sin procesar. También puede ampliar o implementar sus propias claves de encapsulamiento.

Cuando se utiliza el cifrado de sobre, es necesario proteger las claves de encapsulamiento contra el acceso no autorizado. Esto lo puede hacer de cualquiera de las siguientes maneras:

- Utilice un servicio web diseñado para este fin, como AWS Key Management Service (AWS KMS).
- Utilice un módulo de seguridad de hardware (HSM) como los que ofrece AWS CloudHSM.
- Utilice otras herramientas y servicios de administración de claves.

Si no tiene un sistema de administración de claves, le recomendamos que lo haga. AWS KMS Se AWS Encryption SDK integra AWS KMS para ayudarte a proteger y usar tus llaves de embalaje. Sin embargo, AWS Encryption SDK no requiere AWS ningún AWS servicio.

Conjuntos de claves y proveedores de claves maestras

Para especificar las claves de empaquetado que se utilizan para el cifrado y el descifrado, se utiliza un conjunto de claves o un proveedor de claves maestras. Puede utilizar los anillos de claves y los proveedores de claves maestras que AWS Encryption SDK ofrecen o diseñar sus propias implementaciones. El AWS Encryption SDK proporciona conjuntos de claves y proveedores de llaves maestras que son compatibles entre sí y están sujetos a restricciones de idioma. Para obtener más información, consulte Compatibilidad de conjuntos de claves.

Un conjunto de claves genera, cifra y descifra claves de datos. Al definir un conjunto de claves, puede especificar las claves de encapsulamiento que cifran sus claves de datos. La mayoría de los conjuntos de claves especifican al menos una clave de encapsulamiento o un servicio que proporciona y protege las claves de encapsulamiento. También puede definir un conjunto de claves

sin claves de encapsulamiento o un conjunto de claves más complejo con opciones de configuración adicionales. Si necesita ayuda para elegir y usar los llaveros que AWS Encryption SDK definen, consulte. Conjuntos de claves

Los llaveros son compatibles con los siguientes lenguajes de programación:

- SDK de cifrado de AWS para C
- SDK de cifrado de AWS para JavaScript
- AWS Encryption SDK para .NET
- Versión 3. x del SDK de cifrado de AWS para Java
- Versión 4. x de SDK de cifrado de AWS para Python, cuando se usa con la dependencia opcional de la biblioteca de proveedores de material criptográfico (MPL).
- Versión 1. x de la AWS Encryption SDK para Rust
- Versión 0.1. x o posterior del AWS Encryption SDK for Go

Un proveedor de claves maestras es una alternativa a un conjunto de claves. El proveedor de claves maestras devuelve las claves de encapsulamiento (o claves maestras) que especifique. Cada clave maestra está asociada a un proveedor de claves maestras, pero un proveedor de claves maestras suele proporcionar varias claves maestras. Los proveedores de claves maestras son compatibles con Java, Python y la CLI de AWS cifrado.

Debe especificar un conjunto de claves (o un proveedor de claves maestras) para el cifrado. Puede especificar el mismo conjunto de claves (o proveedor de claves maestras) o uno diferente para el descifrado. Al cifrar, AWS Encryption SDK utiliza todas las claves de empaquetado que especifique para cifrar la clave de datos. Al descifrar, el AWS Encryption SDK utiliza únicamente las claves de encapsulamiento que especifique para descifrar una clave de datos cifrados. Especificar las claves de empaquetado para el descifrado es opcional, pero es una AWS Encryption SDK práctica recomendada.

Para obtener más información sobre la especificación de claves de encapsulamiento, consulte Seleccionar las claves de encapsulamiento.

Contexto de cifrado

Para mejorar la seguridad de las operaciones criptográficas, incluya un contexto de cifrado en todas las solicitudes de cifrado de datos. El uso de un contexto de cifrado es opcional, pero es una práctica criptográfica recomendada que le aconsejamos.

Un contexto de cifrado es un conjunto de pares de nombre-valor que contienen datos autenticados adicionales no secretos y arbitrarios. El contexto de cifrado puede contener todos los datos que elija, pero suele constar de datos que son útiles para el registro y el seguimiento, tales como los datos sobre el tipo de archivo, la finalidad o la propiedad. Cuando se descifran los datos, el contexto de cifrado se vincula criptográficamente a los datos cifrados, de tal forma que se requiere el mismo contexto de cifrado para descifrar los datos. El AWS Encryption SDK incluye el contexto de cifrado en texto no cifrado en el encabezado del mensaje cifrado que devuelve.

El contexto de cifrado que AWS Encryption SDK utilizan consiste en el contexto de cifrado que especifique y un par de claves públicas que agrega el administrador de materiales criptográficos (CMM). En concreto, cuando se utiliza un algoritmo de cifrado con firma, el CMM añade un par nombre-valor al contexto de cifrado que consta de un nombre reservado, `aws-crypto-public-key` y un valor que representa la clave de verificación pública. El `aws-crypto-public-key` nombre en el contexto de cifrado lo reserva el nombre AWS Encryption SDK y no se puede utilizar como nombre en ningún otro par del contexto de cifrado. Para obtener más información, consulte AAD en la Referencia del formato de mensajes.

El siguiente contexto de cifrado de ejemplo consta de dos pares de contexto de cifrado especificados en la solicitud y el par de claves públicas que añade el CMM.

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

Para descifrar los datos, se transfieren en el mensaje cifrado. Como AWS Encryption SDK pueden extraer el contexto de cifrado del encabezado del mensaje cifrado, no es necesario que proporcione el contexto de cifrado por separado. No obstante, el contexto de cifrado puede ayudarle a confirmar que está descifrando el mensaje cifrado correcto.

- En la interfaz de línea de comandos de AWS Encryption SDK (CLI), si proporciona un contexto de cifrado en un comando de descifrado, la CLI verifica que los valores están presentes en el contexto de cifrado del mensaje cifrado antes de que devuelva los datos en texto no cifrado.
- En otras implementaciones de lenguaje de programación, la respuesta de descifrado incluye el contexto de cifrado y los datos en texto no cifrado. La función de descifrado en la aplicación debería comprobar siempre que el contexto de cifrado en la respuesta de descifrado incluya el contexto de cifrado en la solicitud de cifrado (o un subconjunto) antes de devolver los datos en texto no cifrado.

> **Note**
>
> Las siguientes versiones admiten el contexto de cifrado CMM requerido, que puede utilizar para requerir un contexto de cifrado en todas las solicitudes de cifrado.
>
> - Versión 3. x del SDK de cifrado de AWS para Java
> - Versión 4. x del AWS Encryption SDK para .NET
> - Versión 4. x de SDK de cifrado de AWS para Python, cuando se usa con la dependencia opcional de la biblioteca de proveedores de material criptográfico (MPL).
> - Versión 1. x de la AWS Encryption SDK para Rust
> - Versión 0.1. x o posterior del AWS Encryption SDK for Go

Al elegir un contexto de cifrado, recuerde que no es un secreto. El contexto de cifrado se muestra en texto no cifrado en el encabezado del mensaje cifrado que devuelve AWS Encryption SDK . Si lo está utilizando AWS Key Management Service, es posible que el contexto de cifrado también aparezca en texto plano en los registros y registros de auditoría, por ejemplo. AWS CloudTrail

Para ver ejemplos de cómo enviar y verificar un contexto de cifrado en el código, consulte los ejemplos de su lenguaje de programación preferido.

Mensaje cifrado

Al cifrar los datos con el AWS Encryption SDK, devuelve un mensaje cifrado.

Un mensaje cifrado es una estructura de datos formateados que incluye los datos cifrados junto con copias cifradas de las claves de datos, el ID del algoritmo y, opcionalmente, un contexto de cifrado y una firma digital. Las operaciones de cifrado del AWS Encryption SDK devuelven un mensaje cifrado y las operaciones de descifrado toman el mensaje cifrado como entrada.

Combinar los datos cifrados y sus claves de datos cifradas simplifica la operación de descifrado y evita tener que almacenar y administrar claves de datos cifradas de forma independiente de los datos que cifran.

Para obtener información técnica sobre el mensaje cifrado, consulte Formato del mensaje cifrado.

Conjunto de algoritmos

AWS Encryption SDK Utiliza un conjunto de algoritmos para cifrar y firmar los datos del mensaje cifrado que devuelven las operaciones de cifrado y descifrado. AWS Encryption SDK admite varios conjuntos de algoritmos. Todos los conjuntos admitidos utilizan Advanced Encryption Standard (AES) como algoritmo principal y lo combinan con otros algoritmos y valores.

AWS Encryption SDK Establece un conjunto de algoritmos recomendado como predeterminado para todas las operaciones de cifrado. Este conjunto predeterminado puede cambiar en función de las mejoras aportadas a los estándares y las prácticas recomendadas. Puede especificar un conjunto de algoritmos alternativo en las solicitudes de cifrado de datos o al crear un administrador de materiales criptográficos (CMM), pero es preferible utilizar el predeterminado, a menos que el alternativo sea necesario para su situación. El valor predeterminado actual es AES-GCM con una función de derivación de extract-and-expand claves (HKDF) basada en HMAC, un compromiso de claves, una firma con un algoritmo de firma digital de curva elíptica (ECDSA) y una clave de cifrado de 256 bits.

Si su aplicación requiere un alto rendimiento y los usuarios que cifran los datos y los que los descifran gozan de la misma confianza, puede considerar la posibilidad de especificar un conjunto de algoritmos sin firma digital. Sin embargo, recomendamos encarecidamente un conjunto de algoritmos que incluya el compromiso clave y una función de derivación de claves. Los conjuntos de algoritmos sin estas características solamente se admiten con fines de compatibilidad con versiones anteriores.

Administrador de materiales criptográficos

El administrador de materiales criptográficos (CMM) reúne los materiales criptográficos que se utilizan para cifrar y descifrar datos. Los materiales criptográficos incluyen las claves de datos cifradas y en texto no cifrado, así como una clave de firma de mensaje opcional. Nunca se interactúa directamente con el CMM. Los métodos de cifrado y descifrado se encargan de ello.

Puede utilizar la CMM predeterminada o la CMM de almacenamiento en caché que proporcionan, o bien escribir una CMM personalizada. AWS Encryption SDK También puede especificar un CMM, pero no es obligatorio. Al especificar un conjunto de claves o un proveedor de claves maestras, se AWS Encryption SDK crea una CMM predeterminada para usted. El CMM predeterminado obtiene los materiales de cifrado o descifrado del conjunto de claves o el proveedor de claves maestras que especifique. Esto podría requerir una llamada a un servicio criptográfico como AWS Key Management Service (AWS KMS).

Dado que el CMM actúa como enlace entre el anillo de claves AWS Encryption SDK y el proveedor de claves maestras, es un punto ideal para la personalización y la ampliación, por ejemplo, para la aplicación de políticas y el almacenamiento en caché. AWS Encryption SDK Proporciona una CMM de almacenamiento en caché para admitir el almacenamiento en caché de las claves de datos.

Cifrado simétrico y asimétrico

El cifrado simétrico utiliza la misma clave para cifrar y descifrar datos.

El cifrado asimétrico utiliza un par de claves de datos relacionados matemáticamente. Una clave del par cifra los datos; solo la otra clave del par puede descifrarlos.

AWS Encryption SDK Utiliza el cifrado de sobres. Cifra los datos con una clave de datos simétrica. Cifra la clave de datos simétrica con una o más claves de encapsulamiento simétricas o asimétricas. Devuelve un mensaje cifrado que incluye los datos cifrados y al menos una copia cifrada de la clave de datos.

Cifrar los datos (cifrado simétrico)

> Para cifrar los datos, AWS Encryption SDK utiliza una clave de datos simétrica y un conjunto de algoritmos que incluye un algoritmo de cifrado simétrico. Para descifrar los datos, AWS Encryption SDK utiliza la misma clave de datos y el mismo conjunto de algoritmos.

Cifrar la clave de datos (cifrado simétrico o asimétrico)

> El conjunto de claves o proveedor de claves maestras que proporciona a una operación de cifrado y descifrado determina cómo se cifra y descifra la clave de datos simétrica. Puede elegir un conjunto de claves o un proveedor de claves maestras que utilice cifrado simétrico, como un AWS KMS anillo de claves, o uno que utilice cifrado asimétrico, como un anillo de claves RSA sin procesar o. `JceMasterKey`

Compromiso clave

AWS Encryption SDK Admite el compromiso clave (también conocido como robustez), una propiedad de seguridad que garantiza que cada texto cifrado solo se pueda descifrar en un único texto plano. Para ello, el compromiso clave garantiza que solo se utilice la clave de datos que cifró su mensaje para descifrarlo. Cifrar y descifrar los datos con un compromiso clave es una práctica recomendada de AWS Encryption SDK.

La mayoría de los cifrados simétricos modernos (incluido el AES) cifran el texto no cifrado con una única clave secreta, como la clave de datos única que el AWS Encryption SDK utiliza para cifrar cada mensaje de texto no cifrado. Al descifrar estos datos con la misma clave de datos, se obtiene un texto no cifrado idéntico al original. El descifrado con una clave diferente suele fallar. Sin embargo, es posible descifrar un texto cifrado con dos claves diferentes. En raras ocasiones, es posible encontrar una clave que pueda descifrar algunos bytes del texto cifrado y convertirlo en un texto no cifrado diferente, pero aún inteligible.

AWS Encryption SDK Siempre cifra cada mensaje de texto sin formato con una clave de datos única. Puede cifrar esa clave de datos con varias claves de encapsulamiento (o claves maestras), pero las claves de encapsulamiento siempre cifran la misma clave de datos. Sin embargo, un mensaje cifrado sofisticado y creado manualmente puede contener diferentes claves de datos, cada una cifrada con una clave de encapsulamiento diferente. Por ejemplo, si un usuario descifra el mensaje cifrado, devuelve 0x0 (falso), mientras que otro usuario que descifra el mismo mensaje cifrado obtiene 0x1 (verdadero).

Para evitar este escenario, AWS Encryption SDK admite el compromiso de clave al cifrar y descifrar. Cuando AWS Encryption SDK cifra un mensaje con un compromiso de clave, vincula criptográficamente la clave de datos única que produjo el texto cifrado a la cadena de compromiso de clave, un identificador de clave de datos no secreto. A continuación, almacena la cadena de compromiso clave en los metadatos del mensaje cifrado. Cuando descifra un mensaje con una clave comprometida, AWS Encryption SDK verifica que la clave de datos sea la única clave de ese mensaje cifrado. Si se produce un error en la verificación de la clave de datos, se produce un error en la operación de descifrado.

Se introduce la asistencia para el compromiso clave en la versión 1.7.x, que puede descifrar los mensajes con un compromiso clave, pero no cifrará con un compromiso clave. Puede utilizar esta versión para aprovechar al máximo la capacidad de descifrar el texto cifrado con el compromiso clave. La versión 2.0.x incluye soporte completo para el compromiso clave. De forma predeterminada, cifra y descifra solo con el compromiso clave. Se trata de una configuración ideal para las aplicaciones que no necesitan descifrar el texto cifrado en versiones anteriores del. AWS Encryption SDK

Si bien la práctica recomendada es cifrar y descifrar con compromiso clave, le dejamos decidir cuándo utilizarla y ajustar el ritmo al que la adopta. A partir de la versión 1.7. x, AWS Encryption SDK admite una política de compromiso que establece el conjunto de algoritmos predeterminado y limita los conjuntos de algoritmos que se pueden utilizar. Esta política determina si sus datos se cifran y descifran con compromiso clave.

El compromiso clave da como resultado un mensaje cifrado un poco más grande (+ 30 bytes) y su procesamiento lleva más tiempo. Si su aplicación es muy sensible al tamaño o al rendimiento, puede optar por excluirse del compromiso clave. Pero hágalo solo si es necesario.

Para obtener más información sobre cómo migrar a versiones 1.7.x y 2.0.x, incluidas sus características de compromiso clave, consulte Migrando su AWS Encryption SDK. Para obtener información técnica sobre el compromiso clave, consulte the section called “Referencia de algoritmos” y the section called “Referencia de formato de mensajes”.

Política de compromiso

Una política de compromiso es una configuración que determina si su aplicación cifra y descifra con compromiso clave. Cifrar y descifrar los datos con un compromiso clave es una práctica recomendada de AWS Encryption SDK.

La política de compromiso tiene tres valores.

> **Note**
>
> Es posible que tenga que desplazarse en forma horizontal o vertical para ver toda la tabla.

Valores de la política de compromiso

Valor	Cifra con un compromiso clave	Cifra sin un compromiso clave	Descifra con un compromiso clave	Descifra sin un compromiso clave
ForbidEncryptAllowDecrypt				
RequireEncryptAllowDecrypt				
RequireEncryptRequireDecrypt				

La configuración de la política de compromiso se introdujo en AWS Encryption SDK la versión 1.7. x. Es válido en todos los lenguajes de programación compatibles.

- `ForbidEncryptAllowDecrypt` descifra con o sin compromiso clave, pero no cifra con compromiso clave. Este valor, introducido en la versión 1.7. x, está diseñado para preparar a todos los hosts que ejecutan su aplicación para que descifren con compromiso de clave antes de que encuentren un texto cifrado con compromiso de clave.
- `RequireEncryptAllowDecrypt` siempre cifra con un compromiso clave. Puede descifrar con o sin compromiso clave. Este valor introducido en la versión 2.0.x le permite empezar a cifrar con un compromiso clave, pero seguir descifrando textos cifrados heredados sin compromiso clave.
- `RequireEncryptRequireDecrypt` cifra y descifra solo con el compromiso clave. Este valor es el predeterminado para la versión 2.0.x. Utilice este valor cuando esté seguro de que todos sus textos cifrados están cifrados con un compromiso clave.

La configuración de la política de compromiso determina qué conjuntos de algoritmos puede utilizar. A partir de la versión 1.7. x, AWS Encryption SDK admite conjuntos de algoritmos para el compromiso clave, con y sin firma. Si especifica un conjunto de algoritmos que entra en conflicto con su política de compromiso, el AWS Encryption SDK devuelve un error.

Si necesita ayuda para establecer su política de compromiso, consulte Establecer su política de compromiso.

Firmas digitales

AWS Encryption SDK Cifra los datos mediante un algoritmo de cifrado autenticado, el AES-GCM, y el proceso de descifrado verifica la integridad y autenticidad de un mensaje cifrado sin utilizar una firma digital. Dado que el AES-GCM utiliza claves simétricas, cualquier persona que pueda descifrar la clave de datos utilizada para descifrar el texto cifrado también podría crear manualmente un nuevo texto cifrado, lo que podría suponer un problema de seguridad. Por ejemplo, si utilizas una AWS KMS key como clave de empaquetado, un usuario con `kms:Decrypt` permisos podría crear textos cifrados sin necesidad de llamar. `kms:Encrypt`

Para evitar este problema, AWS Encryption SDK admite añadir una firma con un algoritmo de firma digital de curva elíptica (ECDSA) al final de los mensajes cifrados. Cuando se utiliza un conjunto de algoritmos de firma, AWS Encryption SDK genera una clave privada temporal y una clave pública para cada mensaje cifrado. AWS Encryption SDK Almacena la clave pública en el contexto de cifrado de la clave de datos y descarta la clave privada. Esto garantiza que nadie pueda crear otra firma que

se verifique con la clave pública. El algoritmo vincula la clave pública a la clave de datos cifrados como datos autenticados adicionales en el encabezado del mensaje, lo que evita que los usuarios que solo pueden descifrar los mensajes alteren la clave pública o afecten a la verificación de la firma.

La verificación de firmas añade un importante coste de rendimiento en el descifrado. Si los usuarios que cifran los datos y los que los descifran tienen el mismo nivel de confianza, considere la posibilidad de utilizar un conjunto de algoritmos que no incluya firmas.

> **Note**
>
> Si el conjunto de claves o el acceso al material criptográfico empaquetado no delimitan entre cifradores y descifradores, las firmas digitales no proporcionan ningún valor criptográfico.

AWS KMS Los anillos de claves, incluido el AWS KMS anillo de claves RSA asimétrico, pueden distinguir entre cifradores y descifradores en función de las políticas clave y de IAM. AWS KMS

Debido a su naturaleza criptográfica, los siguientes conjuntos de claves no pueden distinguir entre cifradores y descifradores:

- AWS KMS Anillo de claves jerárquico
- AWS KMS Llavero ECDH
- Conjunto de claves de AES sin formato
- Conjunto de claves de RSA sin formato
- Llavero ECDH sin procesar

Cómo AWS Encryption SDK funciona

Los flujos de trabajo de esta sección explican cómo AWS Encryption SDK cifra los datos y descifra los mensajes cifrados. Estos flujos de trabajo describen el proceso básico mediante las características predeterminadas. Para obtener más información sobre la definición y el uso de componentes personalizados, consulte el GitHub repositorio de cada implementación de lenguaje compatible.

AWS Encryption SDK Utiliza el cifrado de sobres para proteger sus datos. Cada mensaje se cifra en una clave de datos única. Luego, la clave de datos la cifran las claves de encapsulamiento

que especifique. Para descifrar el mensaje cifrado, AWS Encryption SDK utiliza las claves de empaquetado que especifique para descifrar al menos una clave de datos cifrada. Luego, puede descifrar el texto cifrado y devolver un mensaje de texto no cifrado.

¿Necesita ayuda con la terminología que utilizamos en el AWS Encryption SDK? Consulte the section called “Conceptos”.

Cómo cifra los datos AWS Encryption SDK

AWS Encryption SDK Proporciona métodos que cifran cadenas, matrices de bytes y flujos de bytes. Para ver ejemplos de código, consulte el tema Ejemplos de cada sección de Lenguajes de programación.

1. Cree un conjunto de claves (o un proveedor de claves maestras) que especifique las claves de encapsulamiento que protegen sus datos.
2. Pase el conjunto de claves y los datos del texto no cifrado a un método de cifrado. Recomendamos que pase un contexto de cifrado opcional, no secreto.
3. El método de cifrado solicita al conjunto de claves los materiales de cifrado. El anillo de claves devuelve claves de cifrado de datos únicas para el mensaje: una clave de datos de texto simple y una copia de esa clave de datos cifrada por cada una de las claves de empaquetado especificadas.
4. El método de cifrado utiliza la clave de datos en texto no cifrado para cifrar los datos y, a continuación, la descarta. Si proporcionó un contexto de cifrado (una práctica recomendada de AWS Encryption SDK), el método de cifrado vincula criptográficamente el contexto de cifrado a los datos cifrados.
5. El método de cifrado devuelve un mensaje cifrado que contiene los datos cifrados, las claves de datos cifrados y otros metadatos, incluido el contexto de cifrado, si se utilizó alguno.

Cómo AWS Encryption SDK descifra un mensaje cifrado

AWS Encryption SDK Proporciona métodos que descifran el mensaje cifrado y devuelven texto sin formato. Para ver ejemplos de código, consulte el tema Ejemplos de cada sección de Lenguajes de programación.

El conjunto de claves (o proveedor de claves maestras) que descifra el mensaje cifrado debe ser compatible con el utilizado para cifrar el mensaje. Una de sus claves de encapsulamiento debe

poder descifrar una clave de datos cifrada del mensaje cifrado. Para obtener información sobre la compatibilidad con los conjuntos de claves y los proveedores de claves maestras, consulte the section called “Compatibilidad de conjuntos de claves”.

1. Cree un conjunto de claves o un proveedor de claves maestras con claves de encapsulamiento que puedan descifrar los datos. Puede utilizar el mismo conjunto de claves que proporcionó para el método de cifrado u otro distinto.
2. Pase el mensaje cifrado y el conjunto de claves a un método de descifrado.
3. El método de descifrado pide al conjunto de claves o al proveedor de claves maestras que descifre una de las claves de datos cifradas del mensaje cifrado. Pasa información del mensaje cifrado, incluidas las claves de datos cifradas.
4. El conjunto de claves utiliza sus claves de encapsulamiento para descifrar una de las claves de datos cifradas. Si se realiza correctamente, la respuesta incluye la clave de datos en texto no cifrado. Si ninguna de las claves de encapsulamiento especificadas por el conjunto de claves o el proveedor de claves maestras puede descifrar una clave de datos cifrada, se produce un error en la llamada de descifrado.
5. El método de descifrado utiliza la clave de datos en texto no cifrado para descifrar los datos, descarta la clave de datos de texto no cifrado y devuelve los datos de texto no cifrado.

Conjuntos de algoritmos compatibles en AWS Encryption SDK

Un conjunto de algoritmos es un conjunto de algoritmos criptográficos y sus valores relacionados. Los sistemas criptográficos utilizan la implementación del algoritmo para generar el mensaje de texto cifrado.

El conjunto de algoritmos utiliza el AWS Encryption SDK algoritmo estándar de cifrado avanzado (AES) en Galois/Counter modo (GCM), conocido como AES-GCM, para cifrar los datos sin procesar. AWS Encryption SDK Admite claves de cifrado de 256, 192 y 128 bits. La longitud del vector de inicialización (IV) es siempre de 12 bytes. La longitud de la etiqueta de autenticación es siempre de 16 bytes.

De forma predeterminada, AWS Encryption SDK utiliza un conjunto de algoritmos con AES-GCM con una función de derivación de claves (HKDF) basada en HMAC, firma y una extract-and-expand clave de cifrado de 256 bits. Si la política de compromiso requiere un compromiso clave, AWS Encryption SDK selecciona un conjunto de algoritmos que también soporte el compromiso clave; de lo contrario, selecciona un conjunto de algoritmos con derivación y firma de claves, pero sin compromiso clave.

Recomendado: AES-GCM con derivación de clave, firma y compromiso clave

Se AWS Encryption SDK recomienda un conjunto de algoritmos que obtenga una clave de cifrado AES-GCM mediante el suministro de una clave de cifrado de datos de 256 bits a la función de derivación de claves basada en extract-and-expand HMAC (HKDF). AWS Encryption SDK Agrega una firma del algoritmo de firma digital de curva elíptica (ECDSA). Para respaldar el compromiso clave, este conjunto de algoritmos también deriva una cadena de compromiso clave (un identificador de clave de datos no secreto) que se almacena en los metadatos del mensaje cifrado. Esta cadena de compromiso clave también se obtiene a través de la HKDF mediante un procedimiento similar al de la obtención de la clave de cifrado de datos.

AWS Encryption SDK Suite de algoritmos

Algoritmo de cifrado	Longitud de la clave de cifrado de datos (en bits)	Algoritmo de derivación de clave	Algoritmo de firma	Compromiso clave
AES-GCM	256	HKDF con SHA-384	ECDSA con P-384 y SHA-384	HKDF con SHA-512

La HKDF ayuda a evitar la reutilización accidental de una clave de cifrado de datos y reduce el riesgo de utilizar una clave de datos en exceso.

Para la firma, este conjunto de algoritmos utiliza el ECDSA con un algoritmo de función hash criptográfica (SHA-384). ECDSA se utiliza de forma predeterminada, aunque no se especifique en la política de la clave maestra subyacente. La firma de mensajes verifica que el remitente del mensaje estaba autorizado a cifrar los mensajes y no los repudia. Resulta especialmente útil cuando la política de autorización de una clave maestra permite que un conjunto de usuarios cifre los datos y otro conjunto diferente de usuarios los descifre.

Los conjuntos de algoritmos con un compromiso clave garantizan que cada texto cifrado se descifre en un solo texto simple. Para ello, validan la identidad de la clave de datos utilizada como entrada en el algoritmo de cifrado. Al cifrar, estos conjuntos de algoritmos obtienen una cadena de compromiso clave. Antes de descifrar, validan que la clave de datos coincida con la cadena de compromiso clave. En caso contrario, el comando de descifrado genera un error.

Otros conjuntos de algoritmos admitidos

AWS Encryption SDK Es compatible con los siguientes conjuntos de algoritmos alternativos para garantizar la compatibilidad con versiones anteriores. En general, no se recomiendan estos conjuntos de algoritmos. Sin embargo, reconocemos que la firma puede afectar considerablemente el rendimiento, por lo que ofrecemos un conjunto de compromiso clave con derivación de clave para esos casos. En el caso de las aplicaciones que deben hacer concesiones de rendimiento más significativas, seguimos ofreciendo paquetes que no cuentan con la firma, el compromiso clave y la derivación de clave.

AES-GCM sin un compromiso clave

Los conjuntos de algoritmos sin compromiso clave no validan la clave de los datos antes de descifrarlos. Como resultado, estos conjuntos de algoritmos pueden descifrar un único texto cifrado en diferentes mensajes de texto simple. Sin embargo, dado que los conjuntos de algoritmos con compromiso clave generan un mensaje cifrado un poco más grande (+30 bytes) y tardan más en procesarse, es posible que no sean la mejor opción para todas las aplicaciones.

AWS Encryption SDK Admite un conjunto de algoritmos con derivación, compromiso y firma de claves y otro con derivación y compromiso de claves, pero sin firma. No recomendamos utilizar un conjunto de algoritmos sin un compromiso clave. Si es necesario, le recomendamos un conjunto de algoritmos con derivación de clave y compromiso clave, pero sin firma. Sin embargo, si el perfil de rendimiento de su aplicación admite el uso de un conjunto de algoritmos, se recomienda utilizar un conjunto de algoritmos con compromiso clave, derivación de clave y firma.

AES-GCM sin firma

Los conjuntos de algoritmos sin firma carecen de la firma ECDSA, que proporciona autenticidad y no repudio. Use este conjunto solo cuando los usuarios que cifran los datos y aquellos que los descifran son igual de confiables.

Al utilizar un conjunto de algoritmos sin firma, le recomendamos que elija uno con derivación de clave y compromiso clave.

AES-GCM sin derivación de clave

Los conjuntos de algoritmos sin derivación de clave utilizan la clave de cifrado de datos como clave de cifrado AES-GCM, en lugar de utilizar una función de derivación de clave para obtener una clave única. No recomendamos el uso de este conjunto para generar texto cifrado, pero lo AWS Encryption SDK admite por motivos de compatibilidad.

Para obtener más información acerca de cómo se representan y utilizan estos conjuntos en la biblioteca, consulte the section called “Referencia de algoritmos”.

Uso del AWS Encryption SDK con AWS KMS

Para utilizar el AWS Encryption SDK, debe configurar los llaveros o los proveedores de claves maestras con claves de empaquetado. Si no tiene una infraestructura de claves, le recomendamos que use AWS Key Management Service (AWS KMS). Muchos de los ejemplos de código del AWS Encryption SDK requieren un AWS KMS key.

Para interactuar con ellos AWS KMS, se AWS Encryption SDK requiere el AWS SDK del lenguaje de programación preferido. La biblioteca AWS Encryption SDK cliente funciona con el AWS SDKs para admitir las claves maestras almacenadas en AWS KMS.

Para prepararse para usar el AWS Encryption SDK con AWS KMS

1. Crea un Cuenta de AWS. Para obtener más información, consulte ¿Cómo creo y activo una nueva cuenta de Amazon Web Services? en el Centro de AWS conocimiento.
2. Cree un cifrado AWS KMS key simétrico. Para obtener más información, consulte Creación de claves en la Guía para desarrolladores de AWS Key Management Service .

   > **Tip**
   >
   > Para usarlo AWS KMS key mediante programación, necesitará el ID de clave o el nombre de recurso de Amazon (ARN) del. AWS KMS key Para obtener ayuda para encontrar el ID de la clave y el ARN de una AWS KMS key, consulte Búsqueda del ID y el ARN de la clave en la Guía para desarrolladores de AWS Key Management Service .

3. Genere un ID de clave de acceso y una clave de acceso de seguridad. Puede utilizar el identificador de la clave de acceso y la clave de acceso secreta de un usuario de IAM o bien puede utilizarlos AWS Security Token Service para crear una nueva sesión con credenciales de seguridad temporales que incluyan un identificador de clave de acceso, una clave de acceso secreta y un token de sesión. Como práctica recomendada de seguridad, le recomendamos que utilice credenciales temporales en lugar de las credenciales a largo plazo asociadas a sus cuentas de usuario de IAM o usuario AWS (raíz).

   Para crear un usuario de IAM con una clave de acceso, consulte Creación de usuarios de IAM en la Guía del usuario de IAM.

   Para obtener más información acerca de las credenciales de seguridad temporales, consulte Credenciales de seguridad temporales en la guía del usuario de IAM.

4. Configure sus AWS credenciales siguiendo las instrucciones de las letras AWS SDK para JavaAWS SDK para JavaScript, AWS SDK para Python (Boto)o AWS SDK para C++(para C) y el ID de la clave de acceso y la clave de acceso secreta que generó en el paso 3. Si generó credenciales temporales, también tendrá que especificar el token de sesión.

   Este procedimiento le AWS SDKs permite firmar las solicitudes AWS por usted. En los ejemplos de código con los AWS Encryption SDK que se interactúa se AWS KMS supone que ha completado este paso.
5. Descargue e instale el AWS Encryption SDK. Para obtener información sobre cómo hacerlo, consulte las instrucciones de instalación del lenguaje de programación que desea utilizar.

Mejores prácticas para AWS Encryption SDK

AWS Encryption SDK Está diseñado para facilitarle la protección de sus datos utilizando los estándares y las mejores prácticas de la industria. Si bien se seleccionan muchas de las prácticas recomendadas en los valores predeterminados, algunas son opcionales, pero se recomiendan siempre que sea práctico.

Uso de la versión más reciente

Cuando empiece a utilizar el AWS Encryption SDK, utilice la última versión que se ofrece en su lenguaje de programación preferido. Si ha estado utilizando el AWS Encryption SDK, actualícelo a la versión más reciente lo antes posible. Esto garantiza que está utilizando la configuración recomendada y que aprovecha las nuevas propiedades de seguridad para proteger sus datos. Para obtener más información sobre las versiones compatibles, incluidas las directrices para la migración y la implementación, consulte Compatibilidad y mantenimiento y Versiones del AWS Encryption SDK.

Si una nueva versión deja en desuso algunos elementos del código, sustitúyalos lo antes posible. Las advertencias de obsolescencia y los comentarios del código suelen recomendar una buena alternativa.

Para que las actualizaciones importantes sean más fáciles y menos propensas a errores, en ocasiones ofrecemos una versión temporal o de transición. Utilice estas versiones y la documentación que las acompaña para asegurarse de que puede actualizar su aplicación sin interrumpir el flujo de trabajo de producción.

Use valores predeterminados

El AWS Encryption SDK diseño de las mejores prácticas se basa en sus valores predeterminados. Siempre que sea posible, úselos. Para los casos en los que la configuración predeterminada no sea práctica, ofrecemos alternativas, como conjuntos de algoritmos sin firma. También ofrecemos oportunidades de personalización a los usuarios avanzados, como llaveros personalizados, proveedores de claves maestras y gestores de material criptográfico ()CMMs. Utilice estas alternativas avanzadas con cautela y pida a un ingeniero de seguridad que verifique sus opciones siempre que sea posible.

Cómo utilizar un contexto de cifrado

Para mejorar la seguridad de las operaciones criptográficas, incluya un contexto de cifrado con un valor significativo en todas las solicitudes de cifrado de datos. El uso de un contexto de cifrado

es opcional, pero es una práctica criptográfica recomendada que le aconsejamos. Un contexto de cifrado proporciona datos autenticados adicionales (AAD) para el cifrado autenticado en el AWS Encryption SDK. Aunque no es secreto, el contexto de cifrado puede ayudarle a proteger la integridad y autenticidad de sus datos cifrados.

En el AWS Encryption SDK, solo se especifica un contexto de cifrado al cifrar. Al descifrar, AWS Encryption SDK utiliza el contexto de cifrado del encabezado del mensaje cifrado que devuelve. AWS Encryption SDK Antes de que la aplicación devuelva los datos en texto no cifrado, compruebe que el contexto de cifrado que proporcionó durante el cifrado esté incluido en el contexto de cifrado que se utilizó para descifrar el mensaje. Para obtener más información, consulte los ejemplos en su lenguaje de programación.

Cuando utiliza la interfaz de línea de comandos, AWS Encryption SDK verifica el contexto de cifrado por usted.

Proteja sus claves de encapsulamiento

AWS Encryption SDK Genera una clave de datos única para cifrar cada mensaje de texto sin formato. Luego, cifra la clave de datos con las claves de encapsulamiento que proporciona. Si sus claves de encapsulamiento se pierden o se eliminan, sus datos cifrados son irrecuperables. Si sus claves no están protegidas, es posible que sus datos sean vulnerables.

Utilice claves de encapsulamiento que estén protegidas por una infraestructura de claves segura, como AWS Key Management Service (AWS KMS). Cuando utilice claves de RSA o de AES sin procesar, utilice una fuente de asignación al azar y de almacenamiento duradero que cumpla sus requisitos de seguridad. La mejor práctica es generar y almacenar claves de empaquetado en un módulo de seguridad de hardware (HSM) o en un servicio que proporcione HSMs, por ejemplo AWS CloudHSM, una práctica recomendada.

Utilice los mecanismos de autorización de su infraestructura de claves para limitar el acceso a las claves de encapsulamiento únicamente a los usuarios que lo necesiten. Implemente los principios de las prácticas recomendadas, como el privilegio mínimo. Cuando AWS KMS keys lo utilice, utilice políticas clave y políticas de IAM que implementen los principios de las mejores prácticas.

Especifique sus claves de encapsulamiento

Siempre se recomienda especificar las claves de encapsulamiento de forma explícita al descifrar, así como al cifrar. Cuando lo haga, AWS Encryption SDK utilizará únicamente las claves que especifique. Esta práctica garantiza que solo utilice las claves de cifrado que desee. Al AWS KMS empaquetar las claves, también mejora el rendimiento, ya que evita que utilices claves de otra región Cuenta de AWS o que intentes descifrar con claves para las que no tienes permiso de uso.

Al cifrar, los anillos de claves y los proveedores de claves maestras suministrados requieren que especifique las claves de AWS Encryption SDK empaquetado. Utilizan únicamente las claves de encapsulamiento que especifique. También debe especificar las claves de empaquetado al cifrar y descifrar con anillos de claves AES sin procesar, anillos de claves RSA sin procesar y claves. JCEMaster

Sin embargo, al descifrar con AWS KMS anillos de claves y proveedores de claves maestras, no es necesario especificar las claves de empaquetado. AWS Encryption SDK Pueden obtener el identificador de clave a partir de los metadatos de la clave de datos cifrados. Sin embargo, la práctica recomendada es especificar las claves de encapsulamiento.

Para respaldar esta práctica recomendada cuando se trabaja con el AWS KMS empaquetado de claves, se recomienda lo siguiente:

- Utilice AWS KMS llaveros que especifiquen las claves de empaquetado. Al cifrar y descifrar, estos conjuntos de claves utilizan únicamente las claves de encapsulamiento que especifica.
- Cuando utilice claves AWS KMS maestras y proveedores de claves maestras, utilice los constructores de modo estricto introducidos en la versión 1.7. x del. AWS Encryption SDK Crean proveedores que cifran y descifran solo con las claves de encapsulamiento que especifique. Los constructores para los proveedores de claves maestras que siempre descifran con cualquier clave de encapsulamiento están obsoletos en la versión 1.7.x y se eliminaron en la versión 2.0.x.

Si no es práctico especificar claves de AWS KMS empaquetado para el descifrado, puede utilizar proveedores de detección. Están AWS Encryption SDK en C y JavaScript admiten llaveros de AWS KMS detección. Los proveedores de claves maestras con modo de detección están disponibles para Java y Python en las versiones 1.7.x y posteriores. Estos proveedores de detección, que solo se utilizan para descifrar con claves de AWS KMS empaquetado, les indican explícitamente que utilicen cualquier clave de empaquetado que cifre una clave de datos. AWS Encryption SDK

Si debe utilizar un proveedor de detección, utilice sus características de filtro de detección para limitar las claves de encapsulamiento que utilizan. Por ejemplo, el conjunto de claves regional de detección de AWS KMS usa solo las claves de encapsulamiento en un Región de AWS particular. También puede configurar los conjuntos de AWS KMS claves y los proveedores de claves AWS KMS maestras para que utilicen únicamente las claves de empaquetado, en concreto. Cuentas de AWS Además, como siempre, utilice las políticas clave y las políticas de IAM para controlar el acceso a sus claves de AWS KMS embalaje.

Uso de firmas digitales

Se recomienda utilizar un conjunto de algoritmos para la firma. Las firmas digitales comprueban que el remitente del mensaje estaba autorizado a enviar el mensaje y protegen la integridad del mensaje. Todas las versiones AWS Encryption SDK utilizan conjuntos de algoritmos con firma de forma predeterminada.

Si sus requisitos de seguridad no incluyen las firmas digitales, puede seleccionar un conjunto de algoritmos sin firmas digitales. Sin embargo, se recomienda utilizar firmas digitales, especialmente cuando un grupo de usuarios cifra los datos y otro grupo de usuarios los descifra.

Utilice el compromiso clave

Se recomienda utilizar la característica de seguridad de compromiso clave. Al comprobar la identidad de la clave de datos única que cifró los datos, la clave de confirmación impide descifrar cualquier texto cifrado que pueda dar lugar a más de un mensaje de texto no cifrado.

AWS Encryption SDK Proporciona soporte completo para el cifrado y el descifrado con el compromiso de clave a partir de la versión 2.0. x. De forma predeterminada, todos los mensajes se cifran y descifran con un compromiso clave. Versión 1.7. x de ellos AWS Encryption SDK puede descifrar textos cifrados con un compromiso clave. Está diseñado para ayudar a los usuarios de versiones anteriores a implementar la versión 2.0.x correctamente.

El soporte para el compromiso clave incluye nuevos conjuntos de algoritmos y un nuevo formato de mensaje que produce un texto cifrado de solo 30 bytes más grande que un texto cifrado sin compromiso clave. El diseño minimiza su impacto en el rendimiento para que la mayoría de los usuarios puedan disfrutar de las ventajas de un compromiso clave. Si su aplicación es muy sensible al tamaño y al rendimiento, puede decidir utilizar la configuración de la política de compromiso para deshabilitar el compromiso clave o AWS Encryption SDK permitir el descifrado de los mensajes sin compromiso, pero hágalo solo si es necesario.

Limitar el número de claves de datos cifradas

Se recomienda limitar el número de claves de datos cifradas en los mensajes que descifra, especialmente los que provienen de fuentes que no son de confianza. Descifrar un mensaje con numerosas claves de datos cifradas que no puede descifrar puede provocar demoras prolongadas, aumentar los gastos, limitar su aplicación y otras que comparten su cuenta y, potencialmente, agotar su infraestructura de claves. Sin límites, un mensaje cifrado puede contener hasta 65 535 (2^16 - 1) claves de datos cifrados. Para obtener más información, consulte Limitar las claves de datos cifrados.

Para obtener más información sobre las características de AWS Encryption SDK seguridad en las que se basan estas prácticas recomendadas, consulte Cifrado mejorado del lado del cliente: compromiso explícito KeyIds y clave en el blog de seguridad.AWS

Configuración del AWS Encryption SDK

AWS Encryption SDK Está diseñado para ser fácil de usar. Aunque AWS Encryption SDK tiene varias opciones de configuración, los valores predeterminados se eligen cuidadosamente para que sean prácticos y seguros para la mayoría de las aplicaciones. Sin embargo, es posible que deba ajustar la configuración para mejorar el rendimiento o incluir una característica personalizada en el diseño.

Al configurar la implementación, revise las prácticas AWS Encryption SDK recomendadas e implemente tantas como pueda.

Temas

- Selección de un lenguaje de programación
- Seleccionar las claves de encapsulamiento
- Uso de varias regiones AWS KMS keys
- Elección de un conjunto de algoritmos
- Limitar las claves de datos cifrados
- Crear un filtro de detección
- Configuración del contexto de cifrado requerido (CMM)
- Establecer una política de compromiso
- Trabajo con datos de streaming
- Claves de datos de almacenamiento en caché

Selección de un lenguaje de programación

AWS Encryption SDK Está disponible en varios lenguajes de programación. Las implementaciones del lenguaje están diseñadas para ser totalmente interoperables y ofrecer las mismas características, aunque pueden implementarse de diferentes maneras. Por lo general, se utiliza la biblioteca que es compatible con su aplicación. Sin embargo, puede seleccionar un lenguaje de programación para una implementación concreta. Por ejemplo, si prefiere trabajar con llaveros, puede elegir el SDK de cifrado de AWS para C o el SDK de cifrado de AWS para JavaScript.

Seleccionar las claves de encapsulamiento

AWS Encryption SDK Genera una clave de datos simétrica única para cifrar cada mensaje. A menos que utilice el almacenamiento en caché de claves de datos, no necesita configurar, administrar ni usar las claves de datos. Lo AWS Encryption SDK hace por ti.

Sin embargo, debe seleccionar una o más claves de encapsulamiento para cifrar cada clave de datos. El AWS Encryption SDK también es compatible con las claves AES simétricas y las claves RSA asimétricas en diferentes tamaños. También admite el cifrado simétrico AWS Key Management Service (AWS KMS) AWS KMS keys. Usted es responsable de la seguridad y durabilidad de sus claves de empaquetado, por lo que le recomendamos que utilice una clave de cifrado en un módulo de seguridad de hardware o en un servicio de infraestructura de claves, como AWS KMS.

Para especificar las claves de empaquetado para el cifrado y el descifrado, utilice un conjunto de claves (C, Java JavaScript, .NET y Python) o un proveedor de claves maestras (Java, Python, AWS CLI de cifrado). Puede especificar una clave de encapsulamiento o varias claves de encapsulamiento del mismo tipo o de diferentes tipos. Si utiliza varias claves de encapsulamiento para empaquetar una clave de datos, cada clave de encapsulamiento cifrará una copia de la misma clave de datos. Las claves de datos cifradas (una por clave de empaquetado) se almacenan junto con los datos cifrados en el mensaje cifrado que devuelven. AWS Encryption SDK Para descifrar los datos, primero AWS Encryption SDK deben usar una de sus claves de empaquetado para descifrar una clave de datos cifrada.

Para especificar una AWS KMS key en un conjunto de claves o en un proveedor de claves maestras, utilice un identificador de clave compatible AWS KMS . Para obtener más información sobre los identificadores clave de una AWS KMS clave, consulte los identificadores clave en la AWS Key Management Service Guía para desarrolladores.

- Al cifrar con SDK de cifrado de AWS para Java, SDK de cifrado de AWS para JavaScript, SDK de cifrado de AWS para Python o la CLI de AWS cifrado, puede utilizar cualquier identificador de clave válido (ID de clave, ARN de clave, nombre de alias o ARN de alias) para una clave de KMS. Al cifrar con SDK de cifrado de AWS para C, solo puede utilizar un ID de clave o un ARN de clave.

  Si especifica un nombre de alias o un ARN de alias para una clave KMS al cifrar, el AWS Encryption SDK guarda el ARN de clave actualmente asociado a ese alias; no guarda el alias. Los cambios en el alias no afectan a la clave KMS utilizada para descifrar las claves de datos.

- Al descifrar en modo estricto (en el que se especifican determinadas claves de encapsulamiento), debe utilizar una clave ARN para identificar AWS KMS keys. Este requisito se aplica a todas las implementaciones de lenguaje del AWS Encryption SDK.

  Cuando se cifra con un AWS KMS anillo de claves, AWS Encryption SDK almacena la clave ARN de en los metadatos de la AWS KMS key clave de datos cifrados. Al descifrar en modo estricto, AWS Encryption SDK comprueba que el mismo ARN de clave aparezca en el anillo de claves (o en el proveedor de claves maestras) antes de intentar utilizar la clave de empaquetado para descifrar la clave de datos cifrados. Si utiliza un identificador de clave diferente, no AWS Encryption SDK lo reconocerá ni utilizará AWS KMS key, aunque los identificadores hagan referencia a la misma clave.

Para especificar una clave AES sin procesar o un par de claves RSA sin procesar como clave de encapsulamiento en un conjunto de claves, debe especificar un espacio de nombre y un nombre. En un proveedor de claves maestras, el `Provider ID` es el equivalente al espacio de nombre y el `Key ID` es el equivalente al nombre. Al descifrar, debe utilizar exactamente el mismo espacio de nombre y el mismo nombre para cada clave de encapsulamiento sin procesar que utilizó al cifrar. Si utiliza un nombre o espacio de nombres diferente, no reconocerá ni AWS Encryption SDK utilizará la clave de empaquetado, aunque el material de la clave sea el mismo.

Uso de varias regiones AWS KMS keys

Puede utilizar claves AWS Key Management Service multirregionales (AWS KMS) como claves de empaquetado en el. AWS Encryption SDK Si cifra con una clave multirregional en una Región de AWS, puede desencriptar utilizando una clave multirregional relacionada en otra diferente. Región de AWS La compatibilidad con claves multirregionales se introdujo en la versión 2.3. x de la AWS Encryption SDK y la versión 3.0. x de la CLI AWS de cifrado.

AWS KMS Las claves multirregionales son un conjunto de AWS KMS keys diferentes claves Regiones de AWS que tienen el mismo material y el mismo identificador de clave. Puede usar estas claves relacionadas como si fueran la misma clave en diferentes regiones. Las claves multirregionales son compatibles con los escenarios habituales de recuperación ante desastres y copias de seguridad, que requieren el cifrado en una región y el descifrado en una región diferente sin necesidad de realizar una llamada entre regiones. AWS KMS Para obtener más información sobre las claves de regiones múltiples, consulte Uso de claves de multirregiones en la Guía para desarrolladores de AWS Key Management Service .

Para admitir claves multirregionales, AWS Encryption SDK incluye conjuntos de claves compatibles con AWS KMS múltiples regiones y proveedores de claves maestras. El nuevo multi-Region-aware símbolo de cada lenguaje de programación admite claves de una o varias regiones.

- En el caso de las claves de una sola región, el multi-Region-aware símbolo se comporta igual que el conjunto de claves de una sola región y el proveedor de AWS KMS claves maestras. Intenta descifrar el texto cifrado únicamente con la clave de región única que cifró los datos.
- En el caso de las claves multirregionales, el multi-Region-aware símbolo intenta descifrar el texto cifrado con la misma clave multirregional que cifró los datos o con la réplica multirregional relacionada de la región que especifique.

En los conjuntos de multi-Region-aware claves y en los proveedores de claves maestras que utilizan más de una clave KMS, puede especificar varias claves de una o varias regiones. Sin embargo, solo puede especificar una clave de cada conjunto de claves de réplica multirregionales relacionadas. Si especifica más de un identificador de clave con el mismo ID de clave, se produce un error en la llamada al constructor.

También puede utilizar una clave multirregional con los conjuntos de claves estándar de una sola región y AWS KMS los proveedores de claves maestras. Sin embargo, debe usar la misma clave multirregional en la misma región para cifrar y descifrar. Los conjuntos de claves de una sola región y los proveedores de claves maestras intentan descifrar el texto cifrado únicamente con las claves que cifraron los datos.

Los siguientes ejemplos muestran cómo cifrar y descifrar datos mediante claves multirregionales y los nuevos multi-Region-aware conjuntos de claves y proveedores de claves maestras. En estos ejemplos se cifran los datos de la `us-east-1` región y se descifran los datos de la región mediante las claves de réplica multirregionales relacionadas de cada `us-west-2` región. Antes de ejecutar estos ejemplos, reemplace el ARN de clave multirregional de ejemplo por un valor válido de su Cuenta de AWS.

C

Para cifrar con una clave multirregional, utilice el método `Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder()` para crear una instancia del conjunto de claves. Especifique una clave multirregional.

Este sencillo ejemplo no incluye un contexto de cifrado. Para ver un ejemplo que utiliza un contexto de cifrado en C, consulte Cifrado y descifrado de cadenas.

Para ver un ejemplo completo, consulte kms_multi_region_keys.cpp en el repositorio de SDK de cifrado de AWS para C . 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

Para cifrar con una clave multirregional en la región EE.UU. Este (Norte de Virginia) (us-east-1), cree una instancia de un `CreateAwsKmsMrkKeyringInput` objeto con un identificador de clave para la clave multirregión y un cliente para la región especificada. AWS KMS A continuación, utilice el método `CreateAwsKmsMrkKeyring()` para crear el conjunto de claves.

El método `CreateAwsKmsMrkKeyring()` crea un conjunto de claves con exactamente una clave multirregional. Para cifrar con varias claves de encapsulamiento, incluida una clave multirregional, utilice el método `CreateAwsKmsMrkMultiKeyring()`.

Para ver un ejemplo completo, consulte AwsKmsMrkKeyringExample.cs en el repositorio.NET en. AWS Encryption SDK 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

En este ejemplo, se cifra el archivo de `hello.txt` con una clave multirregional en la región us-east-1. Como en el ejemplo se especifica un ARN de clave con un elemento Region, en este ejemplo no se utiliza el atributo region del parámetro `--wrapping-keys`.

Si el ID de clave de la clave de encapsulamiento no especifica una región, puede utilizar el atributo region de `--wrapping-keys` para especificar la región, por ejemplo, `--wrapping-keys key=$keyID region=us-east-1`.

```
# 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

Para cifrar con una clave multirregional, cree una instancia de `AwsKmsMrkAwareMasterKeyProvider` y especifique una clave multirregional.

Para ver un ejemplo completo, consulte BasicMultiRegionKeyEncryptionExample.javaen el SDK de cifrado de AWS para Java repositorio de 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

Para cifrar con una clave multirregional, use el método `buildAwsKmsMrkAwareStrictMultiKeyringBrowser()` para crear el conjunto de claves y especificar una clave multirregional.

Para ver un ejemplo completo, consulte kms_multi_region_simple.ts en el repositorio de. SDK de cifrado de AWS para 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 SDK de cifrado de AWS para 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

Para cifrar con una clave multirregional, use el método `buildAwsKmsMrkAwareStrictMultiKeyringNode()` para crear el conjunto de claves y especificar una clave multirregional.

Para ver un ejemplo completo, consulte kms_multi_region_simple.ts en el repositorio de. SDK de cifrado de AWS para 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

Para cifrar con una clave multirregional, utilice el método y especifique una clave AWS KMS multirregional. `MRKAwareStrictAwsKmsMasterKeyProvider()`

Para ver un ejemplo completo, consulte mrk_aware_kms_provider.py en el repositorio de SDK de cifrado de AWS para Python . 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
)
```

A continuación, mueva el texto cifrado a la región `us-west-2`. No es necesario volver a cifrar el texto cifrado.

Para descifrar el texto cifrado en modo estricto en la `us-west-2` región, cree una instancia del símbolo multi-Region-aware con la clave ARN de la clave multirregional relacionada en la región. `us-west-2` Si especificas la clave ARN de una clave multirregional relacionada en una región diferente (incluido `us-east-1` el lugar donde se cifró), el multi-Region-aware símbolo realizará una llamada entre regiones para dicha clave. AWS KMS key

Al descifrar en modo estricto, el multi-Region-aware símbolo requiere una clave ARN. Solo acepta un ARN de clave de cada conjunto de claves de varias regiones relacionadas.

Antes de ejecutar estos ejemplos, sustituya el ARN de clave multirregional del ejemplo por un valor válido de su. Cuenta de AWS

C

Para cifrar en el modo estricto con una clave multirregional, utilice el método `Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder()` para crear una instancia del conjunto de claves. Especifique la clave multirregional relacionada en la región local (us-west-2).

Para ver un ejemplo completo, consulte kms_multi_region_keys.cpp en el SDK de cifrado de AWS para C repositorio de. 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

Para descifrar en modo estricto con una sola clave multirregional, use los mismos constructores y métodos que usó para ensamblar la entrada y crear el conjunto de claves para el cifrado. Cree una instancia de un `CreateAwsKmsMrkKeyringInput` objeto con el ARN clave de una clave multirregión relacionada y un AWS KMS cliente para la región EE.UU. Oeste (Oregón) (us-west-2). A continuación, utilice el método `CreateAwsKmsMrkKeyring()` para crear un conjunto de claves multirregional con una clave KMS multirregional.

Para ver un ejemplo completo, consulte AwsKmsMrkKeyringExample.cs en el repositorio.NET de. AWS Encryption SDK 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

Para descifrar con la clave multirregional relacionada en la región us-west-2, utilice el atributo key del parámetro `--wrapping-keys` para especificar su ARN de clave.

```
# 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

Para descifrar en modo estricto, cree una instancia `AwsKmsMrkAwareMasterKeyProvider` y especifique la clave multirregional relacionada en la región local (us-west-2).

Para ver un ejemplo completo, consulte BasicMultiRegionKeyEncryptionExample.java en el SDK de cifrado de AWS para Java repositorio de. 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

Para descifrar en modo estricto, utilice el método `buildAwsKmsMrkAwareStrictMultiKeyringBrowser()` para crear el conjunto de claves y especifique la clave multirregional relacionada en la región local (us-west-2).

Para ver un ejemplo completo, consulte kms_multi_region_simple.ts en el repositorio de. SDK de cifrado de AWS para 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 SDK de cifrado de AWS para 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

Para descifrar en modo estricto, utilice el método `buildAwsKmsMrkAwareStrictMultiKeyringNode()` para crear el conjunto de claves y especifique la clave multirregional relacionada en la región local (us-west-2).

Para ver un ejemplo completo, consulte kms_multi_region_simple.ts en el repositorio de. SDK de cifrado de AWS para 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

Para descifrar en modo estricto, use el método `MRKAwareStrictAwsKmsMasterKeyProvider()` para crear el proveedor de claves maestras. Especifique la clave multirregional relacionada en la región local (us-west-2).

Para ver un ejemplo completo, consulte mrk_aware_kms_provider.py en el repositorio de. SDK de cifrado de AWS para Python 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
)
```

También puede descifrar en modo de detección con claves de múltiples regiones de AWS KMS . Al descifrar en modo de detección, no especifique ningún AWS KMS keys. (Para obtener información sobre los conjuntos de claves de AWS KMS detección de una sola región, consulteUso de un anillo de claves de detección AWS KMS.)

Si ha cifrado con una clave multirregional, el multi-Region-aware símbolo en el modo de descubrimiento intentará descifrarse utilizando una clave multirregional relacionada en la región local. Si no existe ninguno, se produce un error en la llamada. En el modo de detección, no AWS Encryption SDK intentará realizar una llamada entre regiones para obtener la clave multirregional utilizada para el cifrado.

> **Note**
>
> Si utiliza un multi-Region-aware símbolo en el modo de detección para cifrar los datos, se produce un error en la operación de cifrado.

El siguiente ejemplo muestra cómo descifrar con el multi-Region-aware símbolo en el modo de detección. Como no especificas una AWS KMS key, AWS Encryption SDK deben obtener la región de una fuente diferente. Cuando sea posible, especifique la región local de forma explícita. De lo contrario, AWS Encryption SDK obtiene la región local de la región configurada en el AWS SDK para su lenguaje de programación.

Antes de ejecutar estos ejemplos, sustituya el identificador de cuenta del ejemplo y el ARN de clave multirregional por valores válidos de su. Cuenta de AWS

C

Para descifrar en modo de detección con una clave multirregional, utilice el método `Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder()` para crear el conjunto de claves y el método `Aws::Cryptosdk::KmsKeyring::DiscoveryFilter::Builder()` para crear el filtro de detección. Para especificar la región local, defina un `ClientConfiguration` y especifíquelo en el cliente de AWS KMS .

Para ver un ejemplo completo, consulte kms_multi_region_keys.cpp en el SDK de cifrado de AWS para C repositorio de. 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).Buil

/* 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

Para crear un conjunto de claves de multi-Region-aware detección AWS Encryption SDK para .NET, cree una instancia de un `CreateAwsKmsMrkDiscoveryKeyringInput` objeto que utilice un AWS KMS cliente para un objeto concreto Región de AWS y un filtro de detección opcional que limite las claves de KMS a una AWS partición y una cuenta determinadas. Luego, llame al método `CreateAwsKmsMrkDiscoveryKeyring()` con el objeto de entrada. Para ver un ejemplo completo, consulte AwsKmsMrkDiscoveryKeyringExample.cs en el repositorio de.NET AWS Encryption SDK en. GitHub

Para crear un conjunto de claves de multi-Region-aware descubrimiento para más de uno Región de AWS, utilice el `CreateAwsKmsMrkDiscoveryMultiKeyring()` método para crear un conjunto de claves múltiples o utilícelo para crear varios conjuntos de claves de multi-Region-aware descubrimiento y, `CreateAwsKmsMrkDiscoveryKeyring()` a continuación, utilice el `CreateMultiKeyring()` método para combinarlos en un conjunto de claves múltiples.

Para ver un ejemplo, consulte .cs. AwsKmsMrkDiscoveryMultiKeyringExample

```
// 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

Para descifrar en modo de detección, utilice el atributo discovery del parámetro `--wrapping-keys`. Los atributos discovery-account y discovery-partition crean un filtro de detección que es opcional, pero recomendado.

Para especificar la región, este comando incluye el atributo region del parámetro `--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

Para especificar la región local, utilice el parámetro `builder().withDiscoveryMrkRegion`. De lo contrario, el AWS Encryption SDK obtiene la región local de la región configurada en el AWS SDK para Java.

Para ver un ejemplo completo, consulte DiscoveryMultiRegionDecryptionExample.java en el SDK de cifrado de AWS para Java repositorio de. 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

Para descifrar en modo de detección con una clave simétrica multirregional, use el método de `AwsKmsMrkAwareSymmetricDiscoveryKeyringBrowser()`.

Para ver un ejemplo completo, consulte kms_multi_region_discovery.ts en el repositorio de. SDK de cifrado de AWS para 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

Para descifrar en modo de detección con una clave simétrica multirregional, use el método de `AwsKmsMrkAwareSymmetricDiscoveryKeyringNode()`.

Para ver un ejemplo completo, consulte kms_multi_region_discovery.ts en el repositorio de. SDK de cifrado de AWS para 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

Para descifrar en modo de detección con una clave multirregional, use el método de `MRKAwareDiscoveryAwsKmsMasterKeyProvider()`.

Para ver un ejemplo completo, consulte mrk_aware_kms_provider.py en el repositorio de. SDK de cifrado de AWS para Python 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
)
```

Elección de un conjunto de algoritmos

AWS Encryption SDK Admite varios algoritmos de cifrado simétricos y asimétricos para cifrar las claves de datos con las claves de empaquetado que especifique. Sin embargo, cuando utiliza esas claves de datos para cifrar los datos, utiliza de AWS Encryption SDK forma predeterminada un conjunto de algoritmos recomendado que utiliza el algoritmo AES-GCM con la derivación de claves, las firmas digitales y el compromiso de claves. Aunque es probable que el conjunto de algoritmos predeterminado sea adecuado para la mayoría de las aplicaciones, puede elegir un conjunto de algoritmos alternativo. Por ejemplo, algunos modelos de confianza quedarían satisfechos con un conjunto de algoritmos sin firmas digitales. Para obtener información sobre los conjuntos de algoritmos compatibles con el AWS Encryption SDK , consulte Conjuntos de algoritmos compatibles en AWS Encryption SDK.

En los siguientes ejemplos se muestra cómo seleccionar un conjunto de algoritmos alternativo al cifrar. En estos ejemplos se selecciona un conjunto de algoritmos AES-GCM recomendado con derivación y compromiso clave, pero sin firmas digitales. Al cifrar con un conjunto de algoritmos que no incluye firmas digitales, utilice el modo de descifrado solo sin firma al descifrar. Este modo, que falla si encuentra texto cifrado firmado, es el más útil al transmitir el descifrado.

C

Para especificar un conjunto de algoritmos alternativo en el, debe crear una CMM de forma SDK de cifrado de AWS para C explícita. A continuación, use el `aws_cryptosdk_default_cmm_set_alg_id` con el CMM y el conjunto de algoritmos seleccionado.

```
/* 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;
}
```

Al descifrar datos que se cifraron sin firmas digitales, utilice `AWS_CRYPTOSDK_DECRYPT_UNSIGNED`. Esto provoca un error en el descifrado si encuentra texto cifrado firmado.

```
/* 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

Para especificar un conjunto de algoritmos alternativo en AWS Encryption SDK el.NET, especifique la `AlgorithmSuiteId` propiedad de un EncryptInputobjeto. El AWS Encryption SDK archivo for.NET incluye constantes que puede utilizar para identificar el conjunto de algoritmos que prefiera.

AWS Encryption SDK Para .NET no tiene un método para detectar el texto cifrado firmado al transmitir el descifrado porque esta biblioteca no admite la transmisión de datos.

```
// 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

Al cifrar el archivo `hello.txt`, en este ejemplo se utiliza el parámetro `--algorithm` para especificar un conjunto de algoritmos sin firmas digitales.

```
# 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
```

Al descifrar, en este ejemplo se utiliza el parámetro `--decrypt-unsigned`. Se recomienda usar este parámetro para asegurarse de descifrar el texto cifrado sin firmar, especialmente con la CLI, que siempre transmite entradas y salidas.

```
# 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

Para especificar un conjunto de algoritmos alternativo, utilice el método `AwsCrypto.builder().withEncryptionAlgorithm()`. Este ejemplo especifica un conjunto de algoritmos alternativo sin firmas digitales.

```
// 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();
```

Cuando transmitir datos para descifrarlos, utilice el método `createUnsignedMessageDecryptingStream()` para asegurarse de que todo el texto cifrado que vaya a descifrar no esté firmado.

```
// 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

Para especificar un conjunto de algoritmos alternativo, utilice el parámetro `suiteId` con un valor de enum `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, })
```

Al descifrar, utilice el método estándar `decrypt`. SDK de cifrado de AWS para JavaScript en el navegador no tiene un modo `decrypt-unsigned` porque el navegador no admite el streaming.

```
// 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

Para especificar un conjunto de algoritmos alternativo, utilice el parámetro `suiteId` con un valor de enum `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 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, })
```

Al descifrar datos cifrados sin firmas digitales, utilice Stream. decryptUnsignedMessage Este método produce un error si encuentra texto cifrado firmado.

```
// 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

Para especificar un algoritmo de cifrado alternativo, utilice el parámetro `algorithm` con un valor de enum `Algorithm`.

```
# Specify an algorithm suite without signing

# Instantiate a 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]
)
```

```
# 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
)
```

Al descifrar mensajes cifrados sin firmas digitales, utilice el modo de streaming de `decrypt-unsigned`, especialmente al descifrar durante el streaming.

```
# 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
```

Rust

Para especificar un conjunto de algoritmos alternativo en el caso AWS Encryption SDK de Rust, especifique la `algorithm_suite_id` propiedad en su solicitud de cifrado.

```
// Instantiate the AWS Encryption SDK client
let esdk_config = AwsEncryptionSdkConfig::builder().build()?;
let esdk_client = esdk_client::Client::from_conf(esdk_config)?;

// Define the key namespace and key name
let key_namespace: &str = "HSM_01";
let key_name: &str = "AES_256_012";

// Optional: Create an encryption context
let encryption_context = HashMap::from([
    ("encryption".to_string(), "context".to_string()),
    ("is not".to_string(), "secret".to_string()),
    ("but adds".to_string(), "useful metadata".to_string()),
    ("that can help you".to_string(), "be confident that".to_string()),
    ("the data you are handling".to_string(), "is what you think it
 is".to_string()),
]);

// Instantiate the material providers library
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;

// Create Raw AES keyring
let raw_aes_keyring = mpl
    .create_raw_aes_keyring()
    .key_name(key_name)
    .key_namespace(key_namespace)
    .wrapping_key(aws_smithy_types::Blob::new(AESWrappingKey))
    .wrapping_alg(AesWrappingAlg::AlgAes256GcmIv12Tag16)
    .send()
    .await?;

// Encrypt your plaintext data
let plaintext = example_data.as_bytes();

let encryption_response = esdk_client.encrypt()
    .plaintext(plaintext)
    .keyring(raw_aes_keyring.clone())
    .encryption_context(encryption_context.clone())
    .algorithm_suite_id(AlgAes256GcmHkdfSha512CommitKey)
    .send()
    .await?;
```

Go

```
import (
    "context"

 mpl "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygenerated"
 mpltypes "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygeneratedtypes"
 client "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygenerated"
 esdktypes "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygeneratedtypes"
)
// Instantiate the AWS Encryption SDK client
encryptionClient, err := client.NewClient(esdktypes.AwsEncryptionSdkConfig{})
if err != nil {
    panic(err)
}

// Define the key namespace and key name
var keyNamespace = "HSM_01"
var keyName = "AES_256_012"

// Optional: Create an encryption context
encryptionContext := map[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 material providers library
matProv, err := mpl.NewClient(mpltypes.MaterialProvidersConfig{})
if err != nil {
    panic(err)
}

// Create Raw AES keyring
aesKeyRingInput := mpltypes.CreateRawAesKeyringInput{
    KeyName:      keyName,
    KeyNamespace: keyNamespace,
    WrappingKey:  key,
```

```
    WrappingAlg:  mpltypes.AesWrappingAlgAlgAes256GcmIv12Tag16,
}
aesKeyring, err := matProv.CreateRawAesKeyring(context.Background(),
 aesKeyRingInput)
if err != nil {
    panic(err)
}

// Encrypt your plaintext data
algorithmSuiteId := mpltypes.ESDKAlgorithmSuiteIdAlgAes256GcmHkdfSha512CommitKey
res, err := encryptionClient.Encrypt(context.Background(), esdktypes.EncryptInput{
    Plaintext:         []byte(exampleText),
    EncryptionContext: encryptionContext,
    Keyring:           aesKeyring,
    AlgorithmSuiteId:  &algorithmSuiteId,
})
if err != nil {
    panic(err)
}
```

Limitar las claves de datos cifrados

Puede limitar el número de claves de datos cifrados en un mensaje cifrado. Esta característica de práctica recomendada puede ayudarle a detectar un conjunto de claves mal configurado al cifrar o a identificar un texto cifrado malicioso al descifrar. También evita las llamadas innecesarias, costosas y potencialmente exhaustivas a su infraestructura de claves. Limitar las claves de datos cifradas es más valioso al descifrar los mensajes de una fuente que no sea de confianza.

Aunque la mayoría de los mensajes cifrados tienen una clave de datos cifrada para cada clave de encapsulamiento utilizada en el cifrado, un mensaje cifrado puede contener hasta 65 535 claves de datos cifrados. Un agente malicioso podría crear un mensaje cifrado con miles de claves de datos cifradas, sin que se puedan descifrar. Como resultado, AWS Encryption SDK intentaría descifrar cada clave de datos cifrados hasta agotar las claves de datos cifrados del mensaje.

Para limitar las claves de datos cifradas, utilice el parámetro `MaxEncryptedDataKeys`. Este parámetro está disponible para todos los lenguajes de programación compatibles a partir de la versión 1.9.x y 2.2.x del AWS Encryption SDK. Es opcional y válido para cifrar y descifrar. Los siguientes ejemplos descifran los datos que se cifraron con tres claves de encapsulamiento diferentes. El valor `MaxEncryptedDataKeys` se establece en 3.

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

Para limitar las claves de datos cifradas en AWS Encryption SDK para .NET, cree una instancia de un cliente AWS Encryption SDK para .NET y establezca su `MaxEncryptedDataKeys` parámetro opcional en el valor deseado. A continuación, llame al método `Decrypt()` en la instancia AWS Encryption SDK configurada.

```
// 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)
```

Rust

```
// Instantiate the AWS Encryption SDK client with limited encrypted data keys
let esdk_config = AwsEncryptionSdkConfig::builder()
                    .max_encrypted_data_keys(max_encrypted_data_keys)
                    .build()?;
let esdk_client = esdk_client::Client::from_conf(esdk_config)?;

// Define the key namespace and key name
let key_namespace: &str = "HSM_01";
let key_name: &str = "AES_256_012";

// Instantiate the material providers library
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;

//  Generate `max_encrypted_data_keys` raw AES keyrings to use with your keyring
let mut raw_aes_keyrings: Vec<KeyringRef> = vec![];

assert!(max_encrypted_data_keys > 0, "max_encrypted_data_keys MUST be greater than
 0");

let mut i = 0;
while i < max_encrypted_data_keys {
    let aes_key_bytes = generate_aes_key_bytes();
```

```
    let raw_aes_keyring = mpl
        .create_raw_aes_keyring()
        .key_name(key_name)
        .key_namespace(key_namespace)
        .wrapping_key(aes_key_bytes)
        .wrapping_alg(AesWrappingAlg::AlgAes256GcmIv12Tag16)
        .send()
        .await?;

    raw_aes_keyrings.push(raw_aes_keyring);
    i += 1;
}

// Create a Multi Keyring with `max_encrypted_data_keys` AES Keyrings
let generator_keyring = raw_aes_keyrings.remove(0);

let multi_keyring = mpl
    .create_multi_keyring()
    .generator(generator_keyring)
    .child_keyrings(raw_aes_keyrings)
    .send()
    .await?;
```

Go

```
import (
    "context"

 mpl "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygenerated"
 mpltypes "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygeneratedtypes"
 client "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygenerated"
 esdktypes "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygeneratedtypes"
)

// Instantiate the AWS Encryption SDK client with limited encrypted data keys
encryptionClient, err := client.NewClient(esdktypes.AwsEncryptionSdkConfig{
    MaxEncryptedDataKeys: &maxEncryptedDataKeys,
})
if err != nil {
```

```
    panic(err)
}

// Define the key namespace and key name
var keyNamespace = "HSM_01"
var keyName = "RSA_2048_06"

// Instantiate the material providers library
matProv, err := mpl.NewClient(mpltypes.MaterialProvidersConfig{})
if err != nil {
    panic(err)
}

//  Generate `maxEncryptedDataKeys` raw AES keyrings to use with your keyring
rawAESKeyrings := make([]mpltypes.IKeyring, 0, maxEncryptedDataKeys)
var i int64 = 0
for i < maxEncryptedDataKeys {
    key, err := generate256KeyBytesAES()
 if err != nil {
  panic(err)
 }
    aesKeyRingInput := mpltypes.CreateRawAesKeyringInput{
  KeyName:      keyName,
  KeyNamespace: keyNamespace,
  WrappingKey:  key,
  WrappingAlg:  mpltypes.AesWrappingAlgAlgAes256GcmIv12Tag16,
 }
 aesKeyring, err := matProv.CreateRawAesKeyring(context.Background(),
 aesKeyRingInput)
    if err != nil {
  panic(err)
 }
    rawAESKeyrings = append(rawAESKeyrings, aesKeyring)
    i++
}

// Create a Multi Keyring with `max_encrypted_data_keys` AES Keyrings
createMultiKeyringInput := mpltypes.CreateMultiKeyringInput{
    Generator:     rawAESKeyrings[0],
    ChildKeyrings: rawAESKeyrings[1:],
}
multiKeyring, err := matProv.CreateMultiKeyring(context.Background(),
 createMultiKeyringInput)
if err != nil {
```

```
    panic(err)
}
```

Crear un filtro de detección

Al descifrar datos cifrados con claves KMS, se recomienda descifrarlos en modo estricto, es decir, limitar las claves de encapsulamiento utilizadas solo a las que especifique. Sin embargo, si es necesario, también puede descifrar en el modo de detección, en el que no se especifica ninguna clave de encapsulamiento. En este modo, AWS KMS puede descifrar la clave de datos cifrada con la clave de KMS que la cifró, independientemente de quién sea el propietario o tenga acceso a esa clave de KMS.

Si debe descifrar en modo de descubrimiento, le recomendamos que utilice siempre un filtro de descubrimiento, que limite las claves de KMS que se pueden usar a las de una partición Cuenta de AWS AND específica. El filtro de detección es opcional, pero es una práctica recomendada.

Utilice la siguiente tabla para determinar el valor de partición de su filtro de detección.

Region	Partición
Regiones de AWS	`aws`
Regiones de China	`aws-cn`
AWS GovCloud (US) Regions	`aws-us-gov`

En los ejemplos de esta sección se muestra cómo crear un filtro de detección. Antes de usar el código, sustituya los valores de ejemplo por valores válidos para la partición Cuenta de AWS and.

C

Para ver ejemplos completos, consulte kms_discovery.cpp en el SDK de cifrado de AWS para 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).Buil
```

C# / .NET

Para ver un ejemplo completo, consulte DiscoveryFilterExample.cs en el AWS Encryption SDK archivo.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

Para ver un ejemplo completo, consulte DiscoveryDecryptionExample.java en. SDK de cifrado de AWS para Java

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

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

JavaScript (Node and Browser)

Para ver ejemplos completos, consulte kms_filtered_discovery.ts (Node.js) y kms_multi_region_discovery.ts (Navegador) en el SDK de cifrado de AWS para JavaScript.

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

Python

Para ver un ejemplo completo, consulte discovery_kms_provider.py en el SDK de cifrado de AWS para Python.

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

Rust

```
let discovery_filter = DiscoveryFilter::builder()
    .account_ids(vec![111122223333.to_string()])
    .partition("aws".to_string())
    .build()?;
```

Go

```
import (
 mpltypes "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygeneratedtypes"
)

discoveryFilter := mpltypes.DiscoveryFilter{
    AccountIds: []string{111122223333},
    Partition:  "aws",
}
```

Configuración del contexto de cifrado requerido (CMM)

Puede usar el contexto de cifrado CMM requerido para requerir contextos de cifrado en sus operaciones criptográficas. Un contexto de cifrado es un conjunto de pares de clave-valor no secretos. El contexto de cifrado se vincula criptográficamente a los datos cifrados, de tal forma que se requiere el mismo contexto de cifrado para descifrar los datos. Cuando utiliza el contexto de cifrado CMM requerido, puede especificar una o más claves de contexto de cifrado obligatorias (claves obligatorias) que deben incluirse en todas las llamadas de cifrado y descifrado.

> **Note**
>
> El contexto de cifrado requerido (CMM) solo es compatible con las siguientes versiones:
>
> - Versión 3. x del SDK de cifrado de AWS para Java
> - Versión 4. x del AWS Encryption SDK para .NET
> - Versión 4. x de SDK de cifrado de AWS para Python, cuando se usa con la dependencia opcional de la biblioteca de proveedores de material criptográfico (MPL).
> - Versión 0.1. x o posterior del AWS Encryption SDK for Go
>
> Si cifra los datos mediante el contexto de cifrado CMM requerido, solo podrá descifrarlos con una de estas versiones compatibles.

Al cifrar, AWS Encryption SDK comprueba que todas las claves de contexto de cifrado necesarias estén incluidas en el contexto de cifrado que especificó. Firma AWS Encryption SDK los contextos de cifrado que especificó. Solo los pares clave-valor que no son claves obligatorias se serializan y almacenan en texto no cifrado en el encabezado del mensaje cifrado que devuelve la operación de cifrado.

Al descifrar, debe proporcionar un contexto de cifrado que contenga todos los pares clave-valor que representan las claves necesarias. AWS Encryption SDK Utiliza este contexto de cifrado y los pares clave-valor almacenados en el encabezado del mensaje cifrado para reconstruir el contexto de cifrado original que especificó en la operación de cifrado. Si la AWS Encryption SDK no puede reconstruir el contexto de cifrado original, se produce un error en la operación de descifrado. Si

proporciona un par clave-valor que contiene la clave requerida con un valor incorrecto, el mensaje cifrado no se puede descifrar. Debe proporcionar el mismo par clave-valor que se especificó al cifrar.

> **Important**
>
> Considere detenidamente qué valores elige para las claves requeridas en su contexto de cifrado. Debe poder volver a proporcionar las mismas claves y sus valores correspondientes al descifrar. Si no puede reproducir las claves requeridas, el mensaje cifrado no se podrá descifrar.

En los ejemplos siguientes se inicializa un conjunto de AWS KMS claves con el contexto de cifrado CMM requerido.

C# / .NET

```
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);
```

Java

```
// 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()
        );
```

Python

Para utilizar la CMM SDK de cifrado de AWS para Python con el contexto de cifrado requerido, también debe utilizar la biblioteca de proveedores de materiales (MPL).

```
# Instantiate the AWS Encryption SDK client
client = aws_encryption_sdk.EncryptionSDKClient(
    commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

# Create your encryption context
encryption_context: Dict[str, str] = {
    "key1": "value1",
    "key2": "value2",
    "requiredKey1": "requiredValue1",
    "requiredKey2": "requiredValue2"
}

# Create a list of required encryption context keys
required_encryption_context_keys: List[str] = ["requiredKey1", "requiredKey2"]

# Instantiate the material providers library
mpl: AwsCryptographicMaterialProviders = AwsCryptographicMaterialProviders(
    config=MaterialProvidersConfig()
)

# Create the AWS KMS keyring
 keyring_input: CreateAwsKmsKeyringInput = CreateAwsKmsKeyringInput(
    kms_key_id=kms_key_id,
    kms_client=boto3.client('kms', region_name="us-west-2")
)
kms_keyring: IKeyring = mpl.create_aws_kms_keyring(keyring_input)
```

```
# Create the required encryption context CMM
underlying_cmm: ICryptographicMaterialsManager = \
    mpl.create_default_cryptographic_materials_manager(
        CreateDefaultCryptographicMaterialsManagerInput(
            keyring=kms_keyring
        )
    )

required_ec_cmm: ICryptographicMaterialsManager = \
    mpl.create_required_encryption_context_cmm(
        CreateRequiredEncryptionContextCMMInput(
            required_encryption_context_keys=required_encryption_context_keys,
            underlying_cmm=underlying_cmm,
        )
    )
```

Rust

```
// Instantiate the AWS Encryption SDK client
let esdk_config = AwsEncryptionSdkConfig::builder().build()?;
let esdk_client = esdk_client::Client::from_conf(esdk_config)?;

// Create an AWS KMS client
let sdk_config =
 aws_config::load_defaults(aws_config::BehaviorVersion::latest()).await;
let kms_client = aws_sdk_kms::Client::new(&sdk_config);

// Create your encryption context
let encryption_context = HashMap::from([
    ("key1".to_string(), "value1".to_string()),
    ("key2".to_string(), "value2".to_string()),
    ("requiredKey1".to_string(), "requiredValue1".to_string()),
    ("requiredKey2".to_string(), "requiredValue2".to_string()),
]);

// Create a list of required encryption context keys
let required_encryption_context_keys: Vec<String> = vec![
    "requiredKey1".to_string(),
    "requiredKey2".to_string(),
];

// Instantiate the material providers library
```

```
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;

// Create the AWS KMS keyring
let kms_keyring = mpl
    .create_aws_kms_keyring()
    .kms_key_id(kms_key_id)
    .kms_client(kms_client)
    .send()
    .await?;

kms_multi_keyring: IKeyring = mat_prov.create_aws_kms_multi_keyring(
    input=kms_multi_keyring_input
)

// Create the required encryption context CMM
let underlying_cmm = mpl
    .create_default_cryptographic_materials_manager()
    .keyring(kms_keyring)
    .send()
    .await?;

let required_ec_cmm = mpl
    .create_required_encryption_context_cmm()
    .underlying_cmm(underlying_cmm.clone())
    .required_encryption_context_keys(required_encryption_context_keys)
    .send()
    .await?;
```

Go

```
import (
    "context"

 mpl "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygenerated"
 mpltypes "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygeneratedtypes"
 client "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygenerated"
 esdktypes "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygeneratedtypes"
    "github.com/aws/aws-sdk-go-v2/config"
```

```
    "github.com/aws/aws-sdk-go-v2/service/kms"
)

// Instantiate the AWS Encryption SDK client
encryptionClient, err := client.NewClient(esdktypes.AwsEncryptionSdkConfig{})
if err != nil {
    panic(err)
}

// Create an AWS KMS client
cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    panic(err)
}
kmsClient := kms.NewFromConfig(cfg, func(o *kms.Options) {
    o.Region = defaultKmsKeyRegion
})

// Create an encryption context
encryptionContext := map[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",
}

// Create a list of required encryption context keys
requiredEncryptionContextKeys := []string{}
requiredEncryptionContextKeys = append(requiredEncryptionContextKeys,
 "requiredKey1", "requiredKey2")

// Instantiate the material providers library
matProv, err := mpl.NewClient(mpltypes.MaterialProvidersConfig{})
if err != nil {
    panic(err)
}

// Create the AWS KMS keyring
awsKmsKeyringInput := mpltypes.CreateAwsKmsKeyringInput{
    KmsClient: kmsClient,
    KmsKeyId:  utils.GetDefaultKMSKeyId(),
}
```

```
awsKmsKeyring, err := matProv.CreateAwsKmsKeyring(context.Background(),
 awsKmsKeyringInput)
if err != nil {
    panic(err)
}

// Create the required encryption context CMM
underlyingCMM, err :=
 matProv.CreateDefaultCryptographicMaterialsManager(context.Background(),
 mpltypes.CreateDefaultCryptographicMaterialsManagerInput{Keyring: awsKmsKeyring})
if err != nil {
    panic(err)
}
requiredEncryptionContextInput := mpltypes.CreateRequiredEncryptionContextCMMInput{
    UnderlyingCMM: underlyingCMM,
    RequiredEncryptionContextKeys: requiredEncryptionContextKeys,
}
requiredEC, err := matProv.CreateRequiredEncryptionContextCMM(context.Background(),
 requiredEncryptionContextInput)
if err != nil {
    panic(err)
}
```

Establecer una política de compromiso

Una política de compromiso es una configuración que determina si su aplicación cifra y descifra con compromiso clave. Cifrar y descifrar los datos con un compromiso clave es una práctica recomendada de AWS Encryption SDK.

Establecer y ajustar la política de compromisos es un paso fundamental a la hora de migrar desde las versiones 1.7.x y anteriores del AWS Encryption SDK a la versión 2.0.x y posteriores. Esta progresión se explica en detalle en el tema de migración.

El valor predeterminado de la política de compromiso en las últimas versiones del AWS Encryption SDK (a partir de la versión 2.0.x), `RequireEncryptRequireDecrypt`, es ideal para la mayoría de las situaciones. Sin embargo, si necesita descifrar un texto cifrado que se cifró sin compromiso clave, puede que tenga que cambiar su política de compromiso a. `RequireEncryptAllowDecrypt` Para ver ejemplos de cómo establecer una política de compromiso en cada lenguaje de programación, consulte Establecer su política de compromiso.

Trabajo con datos de streaming

Cuando transmita datos para descifrarlos, tenga en cuenta que AWS Encryption SDK devuelven el texto sin formato descifrado una vez finalizadas las comprobaciones de integridad, pero antes de comprobar la firma digital. Para asegurarse de no devolver ni utilizar texto no cifrado hasta que se compruebe la firma, le recomendamos que almacene en búfer el texto no cifrado transmitido hasta que se complete todo el proceso de descifrado.

Este problema surge únicamente cuando se transmite texto cifrado para su descifrado y solo cuando se utiliza un conjunto de algoritmos, como el conjunto de algoritmos predeterminado, que incluye firmas digitales.

Para facilitar el almacenamiento en búfer, algunas implementaciones AWS Encryption SDK lingüísticas, como SDK de cifrado de AWS para JavaScript en Node.js, incluyen una función de almacenamiento en búfer como parte del método de descifrado. La CLI de cifrado de AWS , que siempre transmite entradas y salidas, introdujo un parámetro `--buffer` en las versiones 1.9.x y 2.2.x. En las implementaciones de otros lenguajes, puede utilizar las características de almacenamiento en búfer existentes. (La versión AWS Encryption SDK para .NET no admite la transmisión por secuencias).

Si utiliza un conjunto de algoritmos sin firmas digitales, asegúrese de utilizar la característica de `decrypt-unsigned` en cada implementación lingüística. Esta característica descifra el texto cifrado, pero no lo consigue si encuentra texto cifrado firmado. Para obtener más información, consulte Elección de un conjunto de algoritmos.

Claves de datos de almacenamiento en caché

En general, no se recomienda reutilizar las claves de datos, pero AWS Encryption SDK ofrece una opción de almacenamiento en caché de claves de datos que permite una reutilización limitada de las claves de datos. El almacenamiento en caché de las claves de datos puede mejorar el rendimiento de algunas aplicaciones y reducir las llamadas a su infraestructura clave. Antes de utilizar el almacenamiento en caché de claves de datos en producción, ajuste los umbrales de seguridad y compruebe que las ventajas superan las desventajas de la reutilización de las claves de datos.

Tiendas clave en el AWS Encryption SDK

En el AWS Encryption SDK, un almacén de claves es una tabla de Amazon DynamoDB que conserva los datos jerárquicos utilizados por el conjunto de claves jerárquicas.AWS KMS El almacén de claves ayuda a reducir el número de llamadas que necesita realizar para realizar operaciones criptográficas con el anillo AWS KMS de claves jerárquico.

El almacén de claves conserva y administra las claves de ramificación que el conjunto de claves jerárquico utiliza para cifrar sobres y proteger las claves de cifrado de datos. El almacén de claves almacena la clave de rama activa y todas las versiones anteriores de la clave de rama. La clave de rama activa es la versión más reciente de la clave de rama. El conjunto de claves jerárquico utiliza una clave de cifrado de datos única para cada solicitud de cifrado y cifra cada clave de cifrado de datos con una clave de empaquetado única derivada de la clave de sucursal activa. El conjunto de claves jerárquico depende de la jerarquía establecida entre las claves de ramificación activas y las claves de encapsulamiento derivadas.

La terminología y los conceptos del almacén de claves

Almacén de claves

: La tabla de DynamoDB que conserva los datos jerárquicos, como las claves de bifurcación y las claves de baliza.

Clave raíz

: Una clave KMS de cifrado simétrico que genera y protege las claves de sucursal y baliza del almacén de claves.

Clave de sucursal

: Clave de datos que se reutiliza para obtener una clave de empaquetado única para el cifrado de sobres. Puede crear varias claves de rama en un almacén de claves, pero cada clave de rama solo puede tener una versión de clave de rama activa a la vez. La clave de rama activa es la versión más reciente de la clave de rama.

  Las claves de bifurcación se derivan del AWS KMS keys uso de la GenerateDataKeyWithoutPlaintext operación kms:.

Clave de encapsulamiento

Una clave de datos única que se utiliza para cifrar la clave de cifrado de datos que se utiliza en las operaciones de cifrado.

Las claves de empaquetado se derivan de las claves de ramificación. Para obtener más información sobre el proceso de obtención de claves, consulte los detalles técnicos del conjunto de claves AWS KMS jerárquicas.

Clave de cifrado de datos

Clave de datos que se utiliza en las operaciones de cifrado. El conjunto de claves jerárquico utiliza una clave de cifrado de datos única para cada solicitud de cifrado.

Implementación de permisos de privilegio mínimo

Cuando utilice un almacén de claves y anillos de claves AWS KMS jerárquicos, le recomendamos que siga el principio del privilegio mínimo definiendo las siguientes funciones:

Administrador del almacén de claves

Los administradores del almacén de claves son responsables de crear y administrar el almacén de claves y las claves de sucursal que conserva y protege. Los administradores del almacén de claves deben ser los únicos usuarios con permisos de escritura en la tabla de Amazon DynamoDB que sirve como almacén de claves. Deben ser los únicos usuarios con acceso a operaciones de administrador privilegiadas, como `CreateKey`y. `VersionKey` Solo puede realizar estas operaciones si configura estáticamente las acciones del almacén de claves.

`CreateKey`es una operación privilegiada que puede añadir un nuevo ARN de clave de KMS a su lista de almacenes de claves permitidos. Esta clave KMS puede crear nuevas claves de rama activas. Recomendamos limitar el acceso a esta operación porque, una vez que se agrega una clave KMS al almacén de claves de la sucursal, no se puede eliminar.

Usuario del almacén de claves

En la mayoría de los casos de uso, el usuario del almacén de claves solo interactúa con el almacén de claves a través del anillo de claves jerárquico mientras cifra, descifra, firma y verifica los datos. Como resultado, solo necesitan permisos de lectura para la tabla de Amazon DynamoDB que sirve como almacén de claves. Los usuarios del almacén de claves solo deberían necesitar acceder a las operaciones de uso que posibilitan las operaciones criptográficas,

como`GetActiveBranchKey`, y`GetBranchKeyVersion`. `GetBeaconKey` No necesitan permisos para crear o administrar las claves de sucursal que utilizan.

Puedes realizar operaciones de uso cuando las acciones de tu almacén de claves estén configuradas de forma estática o cuando estén configuradas para su detección. No puede realizar operaciones de administrador (ni`VersionKey`) cuando `CreateKey` las acciones del almacén de claves están configuradas para su descubrimiento.

Si el administrador del almacén de claves de la sucursal ha incluido varias claves de KMS en el almacén de claves de la sucursal, se recomienda que los usuarios del almacén de claves configuren las acciones del almacén de claves para su detección, de modo que su conjunto de claves jerárquico pueda utilizar varias claves de KMS.

Crear un almacén de claves

Antes de poder crear claves de rama o utilizar un conjunto de claves AWS KMS jerárquico, debe crear su almacén de claves, una tabla de Amazon DynamoDB que gestione y proteja las claves de rama.

> **Important**
>
> No elimine la tabla de DynamoDB en la que se conservan las claves de rama. Si elimina esta tabla, no podrá descifrar ningún dato cifrado con el anillo de claves jerárquico.

Siga los procedimientos de creación de una tabla de la Guía para desarrolladores de Amazon DynamoDB y utilice los siguientes valores de cadena obligatorios para la clave de partición y la clave de clasificación.

	Clave de partición	Clave de clasificación
Tabla base	`branch-key-id`	`type`

Nombre del almacén de claves lógicas

Al asignar un nombre a la tabla de DynamoDB que sirve como almacén de claves, es importante tener en cuenta el nombre del almacén de claves lógico que especificará al configurar las acciones del almacén de claves. El nombre del almacén de claves lógico actúa como identificador del almacén de claves y no se puede cambiar una vez que el primer usuario lo haya definido inicialmente. Debe especificar siempre el mismo nombre de almacén de claves lógico en las acciones del almacén de claves.

Debe haber una one-to-one correlación entre el nombre de la tabla de DynamoDB y el nombre del almacén de claves lógicas. El nombre del almacén de claves lógico está enlazado criptográficamente a todos los datos almacenados en la tabla para simplificar las operaciones de restauración de DynamoDB. Si bien el nombre del almacén de claves lógicas puede ser diferente del nombre de la tabla de DynamoDB, se recomienda encarecidamente especificar el nombre de la tabla de DynamoDB como nombre del almacén de claves lógicas. En caso de que el nombre de la tabla cambie después de restaurar la tabla de DynamoDB a partir de una copia de seguridad, el nombre del almacén de claves lógicas se puede asignar al nombre de la nueva tabla de DynamoDB para garantizar que el anillo de claves jerárquico pueda seguir accediendo al almacén de claves.

No incluya información confidencial o delicada en el nombre del almacén de claves lógico. El nombre del almacén de claves lógico se muestra en texto plano en AWS KMS CloudTrail eventos como. `tablename`

Siguientes pasos

1. the section called “Configurar las acciones del almacén de claves”
2. the section called “Crea claves de rama”
3. Cree un conjunto de claves AWS KMS jerárquico

Configurar las acciones del almacén de claves

Las acciones del almacén de claves determinan qué operaciones pueden realizar los usuarios y cómo su conjunto de claves AWS KMS jerárquico utiliza las claves KMS permitidas incluidas en su almacén de claves. AWS Encryption SDK Admite las siguientes configuraciones de acciones de almacenamiento de claves.

Estático

> Al configurar el almacén de claves de forma estática, el almacén de claves solo puede usar la clave de KMS asociada al ARN de clave de KMS que se proporciona al configurar

`kmsConfiguration` el almacén de claves. Se produce una excepción si se encuentra un ARN de clave de KMS diferente al crear, versionar u obtener una clave de rama.

Puede especificar una clave de KMS multirregional en su nombre`kmsConfiguration`, pero todo el ARN de la clave, incluida la región, se conserva en las claves de rama derivadas de la clave de KMS. No puedes especificar una clave en una región diferente, debes proporcionar exactamente la misma clave multirregional para que los valores coincidan.

Al configurar de forma estática las acciones del almacén de claves, puede realizar operaciones de uso (`GetActiveBranchKey``GetBranchKeyVersion`,`GetBeaconKey`) y operaciones administrativas (`CreateKey``VersionKey`). `CreateKey`es una operación privilegiada que puede añadir un nuevo ARN de clave de KMS a su lista de almacenes de claves permitidos. Esta clave KMS puede crear nuevas claves de rama activas. Recomendamos limitar el acceso a esta operación porque, una vez que se agrega una clave KMS al almacén de claves, no se puede eliminar.

Discovery

Al configurar las acciones del almacén de claves para la detección, el almacén de claves puede usar cualquier AWS KMS key ARN que esté incluido en la lista de permitidos del almacén de claves. Sin embargo, se produce una excepción cuando se encuentra una clave KMS multirregional y la región del ARN de la clave no coincide con la región AWS KMS del cliente que se está utilizando.

Al configurar el almacén de claves para la detección, no puede realizar operaciones administrativas, como `CreateKey` y. `VersionKey` Solo puede realizar las operaciones de uso que permiten las operaciones de cifrado, descifrado, firma y verificación. Para obtener más información, consulte the section called “Implementación de permisos de privilegio mínimo”.

Configure las acciones de su almacén de claves

Antes de configurar las acciones del almacén de claves, asegúrese de que se cumplen los siguientes requisitos previos.

- Determine qué operaciones debe realizar. Para obtener más información, consulte the section called “Implementación de permisos de privilegio mínimo”.
- Elija un nombre de almacén de claves lógico

Debe haber una one-to-one correlación entre el nombre de la tabla de DynamoDB y el nombre del almacén de claves lógicas. El nombre del almacén de claves lógico está enlazado criptográficamente a todos los datos almacenados en la tabla para simplificar las operaciones de restauración de DynamoDB. No se puede cambiar una vez que el primer usuario lo haya definido inicialmente. Debe especificar siempre el mismo nombre de almacén de claves lógico en las acciones del almacén de claves. Para obtener más información, consulte logical key store name.

Configuración estática

El siguiente ejemplo configura estáticamente las acciones del almacén de claves. Debe especificar el nombre de la tabla de DynamoDB que sirve como almacén de claves, un nombre lógico para el almacén de claves y el ARN de clave KMS que identifica una clave KMS de cifrado simétrico.

> **Note**
>
> Tenga en cuenta detenidamente el ARN de la clave de KMS que especifique al configurar de forma estática el servicio de almacén de claves. La `CreateKey` operación agrega el ARN de la clave KMS a la lista de permitidos del almacén de claves de la sucursal. Una vez que se agrega una clave KMS al almacén de claves de la sucursal, no se puede eliminar.

Java

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

C# / .NET

```
var kmsConfig = new KMSConfiguration { KmsKeyArn = kmsKeyArn };
  var keystoreConfig = new KeyStoreConfig
```

```
  {
      KmsClient = new AmazonKeyManagementServiceClient(),
      KmsConfiguration = kmsConfig,
      DdbTableName = keyStoreName,
      DdbClient = new AmazonDynamoDBClient(),
      LogicalKeyStoreName = logicalKeyStoreName
  };
  var keystore = new KeyStore(keystoreConfig);
```

Python

```
keystore: KeyStore = KeyStore(
    config=KeyStoreConfig(
        ddb_client=ddb_client,
        ddb_table_name=key_store_name,
        logical_key_store_name=logical_key_store_name,
        kms_client=kms_client,
        kms_configuration=KMSConfigurationKmsKeyArn(
            value=kms_key_id
        ),
    )
)
```

Rust

```
let sdk_config =
 aws_config::load_defaults(aws_config::BehaviorVersion::latest()).await;
let key_store_config = KeyStoreConfig::builder()
    .kms_client(aws_sdk_kms::Client::new(&sdk_config))
    .ddb_client(aws_sdk_dynamodb::Client::new(&sdk_config))
    .ddb_table_name(key_store_name)
    .logical_key_store_name(logical_key_store_name)
    .kms_configuration(KmsConfiguration::KmsKeyArn(kms_key_arn.to_string()))
    .build()?;

let keystore = keystore_client::Client::from_conf(key_store_config)?;
```

Go

```
import (
 keystore "github.com/aws/aws-cryptographic-material-providers-library/mpl/
awscryptographykeystoresmithygenerated"
```

```
 keystoretypes "github.com/aws/aws-cryptographic-material-providers-library/mpl/
awscryptographykeystoresmithygeneratedtypes"
)

kmsConfig := keystoretypes.KMSConfigurationMemberkmsKeyArn{
    Value: kmsKeyArn,
}
keyStore, err := keystore.NewClient(keystoretypes.KeyStoreConfig{
    DdbTableName:        keyStoreTableName,
    KmsConfiguration:    &kmsConfig,
    LogicalKeyStoreName: logicalKeyStoreName,
    DdbClient:           ddbClient,
    KmsClient:           kmsClient,
})
if err != nil {
    panic(err)
}
```

Configuración de detección

En el siguiente ejemplo, se configuran las acciones del almacén de claves para su detección. Debe especificar el nombre de la tabla de DynamoDB que sirve como almacén de claves y un nombre de almacén de claves lógico.

Java

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

C# / .NET

```
var keystoreConfig = new KeyStoreConfig
 {
     KmsClient = new AmazonKeyManagementServiceClient(),
```

```
    KmsConfiguration = new KMSConfiguration {Discovery = new Discovery()},
    DdbTableName = keyStoreName,
    DdbClient = new AmazonDynamoDBClient(),
    LogicalKeyStoreName = logicalKeyStoreName
};
var keystore = new KeyStore(keystoreConfig);
```

Python

```
keystore: KeyStore = KeyStore(
    config=KeyStoreConfig(
        ddb_client=ddb_client,
        ddb_table_name=key_store_name,
        logical_key_store_name=logical_key_store_name,
        kms_client=kms_client,
        kms_configuration=KMSConfigurationDiscovery(
            value=Discovery()
        ),
    )
)
```

Rust

```
let key_store_config = KeyStoreConfig::builder()
        .kms_client(kms_client)
        .ddb_client(ddb_client)
        .ddb_table_name(key_store_name)
        .logical_key_store_name(logical_key_store_name)

 .kms_configuration(KmsConfiguration::Discovery(Discovery::builder().build()?))
        .build()?;
```

Go

```
import (
 keystore "github.com/aws/aws-cryptographic-material-providers-library/mpl/
awscryptographykeystoresmithygenerated"
 keystoretypes "github.com/aws/aws-cryptographic-material-providers-library/mpl/
awscryptographykeystoresmithygeneratedtypes"
)

kmsConfig := keystoretypes.KMSConfigurationMemberdiscovery{}
```

```
keyStore, err := keystore.NewClient(keystoretypes.KeyStoreConfig{
    DdbTableName:        keyStoreName,
    KmsConfiguration:    &kmsConfig,
    LogicalKeyStoreName: logicalKeyStoreName,
    DdbClient:           ddbClient,
    KmsClient:           kmsClient,
})
if err != nil {
    panic(err)
}
```

Crear una clave de rama activa

Una clave de sucursal es una clave de datos derivada de una AWS KMS key que el conjunto de claves AWS KMS jerárquico utiliza para reducir el número de llamadas realizadas. AWS KMS La clave de rama activa es la versión más reciente de la clave de rama. El conjunto de claves jerárquico genera una clave de datos única para cada solicitud de cifrado y cifra cada clave de datos con una clave de empaquetado única derivada de la clave de rama activa.

Para crear una nueva clave de rama activa, debe configurar de forma estática las acciones del almacén de claves. `CreateKeyes` una operación privilegiada que agrega el ARN de clave KMS especificado en la configuración de acciones del almacén de claves a la lista de permitidos del almacén de claves. A continuación, la clave KMS se utiliza para generar la nueva clave de rama activa. Recomendamos limitar el acceso a esta operación porque, una vez que se agrega una clave KMS al almacén de claves, no se puede eliminar.

Puede permitir incluir una clave de KMS en su almacén de claves o puede incluir varias claves de KMS actualizando el ARN de la clave de KMS que especifique en la configuración de acciones de su almacén de claves y volviendo a llamar. `CreateKey` Si permite incluir en la lista varias claves de KMS, los usuarios de su almacén de claves deben configurar sus acciones de almacenamiento de claves para su detección, de modo que puedan usar cualquiera de las claves permitidas del almacén de claves al que tienen acceso. Para obtener más información, consulte the section called “Configurar las acciones del almacén de claves”.

Permisos necesarios

Para crear claves de rama, necesitas los ReEncrypt permisos kms: GenerateDataKeyWithoutPlaintext y kms: en la clave KMS especificada en las acciones de tu almacén de claves.

Crea una clave de rama

La siguiente operación crea una nueva clave de rama activa con la clave KMS que especificó en la configuración de acciones del almacén de claves y agrega la clave de rama activa a la tabla de DynamoDB que sirve como almacén de claves.

Al llamar a `CreateKey`, puede optar por especificar los siguientes valores opcionales.

- `branchKeyIdentifier`: define una `branch-key-id` personalizada.

  Para crear una `branch-key-id` personalizada, también debe incluir un contexto de cifrado adicional con el parámetro `encryptionContext`.
- `encryptionContext`: define un conjunto opcional de pares clave-valor no secretos que proporciona datos autenticados (AAD) adicionales en el contexto de cifrado incluido en la llamada kms:. GenerateDataKeyWithoutPlaintext

  Este contexto de cifrado adicional se muestra con el prefijo `aws-crypto-ec:`.

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();
```

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
```

```
  });
```

Python

```
additional_encryption_context = {"Additional Encryption Context for": "custom branch
 key id"}

branch_key_id: str = keystore.create_key(
    CreateKeyInput(
        branch_key_identifier = "custom-branch-key-id", # OPTIONAL
        encryption_context = additional_encryption_context, # OPTIONAL
    )
)
```

Rust

```
let additional_encryption_context = HashMap::from([
    ("Additional Encryption Context for".to_string(), "custom branch key
 id".to_string())
]);

let branch_key_id = keystore.create_key()
    .branch_key_identifier("custom-branch-key-id") // OPTIONAL
    .encryption_context(additional_encryption_context) // OPTIONAL
    .send()
    .await?
    .branch_key_identifier
    .unwrap();
```

Go

```
encryptionContext := map[string]string{
    "Additional Encryption Context for": "custom branch key id",
}

branchKey, err := keyStore.CreateKey(context.Background(),
 keystoretypes.CreateKeyInput{
    BranchKeyIdentifier: &customBranchKeyId,
    EncryptionContext:   additional_encryption_context,
})
if err != nil {
    return "", err
```

```
}
```

En primer lugar, la operación `CreateKey` genera los siguientes valores.

- Un identificador único universal (UUID) de la versión 4 para el `branch-key-id` (a menos que haya especificado una `branch-key-id` personalizada).
- Un UUID de la versión 4 para la versión de clave de rama
- Una `timestamp` en el formato de fecha y hora ISO 8601 en hora universal coordinada (UTC).

A continuación, la `CreateKey` operación llama a kms: GenerateDataKeyWithoutPlaintext mediante la siguiente solicitud.

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

A continuación, la `CreateKey` operación llama a kms: ReEncrypt para crear un registro activo para la clave de sucursal mediante la actualización del contexto de cifrado.

Por último, la `CreateKey` operación llama a ddb: TransactWriteItems para escribir un nuevo elemento que conserve la clave de rama en la tabla que creó en el paso 2. El objeto tiene los siguientes atributos:

```
{
      "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"
}
```

Rote la clave de rama activa

Solo puede haber una versión activa para cada clave de rama a la vez. Por lo general, cada versión de clave de rama activa se usa para satisfacer múltiples solicitudes. Sin embargo, usted controla el grado en que se reutilizan las claves de rama activas y determina la frecuencia con la que se gira la clave de rama activa.

Las claves de rama no se utilizan para cifrar claves de datos de texto simple. Se utilizan para obtener las claves de encapsulamiento únicas que cifran las claves de datos de texto no cifrado. El proceso de derivación de la clave de encapsulamiento produce una clave de encapsulamiento única de 32 bytes con 28 bytes de asignación al azar. Esto significa que una clave de ramificación puede obtener más de 79 octillones, o 2^{96}, claves de encapsulamiento únicas antes de que se produzca un desgaste criptográfico. A pesar de este riesgo de agotamiento muy bajo, es posible que deba rotar la clave de rama activa debido a reglas comerciales o contractuales o regulaciones gubernamentales.

La versión activa de la clave de rama permanece activa hasta que la rotes. Las versiones anteriores de la clave de ramificación activa no se utilizarán para realizar operaciones de cifrado ni para obtener nuevas claves de empaquetado, pero sí se pueden consultar y proporcionar claves de empaquetado para descifrar las claves de datos que cifraban mientras estaban activas.

Permisos necesarios

Para rotar las claves de ramificación, necesita los ReEncrypt permisos kms: GenerateDataKeyWithoutPlaintext y kms: en la clave KMS especificada en las acciones del almacén de claves.

Rota una clave de rama activa

Utilice la `VersionKey` operación para girar la clave de bifurcación activa. Al girar la clave de rama activa, se crea una nueva clave de rama para sustituir a la versión anterior. La `branch-key-id` no cambia al girar la clave de rama activa. Debe especificar la `branch-key-id` que identifica la clave de rama activa actual cuando llama a `VersionKey`.

Java

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

C# / .NET

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

Python

```
keystore.version_key(
    VersionKeyInput(
        branch_key_identifier=branch_key_id
    )
)
```

Rust

```
keystore.version_key()
        .branch_key_identifier(branch_key_id)
        .send()
        .await?;
```

Go

```
_, err = keyStore.VersionKey(context.Background(), keystoretypes.VersionKeyInput{
    BranchKeyIdentifier: branchKeyId,
})
if err != nil {
    return err
}
```

Conjuntos de claves

Las implementaciones de lenguajes de programación compatibles utilizan anillos de claves para realizar el cifrado de sobres. Los conjuntos de claves generan, cifran y descifran claves de datos. Según el que se use se determinará el origen de las claves de datos únicas que protegen cada mensaje y las claves de encapsulamiento que cifran dichas claves de datos. Al cifrar especifica un conjunto de claves y al descifrar usa ese mismo conjunto de claves u otro conjunto de claves. Puede utilizar los conjuntos de claves que proporciona SDK o escribir sus propios conjuntos de claves personalizados compatibles.

Puede utilizar cada conjunto de claves de forma individual o combinar conjuntos de claves en un conjunto de claves múltiple. Aunque la mayoría de los conjuntos de claves pueden generar, cifrar y descifrar claves de datos, podría crear un conjunto de claves que realice solo una operación particular, por ejemplo, un conjunto de claves que solo genere claves de datos y utilizar dicho conjunto de claves en combinación con otros.

Le recomendamos que utilice un anillo de claves que proteja las claves de empaquetado y realice operaciones criptográficas dentro de un límite seguro, como el anillo de claves, que utiliza ese AWS KMS anillo de claves AWS KMS keys que nunca deja () sin cifrar. AWS Key Management ServiceAWS KMS También puede crear un conjunto de claves que utilice claves empaquetadoras almacenadas en los módulos de seguridad del hardware (HSMs) o protegidas por otros servicios de claves maestras. Para obtener más información, consulte el tema que trata de la interfaz del conjunto de claves en la especificación de AWS Encryption SDK .

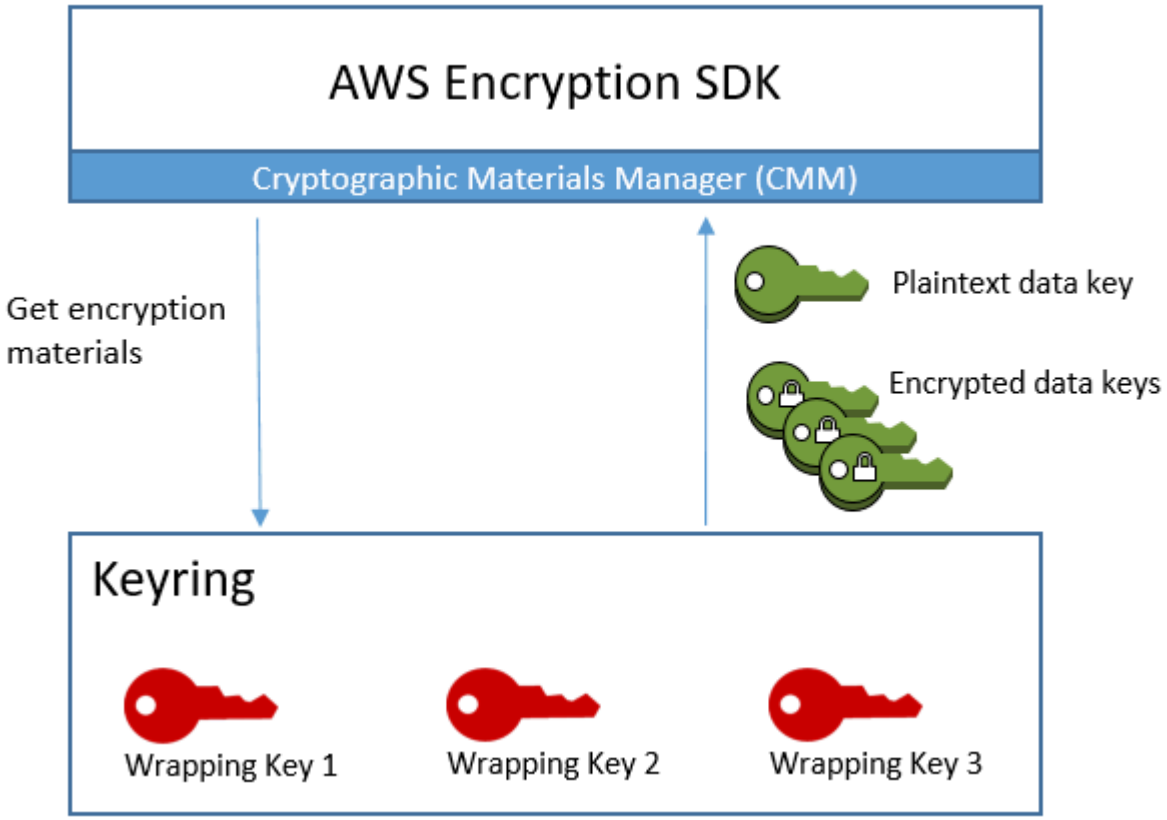
Los llaveros cumplen la función de claves maestras y proveedores de claves maestras que se utilizan en las implementaciones de otros lenguajes de programación. Si utiliza implementaciones de diferentes lenguajes AWS Encryption SDK para cifrar y descifrar sus datos, asegúrese de utilizar conjuntos de claves y proveedores de claves maestras compatibles. Para obtener más información, consulte Compatibilidad de conjuntos de claves.

En este tema se explica cómo utilizar la función de llavero del llavero AWS Encryption SDK y cómo elegir un llavero.

Cómo funcionan los conjuntos de claves

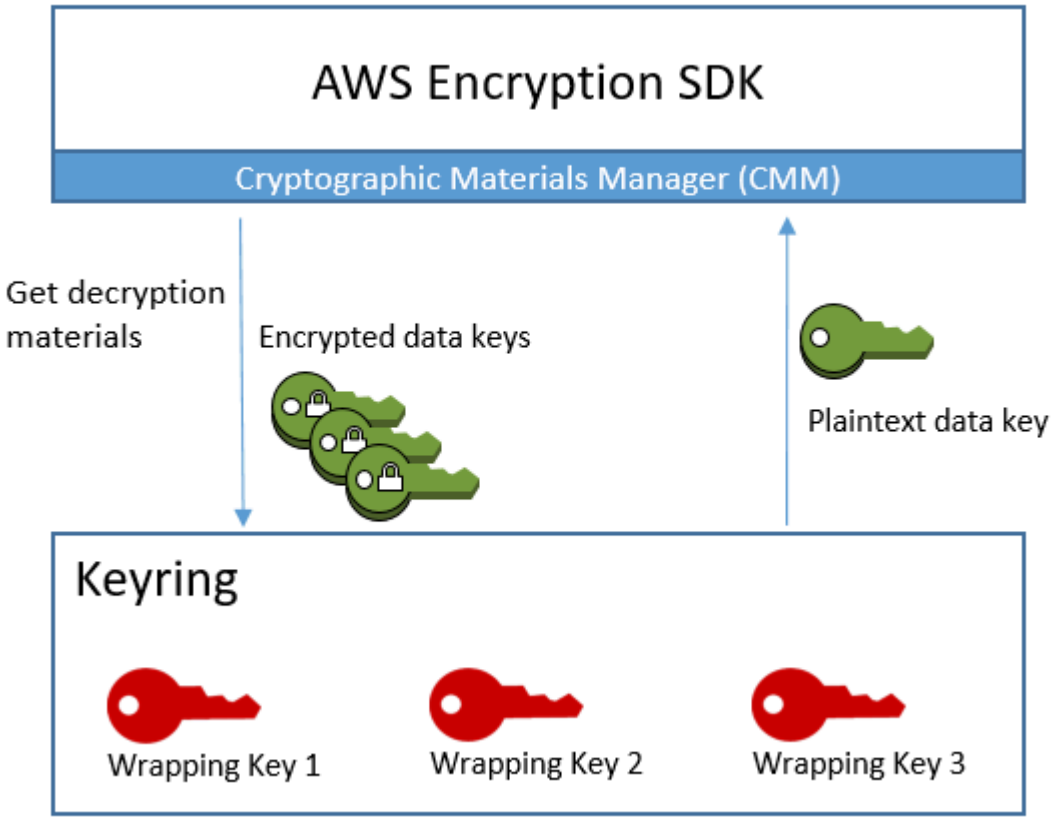
Al cifrar datos, AWS Encryption SDK solicita materiales de cifrado al conjunto de claves. Este devuelve una clave en texto no cifrado y una copia de dicha clave cifrada por cada una de las

claves de encapsulamiento del conjunto de claves. AWS Encryption SDK Utiliza la clave de texto sin formato para cifrar los datos y, a continuación, destruye la clave de datos de texto sin formato. A continuación, AWS Encryption SDK devuelve un mensaje cifrado que incluye las claves de datos cifrados y los datos cifrados.



Al descifrar datos, puede utilizar el mismo conjunto de claves que utilizó para cifrar los datos o uno diferente. Para descifrar los datos, un conjunto de claves de descifrado debe incluir (o tener acceso a) al menos una clave de encapsulamiento en el conjunto de claves de cifrado.

AWS Encryption SDK Pasa las claves de datos cifrados del mensaje cifrado al conjunto de claves y pide al conjunto de claves que descifre cualquiera de ellas. Este utiliza sus claves de encapsulamiento para descifrar una de las claves de datos cifradas y devuelve una clave de datos en texto no cifrado. El AWS Encryption SDK usa la clave en texto no cifrado para descifrar los datos. Si ninguna de las claves de encapsulamiento del conjunto de claves puede descifrar ninguna de las claves de datos cifradas, se producirá un error en la operación de descifrado.



Puede utilizar un único conjunto de claves o además combinar conjuntos de claves del mismo tipo o de un tipo distinto en un conjunto de claves múltiple. Al cifrar los datos, el conjunto de claves múltiple devuelve una copia de la clave de datos cifrada por todas las claves de encapsulamiento en todos los conjuntos de claves que componen el conjunto de claves múltiple. Puede descifrar los datos utilizando un conjunto de claves configurado con cualquiera de las claves de encapsulamiento del conjunto de claves múltiple.

Compatibilidad de conjuntos de claves

Si bien las diferentes implementaciones lingüísticas AWS Encryption SDK presentan algunas diferencias arquitectónicas, son totalmente compatibles y están sujetas a las restricciones del idioma. Puede cifrar los datos utilizando una implementación de lenguaje de programación y descifrarlos con cualquier otra implementación de lenguaje. Sin embargo, debe utilizar las mismas o claves maestras correspondientes para cifrar y descifrar las claves de datos. Para obtener información sobre las restricciones de idioma, consulte el tema sobre cada implementación lingüística, por ejemplo, the section called “Compatibilidad” en el SDK de cifrado de AWS para JavaScript tema.

Los llaveros se admiten en los siguientes lenguajes de programación:

- SDK de cifrado de AWS para C
- SDK de cifrado de AWS para JavaScript
- AWS Encryption SDK para .NET
- Versión 3. x del SDK de cifrado de AWS para Java
- Versión 4. x de SDK de cifrado de AWS para Python, cuando se usa con la dependencia opcional de la biblioteca de proveedores de material criptográfico (MPL).
- AWS Encryption SDK para Rust
- AWS Encryption SDK para Go

Diferentes requisitos para los conjuntos de claves de cifrado

En las implementaciones AWS Encryption SDK lingüísticas distintas del SDK de cifrado de AWS para C, todas las claves de empaquetado de un anillo de claves de cifrado (o conjunto de múltiples claves) o de un proveedor de claves maestras deben poder cifrar la clave de datos. Si alguna de las claves de encapsulamiento no se cifra, el método de cifrado falla. Como resultado, la persona que llama debe tener los permisos necesarios para todas las claves del conjunto de claves. Si utiliza un conjunto de claves de detección para cifrar los datos, solo o en un conjunto de claves múltiple, la operación de cifrado no se realizará correctamente.

La excepción es la siguiente SDK de cifrado de AWS para C, en la que la operación de cifrado ignora un anillo de claves de detección estándar, pero falla si se especifica un anillo de claves de descubrimiento multirregional, solo o en un anillo de claves múltiples.

Conjuntos de claves compatibles y proveedores de claves maestras

La siguiente tabla muestra qué claves maestras y proveedores de claves maestras son compatibles con los llaveros que suministra. AWS Encryption SDK Cualquier pequeña incompatibilidad por restricciones de lenguaje se explica en el tema que trata la implementación del lenguaje.

Conjunto de claves:	Proveedor de claves maestras:
AWS KMS llavero	KMSMasterClave (Java) KMSMasterKeyProvider (Java)

Conjunto de claves:	Proveedor de claves maestras:
	KMSMasterClave (Python) KMSMasterKeyProvider (Python) **Note** SDK de cifrado de AWS para Python Y SDK de cifrado de AWS para Java no incluyen una clave maestra o un proveedor de claves maestras que sea equivalente al conjunto de claves de descubrimiento AWS KMS regional.
AWS KMS Conjunto de claves jerárquico	Compatible con los siguientes lenguajes y versiones de programación: • Versión 3. x del SDK de cifrado de AWS para Java • Versión 4. x del AWS Encryption SDK para .NET • Versión 4. x de SDK de cifrado de AWS para Python, cuando se usa con la dependencia opcional de la biblioteca de proveedores de material criptográfico (MPL). • Versión 1. x de la AWS Encryption SDK para Rust • Versión 0.1. x o posterior del AWS Encryption SDK for Go
AWS KMS Llavero ECDH	Compatible con los siguientes lenguajes y versiones de programación: • Versión 3. x del SDK de cifrado de AWS para Java • Versión 4. x del AWS Encryption SDK para .NET • Versión 4. x de SDK de cifrado de AWS para Python, cuando se usa con la dependencia opcional de la biblioteca de proveedores de material criptográfico (MPL). • Versión 1. x de la AWS Encryption SDK para Rust • Versión 0.1. x o posterior del AWS Encryption SDK for Go

Conjunto de claves:	Proveedor de claves maestras:
Conjunto de claves de AES sin formato	Cuando se utilizan con claves de cifrado simétricas: JceMasterKey(Java) RawMasterKey(Python)
Conjunto de claves de RSA sin formato	Cuando se utilizan con claves de cifrado asimétricas: JceMasterKey(Java) RawMasterKey(Python) **Note** El conjunto de claves de RSA no admite claves de KMS asimétricas. Si desea utilizar claves asimétricas de RSA KMS, versión 4. x de la versión AWS Encryption SDK para .NET admite conjuntos de AWS KMS claves que utilizan cifrado simétrico (`SYMMETRIC_DEFAULT`) o RSA asimétrico. AWS KMS keys
Anillo de claves ECDH sin procesar	Compatible con los siguientes lenguajes y versiones de programación: • Versión 3. x del SDK de cifrado de AWS para Java • Versión 4. x del AWS Encryption SDK para .NET • Versión 4. x de SDK de cifrado de AWS para Python, cuando se usa con la dependencia opcional de la biblioteca de proveedores de material criptográfico (MPL). • Versión 1. x de la AWS Encryption SDK para Rust • Versión 0.1. x o posterior del AWS Encryption SDK for Go

AWS KMS llaveros

Un AWS KMS anillo de claves se utiliza AWS KMS keyspara generar, cifrar y descifrar claves de datos. AWS Key Management Service (AWS KMS) protege las claves de KMS y realiza operaciones

criptográficas dentro del límite del FIPS. Le recomendamos que utilice un conjunto de claves de AWS KMS o un conjunto de claves con propiedades de seguridad similares, siempre que sea posible.

Todas las implementaciones de lenguajes de programación que admiten conjuntos de claves admiten conjuntos de AWS KMS claves KMS de cifrado simétrico. Las siguientes implementaciones de lenguajes de programación también admiten conjuntos de AWS KMS claves RSA KMS asimétricas:

- Versión 3. x del SDK de cifrado de AWS para Java
- Versión 4. x del AWS Encryption SDK para .NET
- Versión 4. x de SDK de cifrado de AWS para Python, cuando se usa con la dependencia opcional de la biblioteca de proveedores de material criptográfico (MPL).
- Versión 1. x de la AWS Encryption SDK para Rust
- Versión 0.1. x o posterior del AWS Encryption SDK for Go

Si intenta incluir una clave de KMS asimétrica en un conjunto de claves de cifrado en cualquier implementación de otro lenguaje, se produce un error en la llamada de cifrado. Si la incluye en un conjunto de claves de descifrado, se ignora.

A partir de la versión 2.3, puede utilizar una clave AWS KMS multirregional en un AWS KMS conjunto de claves o en un proveedor de claves maestras. x de la AWS Encryption SDK y la versión 3.0. x de la CLI AWS de cifrado. Para obtener detalles y ejemplos del uso del multi-Region-aware símbolo, consulteUso de varias regiones AWS KMS keys. Para obtener más información sobre las claves de regiones múltiples, consulte Uso de claves de multirregiones en la Guía para desarrolladores de AWS Key Management Service .

> **Note**
>
> Todas las menciones a los llaveros KMS en el AWS Encryption SDK hacen referencia a los AWS KMS llaveros.

AWS KMS Los llaveros pueden incluir dos tipos de claves de embalaje:

- Clave generadora: genera una clave de datos en texto no cifrado y la cifra. Un conjunto de claves que cifra datos debe tener una clave generadora.

- Claves adicionales: cifra la clave de datos de texto sin formato que generó la clave del generador. AWS KMS los llaveros pueden tener cero o más claves adicionales.

Debe tener una clave generadora para cifrar los mensajes. Cuando un conjunto de AWS KMS claves tiene solo una clave KMS, esa clave se usa para generar y cifrar la clave de datos. Al descifrar, la clave generadora es opcional y se ignora la distinción entre claves generadoras y claves adicionales.

Como todos los llaveros, los AWS KMS llaveros se pueden utilizar de forma independiente o en un llavero múltiple con otros llaveros del mismo tipo o de un tipo diferente.

Temas

- AWS KMS Permisos necesarios para los llaveros
- Identificarse AWS KMS keys en un AWS KMS llavero
- Crear un anillo de claves AWS KMS
- Uso de un anillo de claves de detección AWS KMS
- Uso de un anillo de claves de detección AWS KMS regional

AWS KMS Permisos necesarios para los llaveros

AWS Encryption SDK No requiere ni depende de ninguno Servicio de AWS. Cuenta de AWS Sin embargo, para usar un AWS KMS llavero, necesitas tener Cuenta de AWS los siguientes permisos mínimos AWS KMS keys en el llavero.

- Para cifrar con un AWS KMS anillo de claves, necesita el GenerateDataKey permiso kms: en la clave del generador. Necesita el permiso KMS:Encrypt para todas las claves adicionales del anillo de claves. AWS KMS
- Para descifrar con un AWS KMS anillo de claves, necesita el permiso KMS:Decrypt en al menos una clave del anillo de claves. AWS KMS
- Para cifrar con un conjunto de claves múltiples compuesto por AWS KMS anillos de claves, necesita el permiso kms: en la clave generadora del conjunto de claves del generador. GenerateDataKey Necesita el permiso KMS:Encrypt para el resto de claves de todos los demás conjuntos de claves. AWS KMS
- Para cifrar con un AWS KMS anillo de claves RSA asimétrico, no necesita kms: GenerateDataKey ni KMS:Encrypt porque debe especificar el material de clave pública que desea utilizar para el cifrado al crear el anillo de claves. No se realizan llamadas AWS KMS al cifrar con este anillo de

claves. Para descifrar con un AWS KMS anillo de claves RSA asimétrico, necesita el permiso KMS:Decrypt.

Para obtener información detallada sobre los permisos AWS KMS keys, consulte el acceso y los permisos a las claves de KMS en la Guía para desarrolladores.AWS Key Management Service

Identificarse AWS KMS keys en un AWS KMS llavero

Un AWS KMS llavero puede incluir uno o más. AWS KMS keys Para especificar un elemento AWS KMS key en un conjunto de AWS KMS claves, utilice un identificador de AWS KMS clave compatible. Los identificadores clave que puede utilizar para identificar un elemento de un AWS KMS key conjunto de claves varían según la operación y la implementación del idioma. Para obtener más información sobre los identificadores clave de una AWS KMS key, consulte Identificadores clave en la Guía para desarrolladores de AWS Key Management Service .

Como práctica recomendada, utilice el identificador de clave más práctico que sea práctico para su tarea.

- En un anillo de claves de cifrado para el SDK de cifrado de AWS para C, puede utilizar un ARN de clave o un alias ARN para identificar las claves de KMS. En todas las implementaciones de otros lenguajes, puede utilizar un identificador de clave, un ARN de clave, un nombre de alias o un ARN de alias para cifrar los datos.
- En un conjunto de claves de descifrado, debe usar el ARN de una clave para identificar AWS KMS keys. Este requisito se aplica a todas las implementaciones de lenguaje del AWS Encryption SDK. Para obtener más información, consulte Seleccionar las claves de encapsulamiento.
- En un conjunto de claves usado para cifrar y descifrar, debe usar el ARN de una clave para identificar AWS KMS keys. Este requisito se aplica a todas las implementaciones de lenguaje del AWS Encryption SDK.

Si especifica un nombre de alias o un ARN de alias para una clave de KMS en un conjunto de claves de cifrado, la operación de cifrado guarda el ARN de clave actualmente asociado al alias en los metadatos de la clave de datos cifrada. No guarda el alias. Los cambios en el alias no afectan a la clave de KMS utilizada para descifrar las claves de datos cifrados.

Crear un anillo de claves AWS KMS

Puede configurar cada AWS KMS llavero con uno AWS KMS key o varios AWS KMS keys en la misma o en una diferente Cuentas de AWS . Regiones de AWS AWS KMS keys Debe ser una clave KMS de cifrado simétrico (SYMMETRIC_DEFAULT) o una clave KMS RSA asimétrica. También puede utilizar una clave KMS de múltiples regiones de cifrado simétrico. Puede utilizar uno o varios conjunto de claves de AWS KMS en un conjunto de claves múltiple.

Puede crear un conjunto de AWS KMS claves que cifre y descifre los datos, o puede crear conjuntos de claves específicos para cifrar o descifrar. AWS KMS Al crear un conjunto de AWS KMS claves para cifrar datos, debe especificar una clave generadora, que es aquella que se utiliza para generar una clave de datos en AWS KMS key texto plano y cifrarla. La clave de datos no está relacionada matemáticamente con la clave de KMS. A continuación, si lo desea, puede especificar otras claves adicionales AWS KMS keys que cifren la misma clave de datos en texto plano. Para descifrar un campo cifrado protegido por este anillo de claves, el anillo de claves de descifrado que utilice debe incluir al menos una de las AWS KMS keys definidas en el anillo de claves, o no. AWS KMS keys(Un anillo de AWS KMS claves sin un número se conoce como anillo de claves de AWS KMS keys detección).AWS KMS

En las implementaciones AWS Encryption SDK lingüísticas distintas del SDK de cifrado de AWS para C, todas las claves de empaquetado de un anillo de claves de cifrado o de varios anillos de claves deben poder cifrar la clave de datos. Si alguna de las claves de encapsulamiento no se cifra, el método de cifrado falla. Como resultado, la persona que llama debe tener los permisos necesarios para todas las claves del conjunto de claves. Si utiliza un conjunto de claves de detección para cifrar los datos, solo o en un conjunto de claves múltiple, la operación de cifrado no se realizará correctamente. La excepción es la siguiente SDK de cifrado de AWS para C, en la que la operación de cifrado ignora un anillo de claves de detección estándar, pero falla si se especifica un anillo de claves de descubrimiento multirregional, solo o en un anillo de claves múltiples.

En los ejemplos siguientes se crea un AWS KMS anillo de claves con una clave generadora y una clave adicional. Tanto la clave generadora como la clave adicional son claves KMS de cifrado simétrico. En estos ejemplos, se utiliza la clave ARNs para identificar las claves de KMS. Esta es una práctica recomendada para AWS KMS los anillos de claves utilizados para el cifrado y un requisito para los AWS KMS anillos de claves utilizados para el descifrado. Para obtener más información, consulte Identificarse AWS KMS keys en un AWS KMS llavero.

C

Para identificar un AWS KMS key en un anillo de claves de cifrado en SDK de cifrado de AWS para C, especifique un ARN de clave o un ARN de alias. En un conjunto de claves de descifrado, debe usar un ARN de clave. Para obtener más información, consulte Identificarse AWS KMS keys en un AWS KMS llavero.

Para obtener un ejemplo completo, consulte 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

Para crear un conjunto de claves con una o más claves de KMS en el dominio.NET, AWS Encryption SDK utilice el método. `CreateAwsKmsMultiKeyring()` En este ejemplo se utilizan dos claves de AWS KMS . Para especificar una clave KMS, utilice solo el parámetro `Generator`. El parámetro `KmsKeyIds` que especifica claves KMS adicionales es opcional.

La entrada de este conjunto de claves no requiere un AWS KMS cliente. En su lugar, AWS Encryption SDK utiliza el AWS KMS cliente predeterminado para cada región, representado por una clave KMS en el anillo de claves. Por ejemplo, si la clave KMS identificada por el valor del `Generator` parámetro se encuentra en la región EE.UU. Oeste (Oregón) (`us-west-2`), se AWS Encryption SDK crea un AWS KMS cliente predeterminado para la `us-west-2` región. Si necesita personalizar el cliente de AWS KMS , utilice el método `CreateAwsKmsKeyring()`.

Al especificar un AWS KMS key para un conjunto de claves de cifrado en para.NET, puede utilizar cualquier identificador de clave válido: un ID de clave, un ARN de clave, un nombre de alias o un ARN de alias. AWS Encryption SDK Si necesita ayuda para identificarlo AWS KMS keys en un conjunto de AWS KMS claves, consulte. Identificarse AWS KMS keys en un AWS KMS llavero

En el siguiente ejemplo se utiliza la versión 4. x AWS Encryption SDK para .NET y el `CreateAwsKmsKeyring()` método para personalizar el AWS KMS cliente.

```
// 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 = mpl.CreateAwsKmsMultiKeyring(createEncryptKeyringInput);
```

JavaScript Browser

Al especificar un AWS KMS key para un conjunto de claves de cifrado en SDK de cifrado de AWS para JavaScript, puede utilizar cualquier identificador de clave válido: un ID de clave, un ARN de clave, un nombre de alias o un ARN de alias. Si necesita ayuda para identificarlo AWS KMS keys en un conjunto de AWS KMS claves, consulte. Identificarse AWS KMS keys en un AWS KMS llavero

En el siguiente ejemplo, se utiliza la `buildClient` función para especificar la política de compromiso predeterminada,`REQUIRE_ENCRYPT_REQUIRE_DECRYPT`. También puede utilizarla `buildClient` para limitar el número de claves de datos cifradas en un mensaje cifrado. Para obtener más información, consulte the section called “Limitar las claves de datos cifrados”.

Para ver un ejemplo completo, consulte kms_simple.ts en el repositorio de. SDK de cifrado de AWS para JavaScript GitHub

```
import {
  KmsKeyringNode,
  buildClient,
  CommitmentPolicy,
} from '@aws-crypto/client-node'

const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)
```

```
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

Al especificar un AWS KMS key para un conjunto de claves de cifrado en SDK de cifrado de AWS para JavaScript, puede utilizar cualquier identificador de clave válido: un ID de clave, un ARN de clave, un nombre de alias o un ARN de alias. Si necesita ayuda para identificarlo AWS KMS keys en un conjunto de AWS KMS claves, consulte. Identificarse AWS KMS keys en un AWS KMS llavero

En el siguiente ejemplo, se utiliza la `buildClient` función para especificar la política de compromiso predeterminada,`REQUIRE_ENCRYPT_REQUIRE_DECRYPT`. También puede utilizarla `buildClient` para limitar el número de claves de datos cifradas en un mensaje cifrado. Para obtener más información, consulte the section called “Limitar las claves de datos cifrados”.

Para ver un ejemplo completo, consulte kms_simple.ts en el repositorio de. SDK de cifrado de AWS para JavaScript GitHub

```
import {
  KmsKeyringNode,
  buildClient,
  CommitmentPolicy,
} from '@aws-crypto/client-node'

const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

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

Para crear un conjunto de claves con una o más claves, utilice el método. AWS KMS `CreateAwsKmsMultiKeyring()` En este ejemplo, se utilizan dos claves KMS. Para especificar una clave KMS, utilice solo el parámetro `generator`. El parámetro `kmsKeyIds` que especifica claves KMS adicionales es opcional.

La entrada de este conjunto de claves no acepta un AWS KMS cliente. En su lugar, AWS Encryption SDK utiliza el AWS KMS cliente predeterminado para cada región, representado por una clave KMS en el anillo de claves. Por ejemplo, si la clave KMS identificada por el valor del `Generator` parámetro se encuentra en la región EE.UU. Oeste (Oregón) (`us-west-2`), se AWS Encryption SDK crea un AWS KMS cliente predeterminado para la `us-west-2` región. Si necesita personalizar el cliente de AWS KMS , utilice el método `CreateAwsKmsKeyring()`.

Al especificar un AWS KMS key para un conjunto de claves de cifrado en SDK de cifrado de AWS para Java, puede utilizar cualquier identificador de clave válido: un ID de clave, un ARN de clave, un nombre de alias o un ARN de alias. Si necesita ayuda para identificarlo AWS KMS keys en un conjunto de AWS KMS claves, consulte. Identificarse AWS KMS keys en un AWS KMS llavero

Para ver un ejemplo completo, consulte BasicEncryptionKeyringExample.java en el SDK de cifrado de AWS para Java repositorio de. 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("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);
```

Python

Para crear un conjunto de claves con una o más AWS KMS claves, utilice el `create_aws_kms_multi_keyring()` método. En este ejemplo, se utilizan dos claves KMS. Para especificar una clave KMS, utilice solo el parámetro `generator`. El parámetro `kms_key_ids` que especifica claves KMS adicionales es opcional.

La entrada de este conjunto de claves no acepta un AWS KMS cliente. En su lugar, AWS Encryption SDK utiliza el AWS KMS cliente predeterminado para cada región, representado por una clave KMS en el anillo de claves. Por ejemplo, si la clave KMS identificada por el valor del `generator` parámetro se encuentra en la región EE.UU. Oeste (Oregón) (`us-west-2`), se AWS Encryption SDK crea un AWS KMS cliente predeterminado para la `us-west-2` región. Si necesita personalizar el cliente de AWS KMS , utilice el método `create_aws_kms_keyring()`.

Al especificar un AWS KMS key para un conjunto de claves de cifrado en SDK de cifrado de AWS para Python, puede utilizar cualquier identificador de clave válido: un ID de clave, un ARN de clave, un nombre de alias o un ARN de alias. Si necesita ayuda para identificarlo AWS KMS keys en un conjunto de AWS KMS claves, consulte. Identificarse AWS KMS keys en un AWS KMS llavero

En el siguiente ejemplo, se crea una instancia del AWS Encryption SDK cliente con la política de compromiso predeterminada,. `REQUIRE_ENCRYPT_REQUIRE_DECRYPT` Para ver un ejemplo completo, consulte aws_kms_multi_keyring_example.py en el SDK de cifrado de AWS para Python repositorio de. GitHub

```
# Instantiate the AWS Encryption SDK client
client = aws_encryption_sdk.EncryptionSDKClient(
    commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

# Optional: Create an encryption context
encryption_context: Dict[str, str] = {
    "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 material providers library
mat_prov: AwsCryptographicMaterialProviders = AwsCryptographicMaterialProviders(
    config=MaterialProvidersConfig()
)

# Create the AWS KMS keyring
kms_multi_keyring_input: CreateAwsKmsMultiKeyringInput =
 CreateAwsKmsMultiKeyringInput(
    generator="arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    kms_key_ids="arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321"
)

kms_multi_keyring: IKeyring = mat_prov.create_aws_kms_multi_keyring(
    input=kms_multi_keyring_input
)
```

Rust

Para crear un conjunto de claves con una o más AWS KMS claves, utilice el `create_aws_kms_multi_keyring()` método. En este ejemplo, se utilizan dos claves KMS. Para especificar una clave KMS, utilice solo el parámetro `generator`. El parámetro `kms_key_ids` que especifica claves KMS adicionales es opcional.

La entrada de este conjunto de claves no acepta un AWS KMS cliente. En su lugar, AWS Encryption SDK utiliza el AWS KMS cliente predeterminado para cada región, representado por una clave KMS en el anillo de claves. Por ejemplo, si la clave KMS identificada por el valor del `generator` parámetro se encuentra en la región EE.UU. Oeste (Oregón) (`us-west-2`), se AWS Encryption SDK crea un AWS KMS cliente predeterminado para la `us-west-2` región. Si necesita personalizar el cliente de AWS KMS , utilice el método `create_aws_kms_keyring()`.

Cuando especificas un AWS KMS key para un anillo de claves de cifrado en Rust, puedes usar cualquier identificador de clave válido: un ID de clave, un ARN de clave, un nombre de alias o un ARN de alias. AWS Encryption SDK Para obtener ayuda para identificar lo que hay AWS KMS keys en un conjunto de AWS KMS claves, consulte. Identificarse AWS KMS keys en un AWS KMS llavero

En el siguiente ejemplo, se crea una instancia del AWS Encryption SDK cliente con la política de compromiso predeterminada,. `REQUIRE_ENCRYPT_REQUIRE_DECRYPT` Para ver un ejemplo completo, consulta aws_kms_keyring_example.rs en el directorio Rust del repositorio de. aws-encryption-sdk GitHub

```
// Instantiate the AWS Encryption SDK client
let esdk_config = AwsEncryptionSdkConfig::builder().build()?;
let esdk_client = esdk_client::Client::from_conf(esdk_config)?;

// Create an AWS KMS client
let sdk_config =
 aws_config::load_defaults(aws_config::BehaviorVersion::latest()).await;
let kms_client = aws_sdk_kms::Client::new(&sdk_config);

// Optional: Create an encryption context
let encryption_context = HashMap::from([
    ("encryption".to_string(), "context".to_string()),
    ("is not".to_string(), "secret".to_string()),
    ("but adds".to_string(), "useful metadata".to_string()),
    ("that can help you".to_string(), "be confident that".to_string()),
    ("the data you are handling".to_string(), "is what you think it
 is".to_string()),
]);

// Instantiate the material providers library
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;

// Create the AWS KMS keyring
let kms_keyring = mpl
    .create_aws_kms_keyring()
    .kms_key_id(kms_key_id)
    .kms_client(kms_client)
    .send()
    .await?;

kms_multi_keyring: IKeyring = mpl.create_aws_kms_multi_keyring(
    input=kms_multi_keyring_input
)
```

Go

Para crear un conjunto de claves con una o más claves, usa el método. AWS KMS `create_aws_kms_multi_keyring()` En este ejemplo, se utilizan dos claves KMS. Para especificar una clave KMS, utilice solo el parámetro `generator`. El parámetro `kms_key_ids` que especifica claves KMS adicionales es opcional.

La entrada de este conjunto de claves no acepta un AWS KMS cliente. En su lugar, AWS Encryption SDK utiliza el AWS KMS cliente predeterminado para cada región, representado por una clave KMS en el anillo de claves. Por ejemplo, si la clave KMS identificada por el valor del `generator` parámetro se encuentra en la región EE.UU. Oeste (Oregón) (`us-west-2`), se AWS Encryption SDK crea un AWS KMS cliente predeterminado para la `us-west-2` región. Si necesita personalizar el cliente de AWS KMS , utilice el método `create_aws_kms_keyring()`.

Al especificar un AWS KMS key para un conjunto de claves de cifrado en AWS Encryption SDK for Go, puede utilizar cualquier identificador de clave válido: un ID de clave, un ARN de clave, un nombre de alias o un ARN de alias. Si necesita ayuda para identificarlo AWS KMS keys en un conjunto de AWS KMS claves, consulte. Identificarse AWS KMS keys en un AWS KMS llavero

En el siguiente ejemplo, se crea una instancia del AWS Encryption SDK cliente con la política de compromiso predeterminada,. `REQUIRE_ENCRYPT_REQUIRE_DECRYPT`

```
import (
    "context"

 mpl "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygenerated"
 mpltypes "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygeneratedtypes"
 client "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygenerated"
 esdktypes "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygeneratedtypes"
)

// Instantiate the AWS Encryption SDK client
encryptionClient, err := client.NewClient(esdktypes.AwsEncryptionSdkConfig{})
if err != nil {
    panic(err)
}
```

```
// Optional: Create an encryption context
encryptionContext := map[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 material providers library
matProv, err := mpl.NewClient(mpltypes.MaterialProvidersConfig{})
if err != nil {
    panic(err)
}

// Create the AWS KMS keyring
awsKmsMultiKeyringInput := mpltypes.CreateAwsKmsMultiKeyringInput{
    Generator: "&arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    KmsKeyIds: []string{"arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321"},
}
awsKmsMultiKeyring, err := matProv.CreateAwsKmsMultiKeyring(context.Background(),
 awsKmsMultiKeyringInput)
```

AWS Encryption SDK También admite conjuntos de claves que utilizan AWS KMS claves RSA KMS asimétricas. Los AWS KMS llaveros RSA asimétricos solo pueden contener un par de claves.

Para cifrar con un AWS KMS anillo de claves RSA asimétrico, no necesita kms: GenerateDataKey ni KMS:Encrypt, ya que debe especificar el material de clave pública que desea utilizar para el cifrado al crear el anillo de claves. No se realizan llamadas a AWS KMS al cifrar con este conjunto de claves. Para descifrar con un anillo de claves RSA asimétrico, necesita el permiso KMS:Decrypt. AWS KMS

> **Note**
>
> Para crear un conjunto de claves que utilice AWS KMS claves KMS RSA asimétricas, debe utilizar una de las siguientes implementaciones de lenguajes de programación:
>
> - Versión 3. x del SDK de cifrado de AWS para Java
> - Versión 4. x del AWS Encryption SDK para .NET

- Versión 4. x de SDK de cifrado de AWS para Python, cuando se usa con la dependencia opcional de la biblioteca de proveedores de material criptográfico (MPL).
- Versión 1. x de la AWS Encryption SDK para Rust
- Versión 0.1. x o posterior del AWS Encryption SDK for Go

En los siguientes ejemplos, se utiliza el `CreateAwsKmsRsaKeyring` método para crear un AWS KMS anillo de claves con una clave RSA KMS asimétrica. Para crear un AWS KMS anillo de claves RSA asimétrico, proporcione los siguientes valores.

- `kmsClient`: crea un cliente nuevo AWS KMS
- `kmsKeyID`: el ARN clave que identifica su clave RSA KMS asimétrica
- `publicKey`: a ByteBuffer de un archivo PEM codificado en UTF-8 que representa la clave pública de la clave a la que se le pasó `kmsKeyID`
- `encryptionAlgorithm`: el algoritmo de cifrado debe ser o `RSAES_OAEP_SHA_256` `RSAES_OAEP_SHA_1`

C# / .NET

Para crear un conjunto de AWS KMS claves RSA asimétrico, debe proporcionar la clave pública y el ARN de clave privada de su clave RSA KMS asimétrica. La clave pública debe estar codificada con PEM. En el siguiente ejemplo, se crea un AWS KMS anillo de claves con un par de claves RSA asimétricas.

```
// 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);
```

Java

Para crear un conjunto de AWS KMS claves RSA asimétrico, debe proporcionar la clave pública y el ARN de clave privada de su clave RSA KMS asimétrica. La clave pública debe estar codificada con PEM. En el siguiente ejemplo, se crea un AWS KMS anillo de claves con un par de claves RSA asimétricas.

```
// 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);
```

Python

Para crear un conjunto de AWS KMS claves RSA asimétrico, debe proporcionar la clave pública y el ARN de clave privada de su clave RSA KMS asimétrica. La clave pública debe estar codificada con PEM. En el siguiente ejemplo, se crea un AWS KMS anillo de claves con un par de claves RSA asimétricas.

```
# Instantiate the AWS Encryption SDK client
client = aws_encryption_sdk.EncryptionSDKClient(
    commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

# Optional: Create an encryption context
encryption_context: Dict[str, str] = {
    "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 material providers library
mat_prov: AwsCryptographicMaterialProviders = AwsCryptographicMaterialProviders(
    config=MaterialProvidersConfig()
)

# Create the AWS KMS keyring
keyring_input: CreateAwsKmsRsaKeyringInput = CreateAwsKmsRsaKeyringInput(
    public_key="public_key",
    kms_key_id="kms_key_id",
    encryption_algorithm="RSAES_OAEP_SHA_256",
    kms_client=kms_client
)

kms_rsa_keyring: IKeyring = mat_prov.create_aws_kms_rsa_keyring(
    input=keyring_input
)
```

Rust

Para crear un conjunto de AWS KMS claves RSA asimétrico, debe proporcionar la clave pública y el ARN de clave privada de su clave RSA KMS asimétrica. La clave pública debe estar codificada con PEM. En el siguiente ejemplo, se crea un AWS KMS anillo de claves con un par de claves RSA asimétricas.

```
// Instantiate the AWS Encryption SDK client
let esdk_config = AwsEncryptionSdkConfig::builder().build()?;
let esdk_client = esdk_client::Client::from_conf(esdk_config)?;

// Create an AWS KMS client
let sdk_config =
 aws_config::load_defaults(aws_config::BehaviorVersion::latest()).await;
let kms_client = aws_sdk_kms::Client::new(&sdk_config);

// Optional: Create an encryption context
let encryption_context = HashMap::from([
    ("encryption".to_string(), "context".to_string()),
    ("is not".to_string(), "secret".to_string()),
    ("but adds".to_string(), "useful metadata".to_string()),
    ("that can help you".to_string(), "be confident that".to_string()),
    ("the data you are handling".to_string(), "is what you think it
 is".to_string()),
]);

// Instantiate the material providers library
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;

// Create the AWS KMS keyring
let kms_rsa_keyring = mpl
    .create_aws_kms_rsa_keyring()
    .kms_key_id(kms_key_id)
    .public_key(aws_smithy_types::Blob::new(public_key))

 .encryption_algorithm(aws_sdk_kms::types::EncryptionAlgorithmSpec::RsaesOaepSha256)
    .kms_client(kms_client)
    .send()
    .await?;
```

Go

```
import (
    "context"

 mpl "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygenerated"
 mpltypes "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygeneratedtypes"
 client "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygenerated"
 esdktypes "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygeneratedtypes"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/kms"
)

// Instantiate the AWS Encryption SDK client
encryptionClient, err := client.NewClient(esdktypes.AwsEncryptionSdkConfig{})
if err != nil {
    panic(err)
}

// Create an AWS KMS client
cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    panic(err)
}
kmsClient := kms.NewFromConfig(cfg, func(o *kms.Options) {
    o.Region = KmsKeyRegion
})

// Optional: Create an encryption context
encryptionContext := map[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 material providers library
matProv, err := mpl.NewClient(mpltypes.MaterialProvidersConfig{})
if err != nil {
```

```
    panic(err)
}

// Create the AWS KMS keyring
awsKmsRSAKeyringInput := mpltypes.CreateAwsKmsRsaKeyringInput{
    KmsClient:           kmsClient,
    KmsKeyId:            kmsKeyID,
    PublicKey:           kmsPublicKey,
    EncryptionAlgorithm: kmstypes.EncryptionAlgorithmSpecRsaesOaepSha256,
}
awsKmsRSAKeyring, err := matProv.CreateAwsKmsRsaKeyring(context.Background(),
 awsKmsRSAKeyringInput)
if err != nil {
    panic(err)
}
```

Uso de un anillo de claves de detección AWS KMS

Al descifrar, se recomienda especificar las claves de empaquetado que AWS Encryption SDK pueden utilizar. Para seguir esta práctica recomendada, utilice un conjunto de claves de AWS KMS descifrado que limite las claves de AWS KMS empaquetado a las que especifique. Sin embargo, también puede crear un anillo de claves de AWS KMS detección, es decir, un anillo de AWS KMS claves que no especifique ninguna clave de empaquetado.

AWS Encryption SDK Incluye un conjunto de claves de AWS KMS detección estándar y un anillo de claves de detección para claves de varias regiones. AWS KMS Para obtener información sobre cómo utilizar claves de varias regiones con el AWS Encryption SDK, consulte Uso de varias regiones AWS KMS keys.

Como no especifica ninguna clave de encapsulamiento, un conjunto de claves de detección no puede cifrar los datos. Si utiliza un conjunto de claves de detección para cifrar los datos, solo o en un conjunto de claves múltiple, la operación de cifrado no se realizará correctamente. La excepción es la siguiente SDK de cifrado de AWS para C, en la que la operación de cifrado ignora un conjunto de claves de detección estándar, pero no se produce si se especifica un conjunto de claves de detección multirregional, solo o en un anillo de claves múltiples.

Al descifrar, un anillo de claves de detección permite solicitar AWS KMS el AWS Encryption SDK descifrado de cualquier clave de datos cifrada utilizando la clave AWS KMS key que la cifró, independientemente de quién sea su propietario o tenga acceso a ella. AWS KMS key La llamada

se realiza correctamente solo si el intermediario tiene permiso de `kms:Decrypt` sobre la AWS KMS key.

> **Important**
>
> Si incluye un anillo de claves de AWS KMS descubrimiento en un conjunto de claves de descifrado múltiple, el anillo de claves de descubrimiento anula todas las restricciones de claves de KMS especificadas por otros conjuntos de claves del conjunto de claves múltiples. El conjunto de claves múltiples se comporta como el menos restrictivo. Cuando se utiliza un conjunto de claves de detección de AWS KMS en un conjunto de claves múltiple, no tiene ningún efecto sobre el cifrado.

Incluye AWS KMS un llavero de detección para mayor comodidad. AWS Encryption SDK No obstante, recomendamos que utilice un conjunto de claves más limitado siempre que sea posible por los siguientes motivos.

- Autenticidad: un anillo de claves de AWS KMS detección puede utilizar cualquier clave AWS KMS key que se haya utilizado para cifrar una clave de datos del mensaje cifrado, con el fin de que la persona que llama tenga permiso para usarla para descifrarla. AWS KMS key Es posible que esta no sea la AWS KMS key que la persona que llama pretende usar. Por ejemplo, es posible que una de las claves de datos cifradas se haya cifrado con un sistema menos seguro AWS KMS key que cualquiera pueda utilizar.
- Latencia y rendimiento: un AWS KMS conjunto de claves de detección puede ser considerablemente más lento que otros, ya que AWS Encryption SDK intenta descifrar todas las claves de datos cifradas, incluidas las cifradas en otras regiones y AWS KMS keys en otras regiones, Cuentas de AWS y la persona AWS KMS keys que llama no tiene permiso para utilizarlas para descifrarlas.

Si utiliza un conjunto de claves de detección, le recomendamos que utilice un filtro de detección para limitar las claves de KMS que se pueden utilizar a las que se encuentran en particiones Y especificadas. Cuentas de AWS Los filtros de detección son compatibles con las versiones 1.7.x y versiones posteriores del AWS Encryption SDK. Para obtener ayuda para encontrar el ID y la partición de su cuenta, consulte Sus Cuenta de AWS identificadores y el formato ARN en. Referencia general de AWS

El siguiente código crea una instancia de un conjunto de claves de AWS KMS detección con un filtro de detección que limita las claves de KMS que se AWS Encryption SDK pueden utilizar a las de la `aws` partición y de la cuenta de ejemplo 111122223333.

Antes de usar este código, sustituya los valores del ejemplo Cuenta de AWS y de la partición por valores válidos para la partición y. Cuenta de AWS Si sus claves de KMS se encuentran en regiones de China, use el valor de la partición de `aws-cn`. Si las claves KMS están en AWS GovCloud (US) Regions, use el valor de la partición de `aws-us-gov`. Para todas las demás Regiones de AWS, utilice el valor de la partición de `aws`.

C

Para ver un ejemplo completo, consulte: 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

En el ejemplo siguiente se utiliza la versión 4.x del AWS Encryption SDK para .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 =
 mpl.CreateAwsKmsDiscoveryKeyring(kmsDiscoveryKeyringInput);
```

JavaScript Browser

En JavaScript, debe especificar de forma explícita la propiedad de descubrimiento.

En el siguiente ejemplo, se utiliza la `buildClient` función para especificar la política de compromiso predeterminada,`REQUIRE_ENCRYPT_REQUIRE_DECRYPT`. También puede utilizarla `buildClient` para limitar el número de claves de datos cifradas en un mensaje cifrado. Para obtener más información, consulte the section called “Limitar las claves de datos cifrados”.

```
import {
  KmsKeyringBrowser,
  buildClient,
  CommitmentPolicy,
} from '@aws-crypto/client-browser'

const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

const clientProvider = getClient(KMS, { credentials })

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

JavaScript Node.js

En JavaScript, debe especificar de forma explícita la propiedad de detección.

En el siguiente ejemplo, se utiliza la `buildClient` función para especificar la política de compromiso predeterminada,`REQUIRE_ENCRYPT_REQUIRE_DECRYPT`. También puede utilizarla `buildClient` para limitar el número de claves de datos cifradas en un mensaje cifrado. Para obtener más información, consulte the section called “Limitar las claves de datos cifrados”.

```
import {
```

```
  KmsKeyringNode,
  buildClient,
  CommitmentPolicy,
} from '@aws-crypto/client-node'

const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

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);
```

Python

```
# Instantiate the AWS Encryption SDK
client = aws_encryption_sdk.EncryptionSDKClient(
    commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

# Create a boto3 client for AWS KMS
kms_client = boto3.client('kms', region_name=aws_region)

# Optional: Create an encryption context
encryption_context: Dict[str, str] = {
```

```
    "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 material providers
mat_prov: AwsCryptographicMaterialProviders = AwsCryptographicMaterialProviders(
    config=MaterialProvidersConfig()
)

# Create the AWS KMS discovery keyring
discovery_keyring_input: CreateAwsKmsDiscoveryKeyringInput =
 CreateAwsKmsDiscoveryKeyringInput(
    kms_client=kms_client,
    discovery_filter=DiscoveryFilter(
        account_ids=[aws_account_id],
        partition="aws"
    )
)

discovery_keyring: IKeyring = mat_prov.create_aws_kms_discovery_keyring(
    input=discovery_keyring_input
)
```

Rust

```
// Instantiate the AWS Encryption SDK
let esdk_config = AwsEncryptionSdkConfig::builder().build()?;
let esdk_client = esdk_client::Client::from_conf(esdk_config)?;

// Create a AWS KMS client.
let sdk_config =
 aws_config::load_defaults(aws_config::BehaviorVersion::latest()).await;
let kms_client = aws_sdk_kms::Client::new(&sdk_config);


// Instantiate the material providers library
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;
```

```
// Create discovery filter
let discovery_filter = DiscoveryFilter::builder()
    .account_ids(vec![aws_account_id.to_string()])
    .partition("aws".to_string())
    .build()?;

// Create the AWS KMS discovery keyring
let discovery_keyring = mpl
    .create_aws_kms_discovery_keyring()
    .kms_client(kms_client.clone())
    .discovery_filter(discovery_filter)
    .send()
    .await?;
```

Go

```
import (
    "context"

 mpl "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygenerated"
 mpltypes "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygeneratedtypes"
 client "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygenerated"
 esdktypes "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygeneratedtypes"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/kms"
)

// Instantiate the AWS Encryption SDK client
encryptionClient, err := client.NewClient(esdktypes.AwsEncryptionSdkConfig{})
if err != nil {
    panic(err)
}

// Create an AWS KMS client
cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    panic(err)
}
kmsClient := kms.NewFromConfig(cfg, func(o *kms.Options) {
```

```
    o.Region = KmsKeyRegion
})

// Optional: Create an encryption context
encryptionContext := map[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 material providers library
matProv, err := mpl.NewClient(mpltypes.MaterialProvidersConfig{})
if err != nil {
    panic(err)
}

// Create discovery filter
discoveryFilter := mpltypes.DiscoveryFilter{
    AccountIds: []string{kmsKeyAccountID},
    Partition:  "aws",
}
awsKmsDiscoveryKeyringInput := mpltypes.CreateAwsKmsDiscoveryKeyringInput{
    KmsClient:       kmsClient,
    DiscoveryFilter: &discoveryFilter,
}
awsKmsDiscoveryKeyring, err :=
 matProv.CreateAwsKmsDiscoveryKeyring(context.Background(),
 awsKmsDiscoveryKeyringInput)
if err != nil {
    panic(err)
}
```

Uso de un anillo de claves de detección AWS KMS regional

Un conjunto de claves de detección AWS KMS regional es un conjunto de claves que no especifica las claves ARNs de KMS. En su lugar, AWS Encryption SDK permite descifrar utilizando únicamente las claves KMS en particular. Regiones de AWS

Al descifrar con un conjunto de claves de detección AWS KMS regional, AWS Encryption SDK descifra cualquier clave de datos cifrada que se haya cifrado con una AWS KMS key de las

especificadas. Región de AWS Para tener éxito, la persona que llama debe tener `kms:Decrypt` permiso en al menos una de las especificadas Región de AWS que AWS KMS keys cifraron una clave de datos.

Al igual que otros conjuntos de claves de detección, el conjunto de claves de detección regional no tiene ningún efecto sobre el cifrado. Solo funciona cuando se descifran mensajes cifrados. Si utiliza un conjunto de claves de detección regional en un conjunto de claves múltiples que se utiliza para cifrar y descifrar, solo es efectivo al descifrar. Si utiliza un conjunto de claves de detección multirregional para cifrar los datos, solo o en un conjunto de claves múltiples, la operación de cifrado no se realizará correctamente.

> **Important**
>
> Si incluye un conjunto de claves de detección AWS KMS regional en un conjunto de claves de descifrado múltiple, el conjunto de claves de descubrimiento regional anula todas las restricciones de claves de KMS especificadas en otros anillos de claves del conjunto de claves múltiples. El conjunto de claves múltiples se comporta como el menos restrictivo. Cuando se utiliza un conjunto de claves de detección de AWS KMS en un conjunto de claves múltiple, no tiene ningún efecto sobre el cifrado.

El conjunto de claves de detección regional SDK de cifrado de AWS para C intenta descifrar únicamente las claves de KMS de la región especificada. Cuando se utiliza un conjunto de claves de detección en SDK de cifrado de AWS para JavaScript y AWS Encryption SDK para .NET, se configura la región en el cliente. AWS KMS Estas AWS Encryption SDK implementaciones no filtran las claves de KMS por región, pero no AWS KMS permiten descifrar las claves de KMS de fuera de la región especificada.

Si utiliza un conjunto de claves de detección, le recomendamos que utilice un filtro de detección para limitar las claves de KMS utilizadas en el descifrado a las de las particiones Y especificadas. Cuentas de AWS Los filtros de detección son compatibles con las versiones 1.7.x y versiones posteriores del AWS Encryption SDK.

Por ejemplo, el código siguiente crea un conjunto de claves de detección AWS KMS regional con un filtro de detección. Este conjunto de claves limita las AWS Encryption SDK claves KMS de la cuenta 111122223333 en la región EE.UU. Oeste (Oregón) (us-west-2).

C

Para ver este conjunto de claves y el método `create_kms_client`, en un ejemplo práctico, consulte 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

La versión AWS Encryption SDK para .NET no tiene un conjunto de claves de descubrimiento regional específico. Sin embargo, puede utilizar varias técnicas para limitar las claves de KMS que se utilizan al descifrar a una región determinada.

La forma más eficaz de limitar las regiones de un conjunto de claves de descubrimiento es utilizar un conjunto de claves de multi-Region-aware descubrimiento, incluso si se han cifrado los datos con claves de una sola región. Cuando encuentra claves de una sola región, el multi-Region-aware llavero no utiliza ninguna función multirregional.

El conjunto de claves devuelto por el `CreateAwsKmsMrkDiscoveryKeyring()` método filtra las claves de KMS por región antes de llamar a AWS KMS. Envía una solicitud de descifrado AWS KMS solo cuando la clave de datos cifrada se cifró con una clave KMS de la región especificada por el `Region` parámetro del objeto. `CreateAwsKmsMrkDiscoveryKeyringInput`

En los ejemplos siguientes se utiliza la versión 4.x del AWS Encryption SDK para .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 =
 mpl.CreateAwsKmsMrkDiscoveryKeyring(regionalDiscoveryKeyringInput);
```

También puede limitar las claves de KMS a una determinada región Región de AWS especificando una región en su instancia del AWS KMS cliente (AmazonKeyManagementServiceClient). Sin embargo, esta configuración es menos eficiente y potencialmente más costosa que el uso de un conjunto de claves de multi-Region-aware detección. En lugar de filtrar las claves de KMS por región antes de realizar la llamada AWS KMS, el AWS Encryption SDK dominio.NET AWS KMS solicita cada clave de datos cifrada (hasta que descifra una) y se basa en AWS KMS limitar las claves de KMS que utiliza a la región especificada.

En el ejemplo siguiente se utiliza la versión 4.x del AWS Encryption SDK para .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 =
 mlp.CreateAwsKmsDiscoveryKeyring(createRegionalDiscoveryKeyringInput);
```

JavaScript Browser

En el siguiente ejemplo, se utiliza la `buildClient` función para especificar la política de compromiso predeterminada,`REQUIRE_ENCRYPT_REQUIRE_DECRYPT`. También puede utilizarla `buildClient` para limitar el número de claves de datos cifradas en un mensaje cifrado. Para obtener más información, consulte the section called “Limitar las claves de datos cifrados”.

```
import {
  KmsKeyringNode,
  buildClient,
  CommitmentPolicy,
} from '@aws-crypto/client-node'

const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

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

En el siguiente ejemplo, se utiliza la `buildClient` función para especificar la política de compromiso predeterminada,`REQUIRE_ENCRYPT_REQUIRE_DECRYPT`. También puede utilizarla `buildClient` para limitar el número de claves de datos cifradas en un mensaje cifrado. Para obtener más información, consulte the section called “Limitar las claves de datos cifrados”.

Para ver este conjunto de claves y la `limitRegions` función en un ejemplo práctico, consulte kms_regional_discovery.ts.

```
import {
```

```
  KmsKeyringNode,
  buildClient,
  CommitmentPolicy,
} from '@aws-crypto/client-node'

const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

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);
```

Python

```
# Instantiate the AWS Encryption SDK
client = aws_encryption_sdk.EncryptionSDKClient(
    commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

# Create a boto3 client for AWS KMS
kms_client = boto3.client('kms', region_name=aws_region)
```

```
# Optional: Create an encryption context
encryption_context: Dict[str, str] = {
    "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 material providers
mat_prov: AwsCryptographicMaterialProviders = AwsCryptographicMaterialProviders(
    config=MaterialProvidersConfig()
)

# Create the AWS KMS regional discovery keyring
regional_discovery_keyring_input: CreateAwsKmsMrkDiscoveryKeyringInput = \
    CreateAwsKmsMrkDiscoveryKeyringInput(
        kms_client=kms_client,
        region=mrk_replica_decrypt_region,
        discovery_filter=DiscoveryFilter(
            account_ids=[111122223333],
            partition="aws"
        )
)

    regional_discovery_keyring: IKeyring =
 mat_prov.create_aws_kms_mrk_discovery_keyring(
        input=regional_discovery_keyring_input
    )
```

Rust

```
// Instantiate the AWS Encryption SDK
let esdk_config = AwsEncryptionSdkConfig::builder().build()?;
let esdk_client = esdk_client::Client::from_conf(esdk_config)?;

// Optional: Create an encryption context
let encryption_context = HashMap::from([
    ("encryption".to_string(), "context".to_string()),
    ("is not".to_string(), "secret".to_string()),
    ("but adds".to_string(), "useful metadata".to_string()),
```

```
    ("that can help you".to_string(), "be confident that".to_string()),
    ("the data you are handling".to_string(), "is what you think it
 is".to_string()),
]);

// Instantiate the material providers library
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;

// Create an AWS KMS client
let decrypt_kms_config = aws_sdk_kms::config::Builder::from(&sdk_config)
    .region(Region::new(mrk_replica_decrypt_region.clone()))
    .build();
let decrypt_kms_client = aws_sdk_kms::Client::from_conf(decrypt_kms_config);


// Create discovery filter
let discovery_filter = DiscoveryFilter::builder()
    .account_ids(vec![aws_account_id.to_string()])
    .partition("aws".to_string())
    .build()?;

// Create the regional discovery keyring
let discovery_keyring = mpl
    .create_aws_kms_mrk_discovery_keyring()
    .kms_client(decrypt_kms_client)
    .region(mrk_replica_decrypt_region)
    .discovery_filter(discovery_filter)
    .send()
    .await?;
```

Go

```
import (
    "context"

 mpl "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygenerated"
 mpltypes "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygeneratedtypes"
 client "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygenerated"
```

```
 esdktypes "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygeneratedtypes"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/kms"
)

// Instantiate the AWS Encryption SDK client
encryptionClient, err := client.NewClient(esdktypes.AwsEncryptionSdkConfig{})
if err != nil {
    panic(err)
}

// Create an AWS KMS client
cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    panic(err)
}
kmsClient := kms.NewFromConfig(cfg, func(o *kms.Options) {
    o.Region = KmsKeyRegion
})

// Optional: Create an encryption context
encryptionContext := map[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",
}

// Create discovery filter
discoveryFilter := mpltypes.DiscoveryFilter{
    AccountIds: []string{awsAccountID},
    Partition:  "aws",
}

// Create the regional discovery keyring
awsKmsMrkDiscoveryInput := mpltypes.CreateAwsKmsMrkDiscoveryKeyringInput{
    KmsClient:       kmsClient,
    Region:          alternateRegionMrkKeyRegion,
    DiscoveryFilter: &discoveryFilter,
}
```

```
awsKmsMrkDiscoveryKeyring, err :=
 matProv.CreateAwsKmsMrkDiscoveryKeyring(context.Background(),
 awsKmsMrkDiscoveryInput)
if err != nil {
    panic(err)
}
```

SDK de cifrado de AWS para JavaScript También exporta una función para Node.js y el navegador. `excludeRegions` Esta función crea un conjunto de claves de detección AWS KMS regional que omite determinadas AWS KMS keys regiones. En el siguiente ejemplo, se crea un conjunto de claves de detección AWS KMS regional que se puede usar AWS KMS keys en la cuenta 111122223333 en todos los países Región de AWS excepto en EE. UU. Este (Virginia del Norte) (us-east-1).

No SDK de cifrado de AWS para C tiene un método análogo, pero puede implementarlo creando uno personalizado. ClientSupplier

Este ejemplo muestra el código de 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 Llaveros jerárquicos

Con el anillo de claves AWS KMS jerárquico, puede proteger sus materiales criptográficos con una clave KMS de cifrado simétrico sin tener que llamar AWS KMS cada vez que cifra o descifra datos. Es una buena opción para las aplicaciones que necesitan minimizar las llamadas y las aplicaciones que pueden reutilizar algunos materiales criptográficos sin infringir sus requisitos de seguridad. AWS KMS

El anillo de claves jerárquico es una solución de almacenamiento en caché de materiales criptográficos que reduce el número de AWS KMS llamadas mediante el uso de claves de rama AWS KMS protegidas que se conservan en una tabla de Amazon DynamoDB y, a continuación, el almacenamiento en caché local de los materiales de clave de rama utilizados en las operaciones de

cifrado y descifrado. La tabla de DynamoDB sirve como almacén de claves que administra y protege las claves de rama. Almacena la clave de rama activa y todas las versiones anteriores de la clave de rama. La clave de rama activa es la versión más reciente de la clave de rama. El conjunto de claves jerárquico utiliza una clave de datos única para cifrar cada mensaje y cifra cada clave de cifrado de datos para cada solicitud de cifrado, y cifra cada clave de cifrado de datos con una clave de empaquetado única derivada de la clave de rama activa. El conjunto de claves jerárquico depende de la jerarquía establecida entre las claves de ramificación activas y las claves de encapsulamiento derivadas.

El conjunto de claves jerárquico suele utilizar cada versión de clave de rama para satisfacer múltiples solicitudes. Sin embargo, usted controla el grado en que se reutilizan las claves de rama activas y determina la frecuencia con la que se gira la clave de rama activa. La versión activa de la clave de rama permanece activa hasta que la gire. Las versiones anteriores de la clave de rama activa no se utilizarán para realizar operaciones de cifrado, pero sí se pueden consultar y utilizar en las operaciones de descifrado.

Al crear una instancia del conjunto de claves jerárquico, se crea una caché local. Se especifica un límite de caché que define el tiempo máximo durante el que los materiales de las claves de ramificación se almacenan en la caché local antes de que caduquen y se expulsen de la caché. El conjunto de claves jerárquico realiza una AWS KMS llamada para descifrar la clave de bifurcación y reunir los materiales de la clave de bifurcación la primera vez que se especifica a en una operación. `branch-key-id` A continuación, los materiales de la clave de rama se almacenan en la memoria caché local y se reutilizan para todas las operaciones de cifrado y descifrado que la `branch-key-id` especifique hasta que caduque el límite de la memoria caché. Almacenar los materiales de las claves de rama en la memoria caché local reduce las llamadas. AWS KMS Por ejemplo, considere un límite de caché de 15 minutos. Si realizas 10 000 operaciones de cifrado dentro de ese límite de caché, el conjunto de AWS KMS claves tradicional necesitaría realizar 10 000 AWS KMS llamadas para cumplir con 10 000 operaciones de cifrado. Si tiene uno activo`branch-key-id`, el conjunto de claves jerárquico solo necesita realizar una AWS KMS llamada para realizar 10 000 operaciones de cifrado.

La memoria caché local separa los materiales de cifrado de los materiales de descifrado. Los materiales de cifrado se ensamblan a partir de la clave de rama activa y se reutilizan en todas las operaciones de cifrado hasta que caduque el límite de la memoria caché. Los materiales de descifrado se recopilan a partir del identificador de la clave de ramificación y la versión que se identifica en los metadatos del campo cifrado, y se reutilizan para todas las operaciones de descifrado relacionadas con el identificador y la versión de la clave de bifurcación hasta que venza el límite de memoria caché. La memoria caché local puede almacenar varias versiones de la misma

clave de rama a la vez. Cuando la caché local está configurada para usar unabranch key ID supplier, también puede almacenar materiales de claves de rama de varias claves de rama activas a la vez.

> **Note**
>
> Todas las menciones del conjunto de claves jerárquico en el AWS Encryption SDK se refieren al conjunto de claves AWS KMS jerárquico.

Compatibilidad con lenguajes de programación

El conjunto de claves jerárquico es compatible con los siguientes lenguajes y versiones de programación:

- Versión 3. x del SDK de cifrado de AWS para Java
- Versión 4. x del AWS Encryption SDK para .NET
- Versión 4. x de SDK de cifrado de AWS para Python, cuando se usa con la dependencia MPL opcional.
- Versión 1. x de la AWS Encryption SDK para Rust
- Versión 0.1. x o posterior del AWS Encryption SDK for Go

Temas

- Funcionamiento
- Requisitos previos
- Permisos necesarios
- Elige un caché
- Crear un conjunto de claves jerárquico

Funcionamiento

Los siguientes tutoriales describen cómo el conjunto de claves jerárquico reúne los materiales de cifrado y descifrado, y las diferentes llamadas que realiza el conjunto de claves para las operaciones de cifrado y descifrado. Para obtener detalles técnicos sobre los procesos de derivación de claves de ajuste y cifrado de claves de datos de texto no cifrado, consulte Detalles técnicos del conjunto de claves jerárquico de AWS KMS.

Cifra y firma

El siguiente tutorial describe cómo el conjunto de claves jerárquico reúne los materiales de cifrado y obtiene una clave de encapsulamiento única.

1. El método de cifrado solicita materiales de cifrado al conjunto de claves jerárquico. El conjunto de claves genera una clave de datos de texto no cifrado y, a continuación, comprueba si hay materiales de derivación válidos en la memoria caché local para generar la clave de encapsulamiento. Si hay materiales de clave de rama válidos, el llavero continúa con el paso 4.
2. Si no hay ningún material de clave de bifurcación válido, el conjunto de claves jerárquico consulta el almacén de claves para encontrar la clave de bifurcación activa.

    a. El almacén de claves llama AWS KMS para descifrar la clave de rama activa y devuelve la clave de rama activa en texto plano. Los datos que identifican la clave de rama activa se serializan para proporcionar datos autenticados adicionales (AAD) en la llamada de descifrado a AWS KMS.

    b. El almacén de claves devuelve la clave de rama en texto simple y los datos que la identifican, como la versión de la clave de rama.
3. El conjunto de claves jerárquico reúne los materiales de las claves de rama (las versiones de las claves de rama y las claves de rama en texto no cifrado) y guarda una copia de los mismos en la memoria caché local.
4. El conjunto de claves jerárquico obtiene una clave de ajuste única de la clave de rama de texto simple y de una sal aleatoria de 16 bytes. Utiliza la clave de encapsulamiento derivada para cifrar una copia de la clave de datos de texto no cifrado.

El método de cifrado utiliza los materiales de cifrado para cifrar los datos. Para obtener más información, consulte Cómo se AWS Encryption SDK cifran los datos.

Descifrar y verificar

El siguiente tutorial describe cómo el conjunto de claves jerárquico reúne los materiales de descifrado y descifra la clave de datos cifrados.

1. El método de descifrado identifica la clave de datos cifrados del mensaje cifrado y la pasa al conjunto de claves jerárquico.
2. El conjunto de claves jerárquico deserializa los datos que identifican la clave de datos cifrada, incluida la versión de la clave de rama, la sal de 16 bytes y otra información que describe cómo se cifró la clave de datos.

Para obtener más información, consulte AWS KMS Detalles técnicos del llavero jerárquico.

3. El conjunto de claves jerárquico comprueba si hay materiales de clave de rama válidos en la caché local que coincidan con la versión de clave de rama identificada en el paso 2. Si hay materiales de clave de rama válidos, el conjunto de claves continúa con el paso 6.
4. Si no hay ningún material de clave de rama válido, el conjunto de claves jerárquico consulta en el almacén de claves la clave de rama que coincida con la versión de clave de rama identificada en el paso 2.

    a. El almacén de claves llama AWS KMS para descifrar la clave de bifurcación y devuelve la clave de bifurcación activa en texto plano. Los datos que identifican la clave de rama activa se serializan para proporcionar datos autenticados adicionales (AAD) en la llamada de descifrado a AWS KMS.

    b. El almacén de claves devuelve la clave de rama en texto simple y los datos que la identifican, como la versión de la clave de rama.
5. El conjunto de claves jerárquico reúne los materiales de las claves de rama (las versiones de las claves de rama y las claves de rama en texto no cifrado) y guarda una copia de los mismos en la memoria caché local.
6. El conjunto de claves jerárquico utiliza los materiales de clave de rama ensamblados y la sal de 16 bytes identificada en el paso 2 para reproducir la clave de encapsulamiento única que cifró la clave de datos.
7. El conjunto de claves jerárquico utiliza la clave de encapsulamiento para descifrar la clave de datos y devuelve la clave de datos en texto no cifrado.

El método de descifrado utiliza los materiales de descifrado y la clave de datos en texto no cifrado para descifrar el mensaje cifrado. Para obtener más información, consulte Cómo se AWS Encryption SDK descifra un mensaje cifrado.

Requisitos previos

Antes de crear y utilizar un conjunto de claves jerárquico, asegúrese de que se cumplen los siguientes requisitos previos.

- Usted, o el administrador del almacén de claves, ha creado un almacén de claves y ha creado al menos una clave de rama activa.
- Ha configurado las acciones del almacén de claves.

> **Note**
>
> La forma en que configure las acciones del almacén de claves determina qué operaciones puede realizar y qué claves de KMS puede utilizar el conjunto de claves jerárquico. Para obtener más información, consulte Acciones del almacén de claves.

- Dispone de los AWS KMS permisos necesarios para acceder y utilizar las llaves del almacén de claves y de la sucursal. Para obtener más información, consulte the section called “Permisos necesarios”.
- Ha revisado los tipos de caché compatibles y ha configurado el tipo de caché que mejor se adapta a sus necesidades. Para obtener más información, consulte the section called “Elige un caché”

Permisos necesarios

AWS Encryption SDK No requiere Cuenta de AWS ni depende de ninguno Servicio de AWS. Sin embargo, para usar un conjunto de claves jerárquico, necesitas Cuenta de AWS los siguientes permisos mínimos en los AWS KMS key cifrados simétricos de tu almacén de claves.

- Para cifrar y descifrar datos con el anillo de claves jerárquico, necesita KMS:Decrypt.
- Para crear y rotar claves de ramificación, necesita kms: y kms:. GenerateDataKeyWithoutPlaintext ReEncrypt

Para obtener más información sobre cómo controlar el acceso a las llaves de su sucursal y al almacén de claves, consultethe section called “Implementación de permisos de privilegio mínimo”.

Elige un caché

El conjunto de claves jerárquico reduce la cantidad de llamadas AWS KMS realizadas al almacenar en caché local los materiales de las claves de sucursal que se utilizan en las operaciones de cifrado y descifrado. Antes de crear su conjunto de claves jerárquico, debe decidir qué tipo de caché desea utilizar. Puedes usar la caché predeterminada o personalizarla para que se adapte mejor a tus necesidades.

El conjunto de claves jerárquico admite los siguientes tipos de caché:

- the section called “Caché predeterminada”

- the section called “MultiThreaded caché”
- the section called “StormTracking caché”
- the section called “Caché compartida”

> **Important**
>
> Todos los tipos de caché compatibles están diseñados para ser compatibles con entornos de subprocesos múltiples.
> Sin embargo, cuando se usa con el SDK de cifrado de AWS para Python, el conjunto de claves jerárquico no es compatible con entornos de subprocesos múltiples. Para obtener más información, consulte el archivo README.rst de Python en el repositorio -library deaws-cryptographic-material-providers. GitHub

Caché predeterminada

La mayoría de usuarios no necesitará modificar sus requisitos de subprocesamiento. La caché predeterminada está diseñada para admitir entornos con muchos subprocesos múltiples. Cuando caduca una entrada de materiales de clave de bifurcación, la caché predeterminada impide que varios subprocesos llamen, AWS KMS ya que notifica a un subproceso que la entrada de materiales de clave de bifurcación va a caducar con 10 segundos de antelación. Esto garantiza que solo un subproceso envíe una solicitud AWS KMS para actualizar la caché.

La memoria StormTracking caché predeterminada y la caché admiten el mismo modelo de subprocesos, pero solo es necesario especificar la capacidad de entrada para utilizar la memoria caché predeterminada. Para personalizaciones de caché más detalladas, utilice. the section called “StormTracking caché”

A menos que desee personalizar el número de entradas de materiales clave de rama que se pueden almacenar en la memoria caché local, no necesita especificar un tipo de memoria caché al crear el conjunto de claves jerárquico. Si no especifica un tipo de caché, el conjunto de claves jerárquico utiliza el tipo de caché predeterminado y establece la capacidad de entrada en 1000.

Para personalizar la caché predeterminada, especifique los siguientes valores:

- Capacidad de entrada: limita el número de entradas de materiales clave de rama que se pueden almacenar en la caché local.

Java

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

C# / .NET

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

Python

```
default_cache = CacheTypeDefault(
    value=DefaultCache(
        entry_capacity=100
    )
)
```

Rust

```
let cache: CacheType = CacheType::Default(
    DefaultCache::builder()
        .entry_capacity(100)
        .build()?,
);
```

Go

```
cache := mpltypes.CacheTypeMemberDefault{
  Value: mpltypes.DefaultCache{
   EntryCapacity: 100,
  },
 }
```

MultiThreaded caché

El uso de la MultiThreaded memoria caché es seguro en entornos de subprocesos múltiples, pero no proporciona ninguna funcionalidad para minimizar las llamadas de Amazon AWS KMS DynamoDB. Como resultado, cuando una entrada de materiales clave de rama caduque, se notificará a todos los subprocesos al mismo tiempo. Esto puede provocar varias AWS KMS llamadas para actualizar la memoria caché.

Para usar la MultiThreaded caché, especifique los siguientes valores:

- Capacidad de entrada: limita el número de entradas de materiales clave de rama que se pueden almacenar en la caché local.
- Tamaño de la cola de poda de entrada: define el número de entradas que se deben podar si se alcanza la capacidad de entrada.

Java

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

C# / .NET

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

Python

```
multithreaded_cache = CacheTypeMultiThreaded(
    value=MultiThreadedCache(
        entry_capacity=100,
        entry_pruning_tail_size=1
```

```
    )
)
```

Rust

```
CacheType::MultiThreaded(
            MultiThreadedCache::builder()
                    .entry_capacity(100)
                    .entry_pruning_tail_size(1)
                    .build()?)
```

Go

```
var entryPruningTailSize int32 = 1
 cache := mpltypes.CacheTypeMemberMultiThreaded{
  Value: mpltypes.MultiThreadedCache{
   EntryCapacity:        100,
   EntryPruningTailSize: &entryPruningTailSize,
  },
 }
```

StormTracking caché

La StormTracking memoria caché está diseñada para soportar entornos con muchos subprocesos múltiples. Cuando una entrada de materiales de clave de rama caduca, la StormTracking caché evita que varios subprocesos AWS KMS llamen, ya que notifica a un subproceso que la entrada de materiales de clave de rama va a caducar con antelación. Esto garantiza que solo un subproceso envíe una solicitud AWS KMS para actualizar la caché.

Para usar la StormTracking caché, especifique los siguientes valores:

- Capacidad de entrada: limita el número de entradas de materiales clave de rama que se pueden almacenar en la caché local.

  Valor predeterminado: 1000 entradas
- Tamaño de la cola de poda de entrada: define el número de entradas de materiales clave para la rama que se deben podar a la vez.

  Valor predeterminado: 1 entrada

- Período de gracia: define el número de segundos antes de la caducidad durante los que se intenta actualizar los materiales clave de la rama.

  Valor predeterminado: 10 segundos
- Intervalo de gracia: define el número de segundos entre los intentos de actualizar los materiales clave de la rama.

  Valor predeterminado: 1 segundo
- Amplificador: define el número de intentos simultáneos que se pueden realizar para actualizar los materiales clave de la rama.

  Valor predeterminado: 20 intentos
- En tiempo de vuelo hasta la vida útil (TTL): define el número de segundos hasta que se agota el tiempo de espera para intentar actualizar los materiales clave de la rama. Cada vez que la caché devuelve `NoSuchEntry` en respuesta a un `GetCacheEntry`, se considera que esa clave de rama está en tránsito hasta que se escribe la misma clave con una entrada `PutCache`.

  Valor predeterminado: 10 segundos
- Suspender: define el número de milisegundos que debe permanecer inactivo un hilo si `fanOut` se supera.

  Valor predeterminado: 20 milisegundos

Java

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

C# / .NET

```
CacheType stormTrackingCache = new CacheType
```

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

Python

```
storm_tracking_cache = CacheTypeStormTracking(
    value=StormTrackingCache(
        entry_capacity=100,
        entry_pruning_tail_size=1,
        fan_out=20,
        grace_interval=1,
        grace_period=10,
        in_flight_ttl=10,
        sleep_milli=20
    )
)
```

Rust

```
CacheType::StormTracking(
                StormTrackingCache::builder()
                    .entry_capacity(100)
                    .entry_pruning_tail_size(1)
                    .grace_period(10)
                    .grace_interval(1)
                    .fan_out(20)
                    .in_flight_ttl(10)
                    .sleep_milli(20)
                    .build()?)
```

Go

```
var entryPruningTailSize int32 = 1
```

```
cache := mpltypes.CacheTypeMemberStormTracking{
 Value: mpltypes.StormTrackingCache{
  EntryCapacity:        100,
  EntryPruningTailSize: &entryPruningTailSize,
  GraceInterval:        1,
  GracePeriod:          10,
  FanOut:               20,
  InFlightTTL:          10,
  SleepMilli:           20,
 },
}
```

Caché compartida

De forma predeterminada, el conjunto de claves jerárquico crea una nueva caché local cada vez que se crea una instancia del conjunto de claves. Sin embargo, la caché compartida puede ayudar a conservar la memoria, ya que permite compartir una caché entre varios conjuntos de claves jerárquicos. En lugar de crear una nueva caché de materiales criptográficos para cada conjunto de claves jerárquico que cree una instancia, la caché compartida solo almacena una caché en la memoria, que puede ser utilizada por todos los anillos de claves jerárquicos que hacen referencia a ella. La caché compartida ayuda a optimizar el uso de la memoria al evitar la duplicación de materiales criptográficos entre los conjuntos de claves. En cambio, los conjuntos de claves jerárquicos pueden acceder a la misma caché subyacente, lo que reduce el consumo total de memoria.

Al crear la caché compartida, se sigue definiendo el tipo de caché. Puede especificar un the section called “Caché predeterminada”the section called “MultiThreaded caché”, o the section called “StormTracking caché” como tipo de caché, o sustituirlo por cualquier caché personalizada compatible.

Particiones

Varios conjuntos de claves jerárquicos pueden utilizar una única caché compartida. Al crear un conjunto de claves jerárquico con una caché compartida, puede definir un ID de partición opcional. El ID de partición distingue qué anillo de claves jerárquico está escribiendo en la memoria caché. Si dos anillos de claves jerárquicos hacen referencia al mismo ID de partición y al mismo ID de clave de ramalogical key store name, los dos anillos de claves compartirán las mismas entradas de caché en la caché. Si crea dos anillos de claves jerárquicos con la misma caché compartida,

pero con una partición diferente IDs, cada conjunto de claves solo accederá a las entradas de la caché desde su propia partición designada dentro de la caché compartida. Las particiones actúan como divisiones lógicas dentro de la caché compartida, lo que permite que cada conjunto de claves jerárquico funcione de forma independiente en la partición designada, sin interferir con los datos almacenados en la otra partición.

Si pretende reutilizar o compartir las entradas de la caché de una partición, debe definir su propio identificador de partición. Al pasar el ID de la partición a su conjunto de claves jerárquico, el conjunto de claves puede reutilizar las entradas de la caché que ya están presentes en la caché compartida, sin tener que recuperar y volver a autorizar los materiales de las claves de rama. Si no especifica un identificador de partición, se asignará automáticamente un identificador de partición único al conjunto de claves cada vez que cree una instancia del conjunto de claves jerárquico.

Los siguientes procedimientos muestran cómo crear una caché compartida con el tipo de caché predeterminado y pasarla a un anillo de claves jerárquico.

1. Cree una `CryptographicMaterialsCache` (CMC) mediante la biblioteca de proveedores de materiales (MPL).

   Java

   ```
   // Instantiate the MPL
   final MaterialProviders matProv =
       MaterialProviders.builder()
           .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
           .build();

   // Create a CacheType object for the Default cache
   final CacheType cache =
       CacheType.builder()
           .Default(DefaultCache.builder().entryCapacity(100).build())
           .build();

   // Create a CMC using the default cache
   final CreateCryptographicMaterialsCacheInput cryptographicMaterialsCacheInput =
       CreateCryptographicMaterialsCacheInput.builder()
           .cache(cache)
           .build();

   final ICryptographicMaterialsCache sharedCryptographicMaterialsCache =
       matProv.CreateCryptographicMaterialsCache(cryptographicMaterialsCacheInput);
   ```

C# / .NET

```
// Instantiate the MPL
var materialProviders = new MaterialProviders(new MaterialProvidersConfig());

// Create a CacheType object for the Default cache
var cache = new CacheType { Default = new DefaultCache{EntryCapacity = 100} };

// Create a CMC using the default cache
var cryptographicMaterialsCacheInput = new
 CreateCryptographicMaterialsCacheInput {Cache = cache};

var sharedCryptographicMaterialsCache =
 materialProviders.CreateCryptographicMaterialsCache(cryptographicMaterialsCacheInput);
```

Python

```
# Instantiate the MPL
mat_prov: AwsCryptographicMaterialProviders = AwsCryptographicMaterialProviders(
    config=MaterialProvidersConfig()
)

# Create a CacheType object for the default cache
cache: CacheType = CacheTypeDefault(
    value=DefaultCache(
        entry_capacity=100,
    )
)

# Create a CMC using the default cache
cryptographic_materials_cache_input = CreateCryptographicMaterialsCacheInput(
    cache=cache,
)

shared_cryptographic_materials_cache =
 mat_prov.create_cryptographic_materials_cache(
    cryptographic_materials_cache_input
)
```

Rust

```
// Instantiate the MPL
```

```
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;

// Create a CacheType object for the default cache
let cache: CacheType = CacheType::Default(
    DefaultCache::builder()
        .entry_capacity(100)
        .build()?,
);

// Create a CMC using the default cache
let shared_cryptographic_materials_cache: CryptographicMaterialsCacheRef = mpl.
    create_cryptographic_materials_cache()
    .cache(cache)
    .send()
    .await?;
```

Go

```
import (
    "context"

 mpl "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygenerated"
 mpltypes "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygeneratedtypes"
)

// Instantiate the MPL
matProv, err := mpl.NewClient(mpltypes.MaterialProvidersConfig{})
if err != nil {
    panic(err)
}

// Create a CacheType object for the default cache
cache := mpltypes.CacheTypeMemberDefault{
    Value: mpltypes.DefaultCache{
        EntryCapacity: 100,
    },
}

// Create a CMC using the default cache
cmcCacheInput := mpltypes.CreateCryptographicMaterialsCacheInput{
```

```
    Cache: &cache,
}
sharedCryptographicMaterialsCache, err :=
 matProv.CreateCryptographicMaterialsCache(context.Background(), cmcCacheInput)
if err != nil {
    panic(err)
}
```

2. Cree un `CacheType` objeto para la caché compartida.

   Pase lo `sharedCryptographicMaterialsCache` que creó en el paso 1 al nuevo `CacheType` objeto.

   Java

   ```
   // Create a CacheType object for the sharedCryptographicMaterialsCache
   final CacheType sharedCache =
       CacheType.builder()
           .Shared(sharedCryptographicMaterialsCache)
           .build();
   ```

   C# / .NET

   ```
   // Create a CacheType object for the sharedCryptographicMaterialsCache
   var sharedCache = new CacheType { Shared = sharedCryptographicMaterialsCache };
   ```

   Python

   ```
   # Create a CacheType object for the shared_cryptographic_materials_cache
   shared_cache: CacheType = CacheTypeShared(
       value=shared_cryptographic_materials_cache
   )
   ```

   Rust

   ```
   // Create a CacheType object for the shared_cryptographic_materials_cache
   let shared_cache: CacheType =
    CacheType::Shared(shared_cryptographic_materials_cache);
   ```

Go

```
// Create a CacheType object for the shared_cryptographic_materials_cache
shared_cache :=
 mpltypes.CacheTypeMemberShared{sharedCryptographicMaterialsCache}
```

3. Pasa el `sharedCache` objeto del paso 2 a tu llavero jerárquico.

Al crear un conjunto de claves jerárquico con una caché compartida, si lo desea, puede definir un conjunto de claves jerárquico `partitionID` para compartir las entradas de la caché entre varios anillos de claves jerárquicos. Si no especifica un identificador de partición, el conjunto de claves jerárquico asigna automáticamente al conjunto de claves un identificador de partición único.

> **Note**
>
> Sus conjuntos de claves jerárquicos compartirán las mismas entradas de caché en una caché compartida si crea dos o más conjuntos de claves que hagan referencia al mismo identificador de partición y al mismo identificador de clave de rama. logical key store name Si no desea que varios conjuntos de claves compartan las mismas entradas de caché, debe utilizar un identificador de partición único para cada conjunto de claves jerárquico.

En el siguiente ejemplo, se crea un conjunto de claves jerárquico con un límite branch key ID supplier de memoria caché de 600 segundos. Para obtener más información sobre los valores definidos en la siguiente configuración de anillo de claves jerárquico, consulte. the section called “Crear un conjunto de claves jerárquico”

Java

```
// Create the Hierarchical keyring
final CreateAwsKmsHierarchicalKeyringInput keyringInput =
    CreateAwsKmsHierarchicalKeyringInput.builder()
        .keyStore(keystore)
        .branchKeyIdSupplier(branchKeyIdSupplier)
        .ttlSeconds(600)
        .cache(sharedCache)
        .partitionID(partitionID)
```

```
        .build();
final IKeyring hierarchicalKeyring =
 matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);
```

C# / .NET

```
// Create the Hierarchical keyring
var createKeyringInput = new CreateAwsKmsHierarchicalKeyringInput
{
   KeyStore = keystore,
   BranchKeyIdSupplier = branchKeyIdSupplier,
   Cache = sharedCache,
   TtlSeconds = 600,
   PartitionId = partitionID
};
var keyring =
 materialProviders.CreateAwsKmsHierarchicalKeyring(createKeyringInput);
```

Python

```
# Create the Hierarchical keyring
keyring_input: CreateAwsKmsHierarchicalKeyringInput =
 CreateAwsKmsHierarchicalKeyringInput(
    key_store=keystore,
    branch_key_id_supplier=branch_key_id_supplier,
    ttl_seconds=600,
    cache=shared_cache,
    partition_id=partition_id
)

hierarchical_keyring: IKeyring = mat_prov.create_aws_kms_hierarchical_keyring(
    input=keyring_input
)
```

Rust

```
// Create the Hierarchical keyring
let keyring1 = mpl
    .create_aws_kms_hierarchical_keyring()
    .key_store(key_store1)
    .branch_key_id(branch_key_id.clone())
```

```
    // CryptographicMaterialsCacheRef is an Rc (Reference Counted), so if you
 clone it to
    // pass it to different Hierarchical Keyrings, it will still point to the
 same
    // underlying cache, and increment the reference count accordingly.
    .cache(shared_cache.clone())
    .ttl_seconds(600)
    .partition_id(partition_id.clone())
    .send()
    .await?;
```

Go

```
// Create the Hierarchical keyring
hkeyringInput := mpltypes.CreateAwsKmsHierarchicalKeyringInput{
    KeyStore:    keyStore1,
    BranchKeyId: &branchKeyId,
    TtlSeconds:  600,
    Cache:       &shared_cache,
    PartitionId: &partitionId,
}
keyring, err := matProv.CreateAwsKmsHierarchicalKeyring(context.Background(),
 hkeyringInput)
if err != nil {
    panic(err)
}
```

Crear un conjunto de claves jerárquico

Para crear un conjunto de claves jerárquico, debe proporcionar los siguientes valores:

- Un nombre de almacén de claves

  El nombre de la tabla de DynamoDB que usted o el administrador del almacén de claves crearon para que sirviera de almacén de claves.

-

  Un tiempo de vida límite de la memoria caché (TTL)

  La cantidad de tiempo en segundos que se puede utilizar una entrada de material de clave de la memoria caché local antes de que caduque. El límite de caché TTL determina la frecuencia con

la que el cliente llama AWS KMS para autorizar el uso de las claves de sucursal. El valor debe ser mayor que cero. Una vez expirado el límite de caché TTL, la entrada no se sirve nunca y se desalojará de la caché local.

- Un identificador de clave de rama

  Puede configurar de forma estática la clave de sucursal `branch-key-id` que identifique una única clave de rama activa en su almacén de claves o proporcionar un proveedor de identificadores de clave de sucursal.

  El proveedor del identificador de la clave de sucursal utiliza los campos almacenados en el contexto de cifrado para determinar qué clave de sucursal se necesita para descifrar un registro.

  Recomendamos encarecidamente utilizar un proveedor de ID de clave de sucursal para las bases de datos multiusuario, en las que cada inquilino tiene su propia clave de sucursal. Puedes usar el identificador de clave de sucursal del proveedor para crear un nombre descriptivo para tu clave IDs de sucursal y así poder reconocer fácilmente el identificador de clave de sucursal correcto para un inquilino específico. Por ejemplo, el nombre descriptivo le permite hacer referencia a una clave de rama como `tenant1` en lugar `deb3f61619-4d35-48ad-a275-050f87e15122`.

  Para las operaciones de descifrado, puede configurar de forma estática un único conjunto de claves jerárquicas para restringir el descifrado a un único usuario, o puede utilizar el proveedor del identificador de clave de sucursal para identificar qué inquilino es responsable de descifrar un registro.

- (Opcional) Una caché

  Si desea personalizar el tipo de caché o el número de entradas de materiales clave de rama que se pueden almacenar en la caché local, especifique el tipo de caché y la capacidad de entrada al inicializar el conjunto de claves.

  El conjunto de claves jerárquico admite los siguientes tipos de caché: predeterminada MultiThreaded StormTracking, y compartida. Para obtener más información y ejemplos que demuestren cómo definir cada tipo de caché, consulte. the section called “Elige un caché”

  Si no especifica una caché, el conjunto de claves jerárquico utiliza automáticamente el tipo de caché predeterminado y establece la capacidad de entrada en 1000.

- (Opcional) Un ID de partición

Si especifica elthe section called “Caché compartida”, puede definir opcionalmente un ID de partición. El ID de partición distingue qué conjunto de claves jerárquico está escribiendo en la memoria caché. Si pretende reutilizar o compartir las entradas de la caché de una partición, debe definir su propio ID de partición. Puede especificar cualquier cadena para el ID de la partición. Si no especifica un identificador de partición, se asigna automáticamente un identificador de partición único al conjunto de claves en el momento de la creación.

Para obtener más información, consulte Partitions.

> **Note**
>
> Sus conjuntos de claves jerárquicos compartirán las mismas entradas de caché en una caché compartida si crea dos o más conjuntos de claves que hagan referencia al mismo identificador de partición y al mismo identificador de clave de logical key store name rama. Si no desea que varios conjuntos de claves compartan las mismas entradas de caché, debe utilizar un identificador de partición único para cada conjunto de claves jerárquico.

- (Opcional) Una lista de tokens de concesión

  Si controla el acceso a la clave KMS de su conjunto de claves jerárquico mediante concesiones, debe proporcionar todos los tokens de concesión necesarios al inicializar el conjunto de claves.

Cree un conjunto de claves jerárquico con un ID de clave de rama estático

Los siguientes ejemplos muestran cómo crear un anillo de claves jerárquico con un identificador de clave de rama estáticothe section called “Caché predeterminada”, el TTL con un límite de caché de 600 segundos.

Java

```
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
final CreateAwsKmsHierarchicalKeyringInput keyringInput =
 CreateAwsKmsHierarchicalKeyringInput.builder()
        .keyStore(branchKeyStoreName)
        .branchKeyId(branch-key-id)
        .ttlSeconds(600)
        .build();
```

```
final Keyring hierarchicalKeyring =
 matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);
```

C# / .NET

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var keyringInput = new CreateAwsKmsHierarchicalKeyringInput
{
   KeyStore = keystore,
   BranchKeyId = branch-key-id,
   TtlSeconds = 600
};
var hierarchicalKeyring = matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);
```

Python

```
mat_prov: AwsCryptographicMaterialProviders = AwsCryptographicMaterialProviders(
    config=MaterialProvidersConfig()
)

keyring_input: CreateAwsKmsHierarchicalKeyringInput =
 CreateAwsKmsHierarchicalKeyringInput(
    key_store=keystore,
    branch_key_id=branch_key_id,
    ttl_seconds=600
)

hierarchical_keyring: IKeyring = mat_prov.create_aws_kms_hierarchical_keyring(
    input=keyring_input
)
```

Rust

```
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;

let hierarchical_keyring = mpl
        .create_aws_kms_hierarchical_keyring()
        .key_store(key_store.clone())
        .branch_key_id(branch_key_id)
        .ttl_seconds(600)
        .send()
```

```
        .await?;
```

Go

```
matProv, err := mpl.NewClient(mpltypes.MaterialProvidersConfig{})
if err != nil {
    panic(err)
}
hkeyringInput := mpltypes.CreateAwsKmsHierarchicalKeyringInput{
    KeyStore:    keyStore,
    BranchKeyId: &branchKeyID,
    TtlSeconds:  600,
}
hKeyRing, err := matProv.CreateAwsKmsHierarchicalKeyring(context.Background(),
 hkeyringInput)
if err != nil {
    panic(err)
}
```

Cree un conjunto de claves jerárquico con un proveedor de ID de clave de sucursal

Los siguientes procedimientos muestran cómo crear un anillo de claves jerárquico con un proveedor de ID de sucursal.

1. Cree un proveedor de ID de clave de sucursal

   En el siguiente ejemplo, se crean nombres descriptivos para dos claves de sucursal y se llama `CreateDynamoDbEncryptionBranchKeyIdSupplier` para crear un proveedor de ID de clave de sucursal.

   Java

   ```
   // Create friendly names for each branch-key-id
   class ExampleBranchKeyIdSupplier implements IDynamoDbKeyBranchKeyIdSupplier {
       private static String branchKeyIdForTenant1;
       private static String branchKeyIdForTenant2;

       public ExampleBranchKeyIdSupplier(String tenant1Id, String tenant2Id) {
           this.branchKeyIdForTenant1 = tenant1Id;
           this.branchKeyIdForTenant2 = tenant2Id;
   ```

```
    }
// Create the branch key ID supplier
final DynamoDbEncryption ddbEnc = DynamoDbEncryption.builder()
        .DynamoDbEncryptionConfig(DynamoDbEncryptionConfig.builder().build())
        .build();
final BranchKeyIdSupplier branchKeyIdSupplier =
 ddbEnc.CreateDynamoDbEncryptionBranchKeyIdSupplier(
    CreateDynamoDbEncryptionBranchKeyIdSupplierInput.builder()
            .ddbKeyBranchKeyIdSupplier(new ExampleBranchKeyIdSupplier(branch-
key-ID-tenant1, branch-key-ID-tenant2))
            .build()).branchKeyIdSupplier();
```

C# / .NET

```
// Create friendly names for each branch-key-id
 class ExampleBranchKeyIdSupplier : DynamoDbKeyBranchKeyIdSupplierBase {
    private String _branchKeyIdForTenant1;
    private String _branchKeyIdForTenant2;

    public ExampleBranchKeyIdSupplier(String tenant1Id, String tenant2Id) {
        this._branchKeyIdForTenant1 = tenant1Id;
        this._branchKeyIdForTenant2 = tenant2Id;
    }
// Create the branch key ID supplier
var ddbEnc = new DynamoDbEncryption(new DynamoDbEncryptionConfig());
var branchKeyIdSupplier = ddbEnc.CreateDynamoDbEncryptionBranchKeyIdSupplier(
    new CreateDynamoDbEncryptionBranchKeyIdSupplierInput
    {
        DdbKeyBranchKeyIdSupplier = new ExampleBranchKeyIdSupplier(branch-key-
ID-tenant1, branch-key-ID-tenant2)
    }).BranchKeyIdSupplier;
```

Python

```
# Create branch key ID supplier that maps the branch key ID to a friendly name
branch_key_id_supplier: IBranchKeyIdSupplier = ExampleBranchKeyIdSupplier(
        tenant_1_id=branch_key_id_a,
        tenant_2_id=branch_key_id_b,
)
```

Rust

```
// Create branch key ID supplier that maps the branch key ID to a friendly name
let branch_key_id_supplier = ExampleBranchKeyIdSupplier::new(
    &branch_key_id_a,
    &branch_key_id_b
);
```

Go

```
// Create branch key ID supplier that maps the branch key ID to a friendly name
keySupplier := branchKeySupplier{branchKeyA: branchKeyA, branchKeyB: branchKeyB}
```

2. Crear un conjunto de claves jerárquico

En los siguientes ejemplos, se inicializa un conjunto de claves jerárquico con el proveedor de ID de clave de sucursal creado en el paso 1, un TLL con un límite de caché de 600 segundos y un tamaño de caché máximo de 1000.

Java

```
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
final CreateAwsKmsHierarchicalKeyringInput keyringInput =
 CreateAwsKmsHierarchicalKeyringInput.builder()
        .keyStore(keystore)
        .branchKeyIdSupplier(branchKeyIdSupplier)
        .ttlSeconds(600)
        .cache(CacheType.builder() //OPTIONAL
                .Default(DefaultCache.builder()
                .entryCapacity(100)
                .build())
        .build();
final Keyring hierarchicalKeyring =
 matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);
```

C# / .NET

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var keyringInput = new CreateAwsKmsHierarchicalKeyringInput
```

```
{
   KeyStore = keystore,
   BranchKeyIdSupplier = branchKeyIdSupplier,
   TtlSeconds = 600,
   Cache = new CacheType
   {
        Default = new DefaultCache { EntryCapacity = 100 }
   }
};
var hierarchicalKeyring = matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);
```

Python

```
mat_prov: AwsCryptographicMaterialProviders = AwsCryptographicMaterialProviders(
    config=MaterialProvidersConfig()
)

keyring_input: CreateAwsKmsHierarchicalKeyringInput =
 CreateAwsKmsHierarchicalKeyringInput(
    key_store=keystore,
    branch_key_id_supplier=branch_key_id_supplier,
    ttl_seconds=600,
    cache=CacheTypeDefault(
        value=DefaultCache(
            entry_capacity=100
        )
    ),
)

hierarchical_keyring: IKeyring = mat_prov.create_aws_kms_hierarchical_keyring(
    input=keyring_input
)
```

Rust

```
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;

let hierarchical_keyring = mpl
    .create_aws_kms_hierarchical_keyring()
    .key_store(key_store.clone())
    .branch_key_id_supplier(branch_key_id_supplier)
    .ttl_seconds(600)
```

```
    .send()
    .await?;
```

Go

```
hkeyringInput := mpltypes.CreateAwsKmsHierarchicalKeyringInput{
    KeyStore:            keyStore,
    BranchKeyIdSupplier: &keySupplier,
    TtlSeconds:          600,
}
hKeyRing, err := matProv.CreateAwsKmsHierarchicalKeyring(context.Background(),
 hkeyringInput)
if err != nil {
    panic(err)
}
```

AWS KMS Llaveros ECDH

Un anillo de claves AWS KMS ECDH utiliza un acuerdo de clave asimétrico AWS KMS keyspara derivar una clave envolvente simétrica compartida entre dos partes. En primer lugar, el conjunto de claves utiliza el algoritmo de acuerdo de claves Elliptic Curve Diffie-Hellman (ECDH) para obtener un secreto compartido a partir de la clave privada del par de claves KMS del remitente y la clave pública del destinatario. A continuación, el conjunto de claves utiliza el secreto compartido para obtener la clave de empaquetado compartida que protege las claves de cifrado de datos. La función de derivación de claves que AWS Encryption SDK utiliza (`KDF_CTR_HMAC_SHA384`) para derivar la clave de empaquetado compartida cumple con las recomendaciones del NIST para la derivación de claves.

La función de derivación de claves devuelve 64 bytes de material de codificación. Para garantizar que ambas partes utilicen el material de codificación correcto, AWS Encryption SDK utiliza los primeros 32 bytes como clave de compromiso y los últimos 32 bytes como clave de empaquetado compartida. Al descifrar, si el conjunto de claves no puede reproducir la misma clave de compromiso y la misma clave de empaquetado compartida que están almacenadas en el texto cifrado del encabezado del mensaje, la operación no se realizará correctamente. Por ejemplo, si cifra los datos con un conjunto de claves configurado con la clave privada de Alice y la clave pública de Bob, un conjunto de claves configurado con la clave privada de Bob y la clave pública de Alice reproducirá la misma clave de compromiso y clave de empaquetado compartida y podrá descifrar los datos. Si la

clave pública de Bob no proviene de un par de claves KMS, entonces Bob puede crear un conjunto de claves ECDH sin procesar para descifrar los datos.

El anillo de claves AWS KMS ECDH cifra los datos con una clave simétrica mediante AES-GCM. A continuación, la clave de datos se cifra sobre con la clave de empaquetado compartida derivada mediante AES-GCM. Cada anillo de claves AWS KMS ECDH solo puede tener una clave de empaquetado compartida, pero puede incluir varios anillos de claves AWS KMS ECDH, solos o con otros, en un conjunto de claves múltiples.

Compatibilidad con lenguajes de programación

El conjunto de claves AWS KMS ECDH se introdujo en la versión 1.5.0 de la biblioteca de proveedores de material criptográfico (MPL) y es compatible con los siguientes lenguajes y versiones de programación:

- Versión 3. x del SDK de cifrado de AWS para Java
- Versión 4. x del AWS Encryption SDK para .NET
- Versión 4. x de SDK de cifrado de AWS para Python, cuando se usa con la dependencia MPL opcional.
- Versión 1. x de la AWS Encryption SDK para Rust
- Versión 0.1. x o posterior del AWS Encryption SDK for Go

Temas

- Permisos necesarios para los AWS KMS llaveros ECDH
- Crear un conjunto de claves ECDH AWS KMS
- Creación de un conjunto de claves AWS KMS de detección del ECDH

Permisos necesarios para los AWS KMS llaveros ECDH

AWS Encryption SDK No requiere una AWS cuenta y no depende de ningún AWS servicio. Sin embargo, para usar un llavero AWS KMS ECDH, necesitas una AWS cuenta y los siguientes permisos mínimos AWS KMS keys en tu llavero. Los permisos varían en función del esquema de acuerdo de claves que utilice.

- Para cifrar y descifrar datos mediante el esquema de acuerdo de `KmsPrivateKeyToStaticPublicKey` claves, necesita kms: GetPublicKey y kms: DeriveSharedSecret en el par de claves KMS asimétricas del remitente. Si proporciona

directamente la clave pública codificada en DER del remitente al crear una instancia de su conjunto de claves, solo necesitará el DeriveSharedSecret permiso kms: en el par de claves KMS asimétricas del remitente.

- Para descifrar los datos mediante el esquema de acuerdo de `KmsPublicKeyDiscovery` claves, necesita los GetPublicKey permisos kms: DeriveSharedSecret y kms: en el par de claves KMS asimétricas especificado.

Crear un conjunto de claves ECDH AWS KMS

Para crear un conjunto de claves AWS KMS ECDH que cifre y descifre los datos, debe utilizar el esquema de acuerdo de claves. `KmsPrivateKeyToStaticPublicKey` Para inicializar un anillo de claves AWS KMS ECDH con el esquema de acuerdo de `KmsPrivateKeyToStaticPublicKey` claves, proporcione los siguientes valores:

- ID del remitente AWS KMS key

  Debe identificar un par de claves KMS de curva elíptica (ECC) asimétrica recomendado por el NIST con un valor de. `KeyUsage` `KEY_AGREEMENT` La clave privada del remitente se utiliza para obtener el secreto compartido.

- (Opcional) Clave pública del remitente

  Debe ser una clave pública X.509 codificada en DER, también conocida como `SubjectPublicKeyInfo` (SPKI), según se define en el RFC 5280.

  La AWS KMS GetPublicKeyoperación devuelve la clave pública de un par de claves KMS asimétricas en el formato codificado DER requerido.

  Para reducir el número de AWS KMS llamadas que realiza tu llavero, puedes proporcionar directamente la clave pública del remitente. Si no se proporciona ningún valor para la clave pública del remitente, el llavero llama AWS KMS para recuperar la clave pública del remitente.

- La clave pública del destinatario

  Debe proporcionar la clave pública X.509 codificada en DER del destinatario, también conocida como `SubjectPublicKeyInfo` (SPKI), tal como se define en el RFC 5280.

  La AWS KMS GetPublicKeyoperación devuelve la clave pública de un par de claves KMS asimétricas en el formato codificado DER requerido.

- Especificación de curva

Identifica la especificación de la curva elíptica en los pares de claves especificados. Los pares de claves del remitente y del destinatario deben tener la misma especificación de curva.

Valores válidos: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

- (Opcional) Una lista de tokens de concesión

Si controla el acceso a la clave KMS de su conjunto de claves AWS KMS ECDH mediante concesiones, debe proporcionar todos los símbolos de concesión necesarios al inicializar el conjunto de claves.

C# / .NET

En el siguiente ejemplo, se crea un anillo de claves AWS KMS ECDH con la clave KMS del remitente, la clave pública del remitente y la clave pública del destinatario. En este ejemplo, se utiliza el `SenderPublicKey` parámetro opcional para proporcionar la clave pública del remitente. Si no proporciona la clave pública del remitente, el conjunto de claves llama AWS KMS para recuperar la clave pública del remitente. Los pares de claves del remitente y del destinatario están en la `ECC_NIST_P256` curva.

```
// 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

En el siguiente ejemplo, se crea un conjunto de claves AWS KMS ECDH con la clave KMS del remitente, la clave pública del remitente y la clave pública del destinatario. En este ejemplo, se utiliza el `senderPublicKey` parámetro opcional para proporcionar la clave pública del remitente. Si no proporciona la clave pública del remitente, el conjunto de claves llama AWS KMS para recuperar la clave pública del remitente. Los pares de claves del remitente y del destinatario están en la `ECC_NIST_P256` curva.

```
// 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();
```

Python

En el siguiente ejemplo, se crea un conjunto de claves AWS KMS ECDH con la clave KMS del remitente, la clave pública del remitente y la clave pública del destinatario. En este ejemplo, se

utiliza el `senderPublicKey` parámetro opcional para proporcionar la clave pública del remitente. Si no proporciona la clave pública del remitente, el conjunto de claves llama AWS KMS para recuperar la clave pública del remitente. Los pares de claves del remitente y del destinatario están en la `ECC_NIST_P256` curva.

```
import boto3
from aws_cryptographic_materialproviders.mpl.models import (
    CreateAwsKmsEcdhKeyringInput,
    KmsEcdhStaticConfigurationsKmsPrivateKeyToStaticPublicKey,
    KmsPrivateKeyToStaticPublicKeyInput,
)
from aws_cryptography_primitives.smithygenerated.aws_cryptography_primitives.models
 import ECDHCurveSpec

# Instantiate the material providers library
mat_prov: AwsCryptographicMaterialProviders = AwsCryptographicMaterialProviders(
    config=MaterialProvidersConfig()
)

# Retrieve public keys
# Must be DER-encoded X.509 public keys
bob_public_key = get_public_key_bytes("arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab")
alice_public_key = get_public_key_bytes("arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321")

# Create the AWS KMS ECDH static keyring
sender_keyring_input = CreateAwsKmsEcdhKeyringInput(
    kms_client = boto3.client('kms', region_name="us-west-2"),
    curve_spec = ECDHCurveSpec.ECC_NIST_P256,
    key_agreement_scheme =
 KmsEcdhStaticConfigurationsKmsPrivateKeyToStaticPublicKey(
        KmsPrivateKeyToStaticPublicKeyInput(
            sender_kms_identifier = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            sender_public_key = bob_public_key,
            recipient_public_key = alice_public_key,

        )
    )
)
```

```
keyring = mat_prov.create_aws_kms_ecdh_keyring(sender_keyring_input)
```

Rust

En el siguiente ejemplo, se crea un conjunto de claves AWS KMS ECDH con la clave KMS del remitente, la clave pública del remitente y la clave pública del destinatario. En este ejemplo, se utiliza el `sender_public_key` parámetro opcional para proporcionar la clave pública del remitente. Si no proporciona la clave pública del remitente, el conjunto de claves llama AWS KMS para recuperar la clave pública del remitente.

```
// Instantiate the AWS Encryption SDK client
let esdk_config = AwsEncryptionSdkConfig::builder().build()?;
let esdk_client = esdk_client::Client::from_conf(esdk_config)?;

// Create the AWS KMS client
let sdk_config =
 aws_config::load_defaults(aws_config::BehaviorVersion::latest()).await;
let kms_client = aws_sdk_kms::Client::new(&sdk_config);

// Optional: Create your encryption context
let encryption_context = HashMap::from([
    ("encryption".to_string(), "context".to_string()),
    ("is not".to_string(), "secret".to_string()),
    ("but adds".to_string(), "useful metadata".to_string()),
    ("that can help you".to_string(), "be confident that".to_string()),
    ("the data you are handling".to_string(), "is what you think it
 is".to_string()),
]);

// Retrieve public keys
// Must be DER-encoded X.509 keys
let public_key_file_content_sender =
 std::fs::read_to_string(Path::new(EXAMPLE_KMS_ECC_PUBLIC_KEY_FILENAME_SENDER))?;
let parsed_public_key_file_content_sender = parse(public_key_file_content_sender)?;
let public_key_sender_utf8_bytes = parsed_public_key_file_content_sender.contents();

let public_key_file_content_recipient =
 std::fs::read_to_string(Path::new(EXAMPLE_KMS_ECC_PUBLIC_KEY_FILENAME_RECIPIENT))?;
let parsed_public_key_file_content_recipient =
 parse(public_key_file_content_recipient)?;
let public_key_recipient_utf8_bytes =
 parsed_public_key_file_content_recipient.contents();
```

```
// Create KmsPrivateKeyToStaticPublicKeyInput
let kms_ecdh_static_configuration_input =
    KmsPrivateKeyToStaticPublicKeyInput::builder()
        .sender_kms_identifier(arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab)
        // Must be a UTF8 DER-encoded X.509 public key
        .sender_public_key(public_key_sender_utf8_bytes)
        // Must be a UTF8 DER-encoded X.509 public key
        .recipient_public_key(public_key_recipient_utf8_bytes)
        .build()?;

let kms_ecdh_static_configuration =
 KmsEcdhStaticConfigurations::KmsPrivateKeyToStaticPublicKey(kms_ecdh_static_configuration_i

// Instantiate the material providers library
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;

// Create AWS KMS ECDH keyring
let kms_ecdh_keyring = mpl
    .create_aws_kms_ecdh_keyring()
    .kms_client(kms_client)
    .curve_spec(ecdh_curve_spec)
    .key_agreement_scheme(kms_ecdh_static_configuration)
    .send()
    .await?;
```

Go

```
import (
    "context"

 mpl "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygenerated"
 mpltypes "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygeneratedtypes"
 client "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygenerated"
 esdktypes "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygeneratedtypes"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/kms"
)
```

```
// Instantiate the AWS Encryption SDK client
encryptionClient, err := client.NewClient(esdktypes.AwsEncryptionSdkConfig{})
if err != nil {
    panic(err)
}

// Create an AWS KMS client
cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    panic(err)
}
kmsClient := kms.NewFromConfig(cfg, func(o *kms.Options) {
    o.Region = KmsKeyRegion
})

// Optional: Create an encryption context
encryptionContext := map[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",
}

// Retrieve public keys
// Must be DER-encoded X.509 keys
publicKeySender, err := utils.LoadPublicKeyFromPEM(kmsEccPublicKeyFileNameSender)
if err != nil {
    panic(err)
}
publicKeyRecipient, err :=
 utils.LoadPublicKeyFromPEM(kmsEccPublicKeyFileNameRecipient)
if err != nil {
    panic(err)
}

// Create KmsPrivateKeyToStaticPublicKeyInput
kmsEcdhStaticConfigurationInput := mpltypes.KmsPrivateKeyToStaticPublicKeyInput{
    RecipientPublicKey:  publicKeyRecipient,
    SenderKmsIdentifier: arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab,
    SenderPublicKey:     publicKeySender,
}
```

```
kmsEcdhStaticConfiguration :=
 &mpltypes.KmsEcdhStaticConfigurationsMemberKmsPrivateKeyToStaticPublicKey{
    Value: kmsEcdhStaticConfigurationInput,
}

// Instantiate the material providers library
matProv, err := mpl.NewClient(mpltypes.MaterialProvidersConfig{})
if err != nil {
    panic(err)
}

// Create AWS KMS ECDH keyring
awsKmsEcdhKeyringInput := mpltypes.CreateAwsKmsEcdhKeyringInput{
    CurveSpec:          ecdhCurveSpec,
    KeyAgreementScheme: kmsEcdhStaticConfiguration,
    KmsClient:          kmsClient,
}
awsKmsEcdhKeyring, err := matProv.CreateAwsKmsEcdhKeyring(context.Background(),
 awsKmsEcdhKeyringInput)
if err != nil {
    panic(err)
}
```

Creación de un conjunto de claves AWS KMS de detección del ECDH

Al descifrar, se recomienda especificar las claves que pueden utilizar. AWS Encryption SDK Para seguir esta práctica recomendada, utilice un anillo de claves AWS KMS ECDH con el esquema de acuerdo de `KmsPrivateKeyToStaticPublicKey` claves. Sin embargo, también puede crear un anillo de claves de detección de AWS KMS ECDH, es decir, un anillo de claves de AWS KMS ECDH que pueda descifrar cualquier mensaje en el que la clave pública del par de claves KMS especificado coincida con la clave pública del destinatario almacenada en el texto cifrado del mensaje.

> **Important**
>
> Al descifrar los mensajes mediante el esquema de acuerdo de `KmsPublicKeyDiscovery` claves, acepta todas las claves públicas, independientemente de quién sea su propietario.

Para inicializar un conjunto de claves del AWS KMS ECDH con el esquema de acuerdo de `KmsPublicKeyDiscovery` claves, proporcione los siguientes valores:

- ID del destinatario AWS KMS key

  Debe identificar un par de claves KMS de curva elíptica (ECC) asimétrica recomendado por el NIST con un valor de. `KeyUsage` `KEY_AGREEMENT`

- Especificación de curva

  Identifica la especificación de la curva elíptica en el par de claves KMS del destinatario.

  Valores válidos: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

- (Opcional) Una lista de tokens de concesión

  Si controla el acceso a la clave KMS de su conjunto de claves AWS KMS ECDH mediante concesiones, debe proporcionar todos los símbolos de concesión necesarios al inicializar el conjunto de claves.

C# / .NET

En el siguiente ejemplo, se crea un anillo de claves de detección de AWS KMS ECDH con un par de claves KMS en la `ECC_NIST_P256` curva. Debe tener los DeriveSharedSecret permisos kms: GetPublicKey y kms: en el key pair de claves KMS especificado. Este conjunto de claves puede descifrar cualquier mensaje en el que la clave pública del par de claves KMS especificado coincida con la clave pública del destinatario almacenada en el texto cifrado del mensaje.

```
// 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/0987dcba-09fe-87dc-65ba-ab0987654321"
    }

};
var createKeyringInput = new CreateAwsKmsEcdhKeyringInput
{
    CurveSpec = ECDHCurveSpec.ECC_NIST_P256,
    KmsClient = new AmazonKeyManagementServiceClient(),
    KeyAgreementScheme = discoveryConfiguration
```

```
};
var keyring = materialProviders.CreateAwsKmsEcdhKeyring(createKeyringInput);
```

Java

En el siguiente ejemplo, se crea un anillo de claves de detección de AWS KMS ECDH con un par de claves KMS en la ECC_NIST_P256 curva. Debe tener los DeriveSharedSecret permisos kms: GetPublicKey y kms: en el key pair de claves KMS especificado. Este conjunto de claves puede descifrar cualquier mensaje en el que la clave pública del par de claves KMS especificado coincida con la clave pública del destinatario almacenada en el texto cifrado del mensaje.

```
// 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/0987dcba-09fe-87dc-65ba-ab0987654321").build()
        ).build())
    .build();
```

Python

En el siguiente ejemplo, se crea un anillo de claves de detección de AWS KMS ECDH con un par de claves KMS en la ECC_NIST_P256 curva. Debe tener los DeriveSharedSecret permisos kms: GetPublicKey y kms: en el key pair de claves KMS especificado. Este conjunto de claves puede descifrar cualquier mensaje en el que la clave pública del par de claves KMS especificado coincida con la clave pública del destinatario almacenada en el texto cifrado del mensaje.

```
import boto3
from aws_cryptographic_materialproviders.mpl.models import (
    CreateAwsKmsEcdhKeyringInput,
    KmsEcdhStaticConfigurationsKmsPublicKeyDiscovery,
    KmsPublicKeyDiscoveryInput,
)
from aws_cryptography_primitives.smithygenerated.aws_cryptography_primitives.models
 import ECDHCurveSpec
```

```
# Instantiate the material providers library
mat_prov: AwsCryptographicMaterialProviders = AwsCryptographicMaterialProviders(
    config=MaterialProvidersConfig()
)

# Create the AWS KMS ECDH discovery keyring
create_keyring_input = CreateAwsKmsEcdhKeyringInput(
    kms_client = boto3.client('kms', region_name="us-west-2"),
    curve_spec = ECDHCurveSpec.ECC_NIST_P256,
    key_agreement_scheme = KmsEcdhStaticConfigurationsKmsPublicKeyDiscovery(
        KmsPublicKeyDiscoveryInput(
            recipient_kms_identifier = "arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321",
        )
    )
)

keyring = mat_prov.create_aws_kms_ecdh_keyring(create_keyring_input)
```

Rust

```
// Instantiate the AWS Encryption SDK client
let esdk_config = AwsEncryptionSdkConfig::builder().build()?;
let esdk_client = esdk_client::Client::from_conf(esdk_config)?;

// Create the AWS KMS client
let sdk_config =
 aws_config::load_defaults(aws_config::BehaviorVersion::latest()).await;
let kms_client = aws_sdk_kms::Client::new(&sdk_config);

// Optional: Create your encryption context
let encryption_context = HashMap::from([
    ("encryption".to_string(), "context".to_string()),
    ("is not".to_string(), "secret".to_string()),
    ("but adds".to_string(), "useful metadata".to_string()),
    ("that can help you".to_string(), "be confident that".to_string()),
    ("the data you are handling".to_string(), "is what you think it
 is".to_string()),
]);

// Create KmsPublicKeyDiscoveryInput
let kms_ecdh_discovery_static_configuration_input =
```

```
    KmsPublicKeyDiscoveryInput::builder()
        .recipient_kms_identifier(ecc_recipient_key_arn)
        .build()?;

let kms_ecdh_discovery_static_configuration =
 KmsEcdhStaticConfigurations::KmsPublicKeyDiscovery(kms_ecdh_discovery_static_configuration_

// Instantiate the material providers library
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;

// Create AWS KMS ECDH discovery keyring
let kms_ecdh_discovery_keyring = mpl
    .create_aws_kms_ecdh_keyring()
    .kms_client(kms_client.clone())
    .curve_spec(ecdh_curve_spec)
    .key_agreement_scheme(kms_ecdh_discovery_static_configuration)
    .send()
    .await?;
```

Go

```
import (
    "context"

 mpl "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygenerated"
 mpltypes "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygeneratedtypes"
 client "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygenerated"
 esdktypes "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygeneratedtypes"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/kms"
)

// Instantiate the AWS Encryption SDK client
encryptionClient, err := client.NewClient(esdktypes.AwsEncryptionSdkConfig{})
if err != nil {
    panic(err)
}
```

```
// Create an AWS KMS client
cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    panic(err)
}
kmsClient := kms.NewFromConfig(cfg, func(o *kms.Options) {
    o.Region = KmsKeyRegion
})

// Optional: Create an encryption context
encryptionContext := map[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",
}

// Create KmsPublicKeyDiscoveryInput
kmsEcdhDiscoveryStaticConfigurationInput := mpltypes.KmsPublicKeyDiscoveryInput{
    RecipientKmsIdentifier: eccRecipientKeyArn,
}
kmsEcdhDiscoveryStaticConfiguration :=
 &mpltypes.KmsEcdhStaticConfigurationsMemberKmsPublicKeyDiscovery{
    Value: kmsEcdhDiscoveryStaticConfigurationInput,
}

// Instantiate the material providers library
matProv, err := mpl.NewClient(mpltypes.MaterialProvidersConfig{})
if err != nil {
    panic(err)
}

// Create AWS KMS ECDH discovery keyring
awsKmsEcdhDiscoveryKeyringInput := mpltypes.CreateAwsKmsEcdhKeyringInput{
    CurveSpec:          ecdhCurveSpec,
    KeyAgreementScheme: kmsEcdhDiscoveryStaticConfiguration,
    KmsClient:          kmsClient,
}
awsKmsEcdhDiscoveryKeyring, err :=
 matProv.CreateAwsKmsEcdhKeyring(context.Background(),
 awsKmsEcdhDiscoveryKeyringInput)
if err != nil {
    panic(err)
```

```
}
```

Conjunto de claves de AES sin formato

AWS Encryption SDK Le permite usar una clave simétrica AES que proporciona como clave de empaquetado que protege su clave de datos. Debe generar, almacenar y proteger el material clave, preferiblemente en un módulo de seguridad de hardware (HSM) o en un sistema de administración de claves. Utilice un conjunto de claves de AES sin procesar cuando necesite proporcionar la clave de encapsulamiento y cifre las claves de datos de forma local o fuera de línea.

El conjunto de claves de AES sin formato usa el algoritmo AES-GCM y una clave de encapsulamiento que especifique como matriz de bytes para cifrar claves de datos. Puede especificar una sola clave de encapsulamiento en cada conjunto de claves de AES sin formato, pero puede incluir varios conjuntos de claves de AES sin formato en cada conjunto de claves múltiple.

El conjunto de claves AES sin procesar es equivalente e interactúa con la JceMasterKeyclase de SDK de cifrado de AWS para Java y la RawMasterKeyclase del SDK de cifrado de AWS para Python cuando se utiliza con claves de cifrado AES. Puede cifrar los datos con una implementación y descifrarlos con cualquier otra implementación mediante la misma clave de encapsulamiento. Para obtener más información, consulte Compatibilidad de conjuntos de claves.

Nombres y espacios de nombres clave

Para identificar la clave de AES, el conjunto de claves de AES sin formato utiliza un espacio de nombres de claves y nombre de clave que usted facilite. Estos valores no son secretos. Aparecen en texto no cifrado en el encabezado del mensaje cifrado que devuelve la operación de cifrado. Recomendamos utilizar un espacio de nombres clave en su HSM o sistema de administración de claves y un nombre de clave que identifique la clave AES en ese sistema.

> **Note**
>
> El espacio de nombres de clave y el nombre de clave son equivalentes a los campos ID de proveedor (o proveedor) e ID de clave en la `JceMasterKey` y la `RawMasterKey`.
> El SDK de cifrado de AWS para C y AWS Encryption SDK para .NET reservan el valor del espacio de nombres de la `aws-kms` clave para las claves de KMS. No utilice este valor de espacio de nombres en un conjunto de claves de AES sin procesar o en un conjunto de claves de RSA sin procesar con estas bibliotecas.

Si crea diferentes conjuntos de claves para cifrar y descifrar un mensaje determinado, el espacio de nombres y los valores de los nombres son fundamentales. Si el espacio de nombres y el nombre de la clave del conjunto de claves de descifrado no coinciden exactamente y distinguen mayúsculas de minúsculas entre el espacio de nombres de la clave y el nombre de la clave del conjunto de claves de cifrado, no se utiliza el conjunto de claves de descifrado, incluso si los bytes del material de la clave son idénticos.

Por ejemplo, puede definir un conjunto de claves de AES sin procesar con el espacio de nombres `HSM_01` y el nombre de la clave `AES_256_012`. A continuación, utilice ese conjunto de claves para cifrar algunos datos. Para descifrar esos datos, cree un conjunto de claves de AES sin procesar con el mismo espacio de nombres, nombre de clave y material de clave.

Los siguientes ejemplos muestran cómo crear un conjunto de claves de AES sin formato. La `AESWrappingKey` variable representa el material clave que proporciona.

C

Para crear una instancia de un conjunto de claves AES sin procesar en el, utilice. SDK de cifrado de AWS para C`aws_cryptosdk_raw_aes_keyring_new()` Para ver un ejemplo completo, consulte 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

Para crear un conjunto de claves AES sin procesar AWS Encryption SDK para .NET, utilice el método. `materialProviders.CreateRawAesKeyring()` Para ver un ejemplo completo, consulte Raw AESKeyring Example.cs.

En el ejemplo siguiente se utiliza la versión 4.x del AWS Encryption SDK para .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

El SDK de cifrado de AWS para JavaScript del navegador obtiene sus primitivas criptográficas de la WebCryptoAPI. Antes de construir el conjunto de claves, debes importar el material clave sin procesar `RawAesKeyringWebCrypto.importCryptoKey()` al backend. WebCrypto Esto garantiza que el conjunto de claves esté completo aunque todas las llamadas sean asíncronas. WebCrypto

A continuación, para crear una instancia de un conjunto de claves de AES sin procesar, utilice el método `RawAesKeyringWebCrypto()`. Debe especificar el algoritmo de encapsulamiento AES ("conjunto de encapsulamiento") en función de la longitud del material clave. Para ver un ejemplo completo, consulte aes_simple.ts (Browser). JavaScript

En el siguiente ejemplo, se utiliza la `buildClient` función para especificar la política de compromiso predeterminada,. `REQUIRE_ENCRYPT_REQUIRE_DECRYPT` También puede utilizarla `buildClient` para limitar el número de claves de datos cifradas en un mensaje cifrado. Para obtener más información, consulte the section called “Limitar las claves de datos cifrados”.

```
import {
  RawAesWrappingSuiteIdentifier,
  RawAesKeyringWebCrypto,
```

```
  synchronousRandomValues,
  buildClient,
  CommitmentPolicy,
} from '@aws-crypto/client-browser'

const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

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

Para crear una instancia de un conjunto de claves AES sin procesar en el SDK de cifrado de AWS para JavaScript archivo Node.js, cree una instancia de la clase. `RawAesKeyringNode` Debe especificar el algoritmo de encapsulamiento AES ("conjunto de encapsulamiento") en función de la longitud del material clave. Para ver un ejemplo completo, consulte aes_simple.ts (Node.js). JavaScript

En el siguiente ejemplo, se utiliza la `buildClient` función para especificar la política de compromiso predeterminada,. `REQUIRE_ENCRYPT_REQUIRE_DECRYPT` También puede utilizarla `buildClient` para limitar el número de claves de datos cifradas en un mensaje cifrado. Para obtener más información, consulte the section called “Limitar las claves de datos cifrados”.

```
import {
```

```
  RawAesKeyringNode,
  buildClient,
  CommitmentPolicy,
  RawAesWrappingSuiteIdentifier,
} from '@aws-crypto/client-node'

const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

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

Para crear una instancia de un conjunto de claves AES sin procesar en el SDK de cifrado de AWS para Java, utilice. `matProv.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);
```

Python

En el siguiente ejemplo, se crea una instancia del AWS Encryption SDK cliente con la política de compromiso predeterminada,. `REQUIRE_ENCRYPT_REQUIRE_DECRYPT` Para ver un ejemplo

completo, consulte raw_aes_keyring_example.py en el SDK de cifrado de AWS para Python repositorio de. GitHub

```
# Instantiate the AWS Encryption SDK client
client = aws_encryption_sdk.EncryptionSDKClient(
    commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

# Define the key namespace and key name
key_name_space = "HSM_01"
key_name = "AES_256_012"

# Optional: Create an encryption context
encryption_context: Dict[str, str] = {
    "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 material providers
mat_prov: AwsCryptographicMaterialProviders = AwsCryptographicMaterialProviders(
    config=MaterialProvidersConfig()
)

# Create Raw AES keyring
keyring_input: CreateRawAesKeyringInput = CreateRawAesKeyringInput(
    key_namespace=key_name_space,
    key_name=key_name,
    wrapping_key=AESWrappingKey,
    wrapping_alg=AesWrappingAlg.ALG_AES256_GCM_IV12_TAG16
)

raw_aes_keyring: IKeyring = mat_prov.create_raw_aes_keyring(
    input=keyring_input
)
```

Rust

```
// Instantiate the AWS Encryption SDK client
let esdk_config = AwsEncryptionSdkConfig::builder().build()?;
let esdk_client = esdk_client::Client::from_conf(esdk_config)?;
```

```
// Define the key namespace and key name
let key_namespace: &str = "HSM_01";
let key_name: &str = "AES_256_012";

// Optional: Create an encryption context
let encryption_context = HashMap::from([
    ("encryption".to_string(), "context".to_string()),
    ("is not".to_string(), "secret".to_string()),
    ("but adds".to_string(), "useful metadata".to_string()),
    ("that can help you".to_string(), "be confident that".to_string()),
    ("the data you are handling".to_string(), "is what you think it
 is".to_string()),
]);

// Instantiate the material providers library
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;

// Create Raw AES keyring
let raw_aes_keyring = mpl
    .create_raw_aes_keyring()
    .key_name(key_name)
    .key_namespace(key_namespace)
    .wrapping_key(aws_smithy_types::Blob::new(AESWrappingKey))
    .wrapping_alg(AesWrappingAlg::AlgAes256GcmIv12Tag16)
    .send()
    .await?;
```

Go

```
import (
    mpl "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygenerated"
 mpltypes "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygeneratedtypes"
    client "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygenerated"
    esdktypes "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygeneratedtypes"
)
//Instantiate the AWS Encryption SDK client.
```

```
encryptionClient, err := client.NewClient(esdktypes.AwsEncryptionSdkConfig{})
if err != nil {
    panic(err)
}
// Define the key namespace and key name
var keyNamespace = "A managed aes keys"
var keyName = "My 256-bit AES wrapping key"

// Optional: Create an encryption context
encryptionContext := map[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 material providers library
matProv, err := mpl.NewClient(mpltypes.MaterialProvidersConfig{})
if err != nil {
    panic(err)
}
// Create Raw AES keyring
aesKeyRingInput := mpltypes.CreateRawAesKeyringInput{
    KeyName:      keyName,
    KeyNamespace: keyNamespace,
    WrappingKey:  aesWrappingKey,
    WrappingAlg:  mpltypes.AesWrappingAlgAlgAes256GcmIv12Tag16,
}
aesKeyring, err := matProv.CreateRawAesKeyring(context.Background(),
 aesKeyRingInput)
if err != nil {
    panic(err)
}
```

Conjuntos de claves de RSA sin formato

El conjunto de claves de RSA sin formato realiza un cifrado y descifrado asimétrico de las claves de datos en la memoria local con las claves privadas y públicas de RSA que especifique. Debe generar, almacenar y proteger la clave privada, preferiblemente en un módulo de seguridad de hardware (HSM) o en un sistema de administración de claves. La función de cifrado cifra la clave de datos

bajo la clave pública de RSA. La función de descifrado descifra la clave de datos utilizando la clave privada. Puede seleccionar de entre los diversos modos de rellenado de RSA.

Un conjunto de claves de RSA sin formato que cifra y descifra debe incluir una clave pública asimétrica y un par de claves privadas. Sin embargo, puede cifrar datos con un conjunto de claves de RSA sin formato que solo tenga una clave pública y puede descifrar datos con un conjunto de claves de RSA sin formato que solo tenga una clave privada. Y puede incluir cualquier conjunto de claves de RSA sin formato en un conjunto de claves múltiple. Si configura un conjunto de claves de RSA sin procesar con una clave pública y una privada, asegúrese de que formen parte del mismo par de claves. Algunas implementaciones lingüísticas del no AWS Encryption SDK construirán un anillo de claves RSA sin procesar con claves de diferentes pares. Otros confían en usted para comprobar que sus claves pertenecen al mismo par de claves.

El anillo de claves RSA sin procesar es equivalente e interactúa con el que se encuentra JceMasterKeyen las claves de cifrado asimétrico RSA SDK de cifrado de AWS para Java y el RawMasterKeyinterior SDK de cifrado de AWS para Python cuando se utilizan con ellas. Puede cifrar los datos con una implementación y descifrarlos con cualquier otra implementación mediante la misma clave de encapsulamiento. Para obtener más información, consulte Compatibilidad de conjuntos de claves.

> **Note**
>
> El conjunto de claves de RSA no admite claves de KMS asimétricas. Si desea utilizar claves RSA KMS asimétricas, los siguientes lenguajes de programación admiten conjuntos de claves que utilizan RSA asimétrica: AWS KMS AWS KMS keys
>
> - Versión 3. x del SDK de cifrado de AWS para Java
> - Versión 4. x del AWS Encryption SDK para .NET
> - Versión 4. x de SDK de cifrado de AWS para Python, cuando se usa con la dependencia opcional de la biblioteca de proveedores de material criptográfico (MPL).
> - Versión 0.1. x o posterior del AWS Encryption SDK for Go
>
> Si cifra los datos con un conjunto de claves RSA sin procesar que incluye la clave pública de una clave RSA KMS, ni él AWS Encryption SDK ni AWS KMS ellos pueden descifrarlos. No puede exportar la clave privada de una clave KMS AWS KMS asimétrica a un conjunto de claves RSA sin procesar. La operación de AWS KMS descifrado no puede descifrar el mensaje cifrado que devuelve. AWS Encryption SDK

Al crear un anillo de claves RSA sin procesar en el SDK de cifrado de AWS para C, asegúrese de proporcionar el contenido del archivo PEM que incluye cada clave como una cadena C terminada en cero, no como una ruta o un nombre de archivo. Cuando cree un llavero de RSA sin formato en JavaScript, tenga en cuenta la posible incompatibilidad con otras implementaciones de lenguaje.

Espacios de nombres y nombres

Para identificar el par de claves, el conjunto de claves de RSA sin formato utiliza un espacio de nombres y nombre que usted facilite. Estos valores no son secretos. Aparecen en texto no cifrado en el encabezado del mensaje cifrado que devuelve la operación de cifrado. Recomendamos usar el espacio de nombres y el nombre de clave que identifican el par de claves RSA (o su clave privada) en su HSM o sistema de administración de claves.

> **Note**
>
> El espacio de nombres y el nombre de clave son equivalentes a los campos ID de proveedor (o proveedor) e ID de clave en `JceMasterKey` y `RawMasterKey`.
> SDK de cifrado de AWS para C Reserva el valor del espacio de nombres clave para las claves de `aws-kms` KMS. No lo utilice en un conjunto de claves de AES sin procesar ni en un conjunto de claves de RSA sin procesar con el SDK de cifrado de AWS para C.

Si crea diferentes conjuntos de claves para cifrar y descifrar un mensaje determinado, el espacio de nombres y los valores de los nombres son fundamentales. Si el espacio de nombres de clave y el nombre de clave del conjunto de claves de descifrado no coinciden exactamente y distinguen mayúsculas de minúsculas entre el espacio de nombres de clave y el nombre de clave del conjunto de claves de cifrado, no se utiliza el conjunto de claves de descifrado, incluso si las claves son del mismo par de claves.

El espacio de nombres de clave y el nombre de clave del material clave de los conjuntos de claves de cifrado y descifrado deben ser los mismos independientemente de que el conjunto de claves contenga la clave pública RSA, la clave privada RSA o ambas claves del par de claves. Por ejemplo, supongamos que cifra los datos con un conjunto de claves de RSA sin procesar para una clave pública RSA con el espacio de nombres `HSM_01` y el nombre de la clave `RSA_2048_06`. Para descifrar esos datos, cree un conjunto de claves de RSA sin procesar con la clave privada (o el mismo par de claves) y el mismo espacio de nombres y nombre de claves.

Modo de relleno

Debe especificar un modo de relleno para los conjunto de claves RSA sin formato utilizados para el cifrado y descifrado, o utilizar características de la implementación de su lenguaje que lo especifiquen por usted.

AWS Encryption SDK Admite los siguientes modos de relleno, sujetos a las limitaciones de cada idioma. Recomendamos un modo de relleno OAEP, especialmente el OAEP con relleno SHA-256 y SHA-256. MGF1 El modo de relleno solo se admite por motivos de compatibilidad con versiones anteriores. PKCS1

- OAEP con relleno SHA-1 y SHA-1 MGF1
- OAEP con relleno SHA-256 y SHA-256 MGF1
- OAEP con relleno SHA-384 y SHA-384 MGF1
- OAEP con relleno SHA-512 y SHA-512 MGF1
- PKCS1 Relleno v1.5

Los siguientes ejemplos muestran cómo crear un conjunto de claves RSA sin procesar con la clave pública y privada de un par de claves RSA y el OAEP con SHA-256 y con el modo de relleno SHA-256. MGF1 Las variables `RSAPublicKey` y `RSAPrivateKey` representan el material clave que proporciona.

C

Para crear un anillo de claves RSA sin procesar en el, utilice. SDK de cifrado de AWS para C`aws_cryptosdk_raw_rsa_keyring_new`

Al crear un anillo de claves RSA sin procesar en el SDK de cifrado de AWS para C, asegúrese de proporcionar el contenido del archivo PEM que incluye cada clave como una cadena C terminada en cero, no como una ruta o un nombre de archivo. Para ver un ejemplo completo, consulte 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

Para crear una instancia de un anillo de claves RSA sin procesar en para.NET, utilice el método. AWS Encryption SDK `materialProviders.CreateRawRsaKeyring()` Para ver un ejemplo completo, consulte Raw Example.cs. RSAKeyring

En el ejemplo siguiente se utiliza la versión 4.x del AWS Encryption SDK para .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

El SDK de cifrado de AWS para JavaScript del navegador obtiene sus primitivas criptográficas de la WebCryptobiblioteca. Antes de construir el conjunto de claves, debe importarlo

`importPublicKey()` and/or `importPrivateKey()` para importar el material clave sin procesar al backend. WebCrypto Esto garantiza que el conjunto de claves esté completo aunque todas las llamadas sean asíncronas. WebCrypto El objeto que utilizan los métodos de importación incluye el algoritmo de encapsulamiento y su modo de relleno.

Tras importar el material clave, utilice el método `RawRsaKeyringWebCrypto()` para crear una instancia del conjunto de claves. Al crear un conjunto de claves RSA sin procesar JavaScript, tenga en cuenta la posible incompatibilidad con las implementaciones de otros lenguajes.

En el siguiente ejemplo, se utiliza la `buildClient` función para especificar la política de compromiso predeterminada,. `REQUIRE_ENCRYPT_REQUIRE_DECRYPT` También puede utilizarla `buildClient` para limitar el número de claves de datos cifradas en un mensaje cifrado. Para obtener más información, consulte the section called “Limitar las claves de datos cifrados”.

Para ver un ejemplo completo, consulte rsa_simple.ts (Browser). JavaScript

```
import {
  RsaImportableKey,
  RawRsaKeyringWebCrypto,
  buildClient,
  CommitmentPolicy,
} from '@aws-crypto/client-browser'

const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

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

Para crear una instancia de un conjunto de claves RSA sin procesar SDK de cifrado de AWS para JavaScript para Node.js, cree una nueva instancia de la clase. `RawRsaKeyringNode` El parámetro `wrapKey` contiene la clave pública. El parámetro `unwrapKey` contiene la clave privada. El constructor `RawRsaKeyringNode` calcula el modo de relleno predeterminado por usted, aunque puede especificar el modo de relleno preferido.

Al crear un conjunto de claves RSA sin procesar JavaScript, tenga en cuenta la posible incompatibilidad con las implementaciones de otros lenguajes.

En el siguiente ejemplo, se utiliza la `buildClient` función para especificar la política de compromiso predeterminada,. `REQUIRE_ENCRYPT_REQUIRE_DECRYPT` También puede utilizarla `buildClient` para limitar el número de claves de datos cifradas en un mensaje cifrado. Para obtener más información, consulte the section called “Limitar las claves de datos cifrados”.

Para ver un ejemplo completo, consulte rsa_simple.ts (Node.js). JavaScript

```
import {
  RawRsaKeyringNode,
  buildClient,
  CommitmentPolicy,
} from '@aws-crypto/client-node'

const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

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);
```

Python

En el siguiente ejemplo, se crea una instancia del cliente con la política de compromiso predeterminada,. AWS Encryption SDK REQUIRE_ENCRYPT_REQUIRE_DECRYPT Para ver un ejemplo completo, consulte raw_rsa_keyring_example.py en el SDK de cifrado de AWS para Python repositorio de. GitHub

```
# Define the key namespace and key name
key_name_space = "HSM_01"
key_name = "RSA_2048_06"

# Instantiate the material providers
mat_prov: AwsCryptographicMaterialProviders = AwsCryptographicMaterialProviders(
    config=MaterialProvidersConfig()
)

# Create Raw RSA keyring
keyring_input: CreateRawRsaKeyringInput = CreateRawRsaKeyringInput(
    key_namespace=key_name_space,
    key_name=key_name,
    padding_scheme=PaddingScheme.OAEP_SHA256_MGF1,
    public_key=RSAPublicKey,
    private_key=RSAPrivateKey
)

raw_rsa_keyring: IKeyring = mat_prov.create_raw_rsa_keyring(
    input=keyring_input
)
```

Rust

```
// Instantiate the AWS Encryption SDK client
```

```
let esdk_config = AwsEncryptionSdkConfig::builder().build()?;
let esdk_client = esdk_client::Client::from_conf(esdk_config)?;

// Optional: Create an encryption context
let encryption_context = HashMap::from([
    ("encryption".to_string(), "context".to_string()),
    ("is not".to_string(), "secret".to_string()),
    ("but adds".to_string(), "useful metadata".to_string()),
    ("that can help you".to_string(), "be confident that".to_string()),
    ("the data you are handling".to_string(), "is what you think it
 is".to_string()),
]);

// Define the key namespace and key name
let key_namespace: &str = "HSM_01";
let key_name: &str = "RSA_2048_06";

// Instantiate the material providers library
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;

// Create Raw RSA keyring
let raw_rsa_keyring = mpl
    .create_raw_rsa_keyring()
    .key_name(key_name)
    .key_namespace(key_namespace)
    .padding_scheme(PaddingScheme::OaepSha256Mgf1)
    .public_key(aws_smithy_types::Blob::new(RSAPublicKey))
    .private_key(aws_smithy_types::Blob::new(RSAPrivateKey))
    .send()
    .await?;
```

Go

```
// Instantiate the material providers library
matProv, err :=
 awscryptographymaterialproviderssmithygenerated.NewClient(awscryptographymaterialproviderss

// Create Raw RSA keyring
rsaKeyRingInput :=
 awscryptographymaterialproviderssmithygeneratedtypes.CreateRawRsaKeyringInput{
 KeyName:       "rsa",
 KeyNamespace:  "rsa-keyring",
```

```
 PaddingScheme:
 awscryptographymaterialproviderssmithygeneratedtypes.PaddingSchemePkcs1,
 PublicKey:     pem.EncodeToMemory(publicKeyBlock),
 PrivateKey:    pem.EncodeToMemory(privateKeyBlock),
}

rsaKeyring, err := matProv.CreateRawRsaKeyring(context.Background(),
 rsaKeyRingInput)
```

Go

```
import (
    "context"

 mpl "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygenerated"
 mpltypes "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygeneratedtypes"
 client "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygenerated"
 esdktypes "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygeneratedtypes"
)

// Instantiate the AWS Encryption SDK client
encryptionClient, err := client.NewClient(esdktypes.AwsEncryptionSdkConfig{})
if err != nil {
    panic(err)
}

// Optional: Create an encryption context
encryptionContext := map[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",
}

// Define the key namespace and key name
var keyNamespace = "HSM_01"
var keyName = "RSA_2048_06"
```

```
// Instantiate the material providers library
matProv, err := mpl.NewClient(mpltypes.MaterialProvidersConfig{})
if err != nil {
    panic(err)
}

// Create Raw RSA keyring
rsaKeyRingInput := mpltypes.CreateRawRsaKeyringInput{
    KeyName:       keyName,
    KeyNamespace:  keyNamespace,
    PaddingScheme: mpltypes.PaddingSchemeOaepSha512Mgf1,
    PublicKey:     (RSAPublicKey),
    PrivateKey:    (RSAPrivateKey),
}
rsaKeyring, err := matProv.CreateRawRsaKeyring(context.Background(),
 rsaKeyRingInput)
if err != nil {
    panic(err)
}
```

Llaveros ECDH sin procesar

El anillo de claves ECDH sin procesar utiliza los pares de claves público-privadas de curva elíptica que usted proporciona para obtener una clave envolvente compartida entre dos partes. En primer lugar, el conjunto de claves obtiene un secreto compartido mediante la clave privada del remitente, la clave pública del destinatario y el algoritmo de acuerdo de claves Elliptic Curve Diffie-Hellman (ECDH). A continuación, el conjunto de claves utiliza el secreto compartido para obtener la clave de empaquetado compartida que protege las claves de cifrado de datos. La función de derivación de claves que AWS Encryption SDK utiliza (`KDF_CTR_HMAC_SHA384`) para derivar la clave de empaquetado compartida cumple con las recomendaciones del NIST para la derivación de claves.

La función de derivación de claves devuelve 64 bytes de material clave. Para garantizar que ambas partes utilicen el material clave correcto, AWS Encryption SDK utiliza los primeros 32 bytes como clave de compromiso y los últimos 32 bytes como clave de empaquetado compartida. Al descifrar, si el conjunto de claves no puede reproducir la misma clave de compromiso y la misma clave de empaquetado compartida que están almacenadas en el texto cifrado del encabezado del mensaje, la operación no se realizará correctamente. Por ejemplo, si cifra los datos con un conjunto de claves configurado con la clave privada de Alice y la clave pública de Bob, un conjunto de claves

configurado con la clave privada de Bob y la clave pública de Alice reproducirá la misma clave de compromiso y clave de empaquetado compartida y podrá descifrar los datos. Si la clave pública de Bob proviene de un AWS KMS key par, Bob puede crear un anillo de claves AWS KMS ECDH para descifrar los datos.

El anillo de claves ECDH sin procesar cifra los datos con una clave simétrica mediante AES-GCM. A continuación, la clave de datos se cifra sobre con la clave de empaquetado compartida derivada mediante AES-GCM. Cada anillo de claves ECDH sin procesar solo puede tener una clave de empaquetado compartida, pero puede incluir varios anillos de claves ECDH sin procesar, solos o con otros, en un conjunto de claves múltiples.

Usted es responsable de generar, almacenar y proteger sus claves privadas, preferiblemente en un módulo de seguridad de hardware (HSM) o en un sistema de administración de claves. Los pares de claves del remitente y del destinatario deben estar en la misma curva elíptica. AWS Encryption SDK Soporta las siguientes especificaciones de curva elíptica:

- `ECC_NIST_P256`
- `ECC_NIST_P384`
- `ECC_NIST_P512`

Compatibilidad con lenguajes de programación

El conjunto de claves ECDH sin procesar se introdujo en la versión 1.5.0 de la biblioteca de proveedores de material criptográfico (MPL) y es compatible con los siguientes lenguajes y versiones de programación:

- Versión 3. x del SDK de cifrado de AWS para Java
- Versión 4. x del AWS Encryption SDK para .NET
- Versión 4. x de SDK de cifrado de AWS para Python, cuando se usa con la dependencia MPL opcional.
- Versión 1. x de la AWS Encryption SDK para Rust
- Versión 0.1. x o posterior del AWS Encryption SDK for Go

Creación de un conjunto de claves ECDH sin procesar

El anillo de claves ECDH sin procesar admite tres esquemas de acuerdo clave:, y. `RawPrivateKeyToStaticPublicKey` `EphemeralPrivateKeyToStaticPublicKey`

`PublicKeyDiscovery` El esquema de acuerdo de claves que seleccione determina qué operaciones criptográficas puede realizar y cómo se ensamblan los materiales de codificación.

Temas

- RawPrivateKeyToStaticPublicKey
- EphemeralPrivateKeyToStaticPublicKey
- PublicKeyDiscovery

RawPrivateKeyToStaticPublicKey

Utilice el esquema de acuerdo de `RawPrivateKeyToStaticPublicKey` claves para configurar de forma estática la clave privada del remitente y la clave pública del destinatario en el conjunto de claves. Este esquema de acuerdo de claves puede cifrar y descifrar datos.

Para inicializar un conjunto de claves ECDH sin procesar con el esquema de acuerdo de `RawPrivateKeyToStaticPublicKey` claves, proporcione los siguientes valores:

- Clave privada del remitente

  Debe proporcionar la clave privada codificada en PEM del remitente (PrivateKeyInfo estructuras PKCS #8), tal como se define en el RFC 5958.
- La clave pública del destinatario

  Debe proporcionar la clave pública X.509 codificada en DER del destinatario, también conocida como `SubjectPublicKeyInfo` (SPKI), tal como se define en el RFC 5280.

  Puede especificar la clave pública de un par de claves KMS de un acuerdo de claves asimétrico o la clave pública de un par de claves generado fuera de AWS.
- Especificación de curva

  Identifica la especificación de la curva elíptica en los pares de claves especificados. Los pares de claves del remitente y del destinatario deben tener la misma especificación de curva.

  Valores válidos: `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

El siguiente ejemplo de Java utiliza el esquema de acuerdo de `RawPrivateKeyToStaticPublicKey` claves para configurar estáticamente la clave privada del remitente y la clave pública del destinatario. Ambos pares de claves están en la `ECC_NIST_P256` curva.

```
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);
}
```

Python

El siguiente ejemplo de Python usa el esquema de acuerdo de `RawEcdhStaticConfigurationsRawPrivateKeyToStaticPublicKey` claves para configurar estáticamente la clave privada del remitente y la clave pública del destinatario. Ambos pares de claves están en la `ECC_NIST_P256` curva.

```
import boto3
from aws_cryptographic_materialproviders.mpl.models import (
    CreateRawEcdhKeyringInput,
    RawEcdhStaticConfigurationsRawPrivateKeyToStaticPublicKey,
    RawPrivateKeyToStaticPublicKeyInput,
)
from aws_cryptography_primitives.smithygenerated.aws_cryptography_primitives.models
 import ECDHCurveSpec

# Instantiate the material providers library
mat_prov: AwsCryptographicMaterialProviders = AwsCryptographicMaterialProviders(
    config=MaterialProvidersConfig()
)

# Must be a PEM-encoded private key
bob_private_key = get_private_key_bytes()
# Must be a DER-encoded X.509 public key
```

```
alice_public_key = get_public_key_bytes()

# Create the raw ECDH static keyring
raw_keyring_input = CreateRawEcdhKeyringInput(
    curve_spec = ECDHCurveSpec.ECC_NIST_P256,
    key_agreement_scheme =
 RawEcdhStaticConfigurationsRawPrivateKeyToStaticPublicKey(
        RawPrivateKeyToStaticPublicKeyInput(
            sender_static_private_key = bob_private_key,
            recipient_public_key = alice_public_key,
        )
    )
)

keyring = mat_prov.create_raw_ecdh_keyring(raw_keyring_input)
```

Rust

El siguiente ejemplo de Python usa el esquema de acuerdo de `raw_ecdh_static_configuration` claves para configurar estáticamente la clave privada del remitente y la clave pública del destinatario. Ambos pares de claves deben estar en la misma curva.

```
// Instantiate the AWS Encryption SDK client
let esdk_config = AwsEncryptionSdkConfig::builder().build()?;
let esdk_client = esdk_client::Client::from_conf(esdk_config)?;

// Optional: Create your encryption context
let encryption_context = HashMap::from([
    ("encryption".to_string(), "context".to_string()),
    ("is not".to_string(), "secret".to_string()),
    ("but adds".to_string(), "useful metadata".to_string()),
    ("that can help you".to_string(), "be confident that".to_string()),
    ("the data you are handling".to_string(), "is what you think it
 is".to_string()),
]);

// Create keyring input
let raw_ecdh_static_configuration_input =
    RawPrivateKeyToStaticPublicKeyInput::builder()
        // Must be a UTF8 PEM-encoded private key
        .sender_static_private_key(private_key_sender_utf8_bytes)
        // Must be a UTF8 DER-encoded X.509 public key
```

```
        .recipient_public_key(public_key_recipient_utf8_bytes)
        .build()?;

let raw_ecdh_static_configuration =
 RawEcdhStaticConfigurations::RawPrivateKeyToStaticPublicKey(raw_ecdh_static_configuration_i

// Instantiate the material providers library
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;

// Create raw ECDH static keyring
let raw_ecdh_keyring = mpl
    .create_raw_ecdh_keyring()
    .curve_spec(ecdh_curve_spec)
    .key_agreement_scheme(raw_ecdh_static_configuration)
    .send()
    .await?;
```

Go

```
import (
    "context"

 mpl "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygenerated"
 mpltypes "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygeneratedtypes"
 client "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygenerated"
 esdktypes "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygeneratedtypes"
)

// Instantiate the AWS Encryption SDK client
encryptionClient, err := client.NewClient(esdktypes.AwsEncryptionSdkConfig{})
if err != nil {
    panic(err)
}

// Optional: Create your encryption context
encryptionContext := map[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",
}

// Create keyring input
rawEcdhStaticConfigurationInput := mpltypes.RawPrivateKeyToStaticPublicKeyInput{
    SenderStaticPrivateKey: privateKeySender,
    RecipientPublicKey:     publicKeyRecipient,
}
rawECDHStaticConfiguration :=
 &mpltypes.RawEcdhStaticConfigurationsMemberRawPrivateKeyToStaticPublicKey{
    Value: rawEcdhStaticConfigurationInput,
}
rawEcdhKeyRingInput := mpltypes.CreateRawEcdhKeyringInput{
    CurveSpec:          ecdhCurveSpec,
    KeyAgreementScheme: rawECDHStaticConfiguration,
}

// Instantiate the material providers library
matProv, err := mpl.NewClient(mpltypes.MaterialProvidersConfig{})
if err != nil {
    panic(err)
}

// Create raw ECDH static keyring
rawEcdhKeyring, err := matProv.CreateRawEcdhKeyring(context.Background(),
 rawEcdhKeyRingInput)
if err != nil {
    panic(err)
}
```

EphemeralPrivateKeyToStaticPublicKey

Los conjuntos de `EphemeralPrivateKeyToStaticPublicKey` claves configurados con el esquema de acuerdo de claves crean un nuevo par de claves localmente y derivan una clave de empaquetado compartida única para cada llamada de cifrado.

Este esquema de acuerdo de claves solo puede cifrar mensajes. Para descifrar los mensajes cifrados con el esquema de acuerdo de `EphemeralPrivateKeyToStaticPublicKey` claves, debe utilizar un esquema de acuerdo de claves de descubrimiento configurado con la clave pública del mismo destinatario. Para descifrar, puede usar un anillo de claves ECDH sin procesar con

el algoritmo de acuerdo de claves o, si la `PublicKeyDiscovery`clave pública del destinatario proviene de un par de claves KMS de acuerdo de claves asimétrico, puede usar un anillo de claves AWS KMS ECDH con el esquema de acuerdo de claves. KmsPublicKeyDiscovery

Para inicializar un conjunto de claves ECDH sin procesar con el esquema de acuerdo de claves, proporcione los siguientes valores`EphemeralPrivateKeyToStaticPublicKey`:

- Clave pública del destinatario

  Debe proporcionar la clave pública X.509 codificada en DER del destinatario, también conocida como `SubjectPublicKeyInfo` (SPKI), tal como se define en el RFC 5280.

  Puede especificar la clave pública de un par de claves KMS de un acuerdo de claves asimétrico o la clave pública de un par de claves generado fuera de AWS.
- Especificación de curva

  Identifica la especificación de la curva elíptica en la clave pública especificada.

  Al cifrar, el anillo de claves crea un nuevo par de claves en la curva especificada y utiliza la nueva clave privada y la clave pública especificada para obtener una clave de empaquetado compartida.

  Valores válidos: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

C# / .NET

En el siguiente ejemplo, se crea un anillo de claves ECDH sin procesar con el `EphemeralPrivateKeyToStaticPublicKey` esquema de acuerdo de claves. Al cifrar, el anillo de claves creará un nuevo par de claves localmente en la curva especificada`ECC_NIST_P256`.

```
// 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

En el siguiente ejemplo, se crea un anillo de claves ECDH sin procesar con el `EphemeralPrivateKeyToStaticPublicKey` esquema de acuerdo de claves. Al cifrar, el anillo de claves creará un nuevo par de claves localmente en la curva especificada`ECC_NIST_P256`.

```
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);
```

```
}
```

Python

En el siguiente ejemplo, se crea un anillo de claves ECDH sin procesar con el `RawEcdhStaticConfigurationsEphemeralPrivateKeyToStaticPublicKey` esquema de acuerdo de claves. Al cifrar, el anillo de claves creará un nuevo par de claves localmente en la curva especificada`ECC_NIST_P256`.

```
import boto3
from aws_cryptographic_materialproviders.mpl.models import (
    CreateRawEcdhKeyringInput,
    RawEcdhStaticConfigurationsEphemeralPrivateKeyToStaticPublicKey,
    EphemeralPrivateKeyToStaticPublicKeyInput,
)
from aws_cryptography_primitives.smithygenerated.aws_cryptography_primitives.models
 import ECDHCurveSpec

# Instantiate the material providers library
mat_prov: AwsCryptographicMaterialProviders = AwsCryptographicMaterialProviders(
    config=MaterialProvidersConfig()
)

# Your get_public_key_bytes must return a DER-encoded X.509 public key
recipient_public_key = get_public_key_bytes()

# Create the raw ECDH ephemeral private key keyring
ephemeral_input = CreateRawEcdhKeyringInput(
    curve_spec = ECDHCurveSpec.ECC_NIST_P256,
    key_agreement_scheme =
 RawEcdhStaticConfigurationsEphemeralPrivateKeyToStaticPublicKey(
        EphemeralPrivateKeyToStaticPublicKeyInput(
            recipient_public_key = recipient_public_key,
        )
    )
)

keyring = mat_prov.create_raw_ecdh_keyring(ephemeral_input)
```

Rust

En el siguiente ejemplo, se crea un anillo de claves ECDH sin procesar con el `ephemeral_raw_ecdh_static_configuration` esquema de acuerdo de claves. Al cifrar, el anillo de claves creará un nuevo par de claves localmente en la curva especificada.

```
// Instantiate the AWS Encryption SDK client
let esdk_config = AwsEncryptionSdkConfig::builder().build()?;
let esdk_client = esdk_client::Client::from_conf(esdk_config)?;

// Optional: Create your encryption context
let encryption_context = HashMap::from([
    ("encryption".to_string(), "context".to_string()),
    ("is not".to_string(), "secret".to_string()),
    ("but adds".to_string(), "useful metadata".to_string()),
    ("that can help you".to_string(), "be confident that".to_string()),
    ("the data you are handling".to_string(), "is what you think it
 is".to_string()),
]);

// Load public key from UTF-8 encoded PEM files into a DER encoded public key.
let public_key_file_content =
 std::fs::read_to_string(Path::new(EXAMPLE_ECC_PUBLIC_KEY_FILENAME_RECIPIENT))?;
let parsed_public_key_file_content = parse(public_key_file_content)?;
let public_key_recipient_utf8_bytes = parsed_public_key_file_content.contents();

// Create EphemeralPrivateKeyToStaticPublicKeyInput
let ephemeral_raw_ecdh_static_configuration_input =
    EphemeralPrivateKeyToStaticPublicKeyInput::builder()
        // Must be a UTF8 DER-encoded X.509 public key
        .recipient_public_key(public_key_recipient_utf8_bytes)
        .build()?;

let ephemeral_raw_ecdh_static_configuration =

 RawEcdhStaticConfigurations::EphemeralPrivateKeyToStaticPublicKey(ephemeral_raw_ecdh_static

// Instantiate the material providers library
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;

// Create raw ECDH ephemeral private key keyring
let ephemeral_raw_ecdh_keyring = mpl
```

```
    .create_raw_ecdh_keyring()
    .curve_spec(ecdh_curve_spec)
    .key_agreement_scheme(ephemeral_raw_ecdh_static_configuration)
    .send()
    .await?;
```

Go

```
import (
    "context"

 mpl "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygenerated"
 mpltypes "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygeneratedtypes"
 client "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygenerated"
 esdktypes "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygeneratedtypes"
)

// Instantiate the AWS Encryption SDK client
encryptionClient, err := client.NewClient(esdktypes.AwsEncryptionSdkConfig{})
if err != nil {
    panic(err)
}

// Optional: Create your encryption context
encryptionContext := map[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",
}

// Load public key from UTF-8 encoded PEM files into a DER encoded public key
publicKeyRecipient, err := LoadPublicKeyFromPEM(eccPublicKeyFileNameRecipient)
if err != nil {
    panic(err)
}

// Create EphemeralPrivateKeyToStaticPublicKeyInput
```

```
ephemeralRawEcdhStaticConfigurationInput :=
 mpltypes.EphemeralPrivateKeyToStaticPublicKeyInput{
    RecipientPublicKey: publicKeyRecipient,
}
ephemeralRawECDHStaticConfiguration :=
    mpltypes.RawEcdhStaticConfigurationsMemberEphemeralPrivateKeyToStaticPublicKey{
        Value: ephemeralRawEcdhStaticConfigurationInput,
    }

// Instantiate the material providers library
matProv, err := mpl.NewClient(mpltypes.MaterialProvidersConfig{})
if err != nil {
    panic(err)
}

// Create raw ECDH ephemeral private key keyring
rawEcdhKeyRingInput := mpltypes.CreateRawEcdhKeyringInput{
    CurveSpec:          ecdhCurveSpec,
    KeyAgreementScheme: &ephemeralRawECDHStaticConfiguration,
}
ecdhKeyring, err := matProv.CreateRawEcdhKeyring(context.Background(),
 rawEcdhKeyRingInput)
if err != nil {
    panic(err)
}
```

PublicKeyDiscovery

Al descifrar, se recomienda especificar las claves de empaquetado que pueden utilizar. AWS Encryption SDK Para seguir esta práctica recomendada, utilice un conjunto de claves ECDH que especifique tanto la clave privada del remitente como la clave pública del destinatario. Sin embargo, también puede crear un conjunto de claves de detección de ECDH sin procesar, es decir, un conjunto de claves ECDH sin procesar que pueda descifrar cualquier mensaje en el que la clave pública de la clave especificada coincida con la clave pública del destinatario almacenada en el texto cifrado del mensaje. Este esquema de acuerdo de claves solo puede descifrar mensajes.

> **Important**
>
> Al descifrar los mensajes mediante el esquema de acuerdo de `PublicKeyDiscovery` claves, acepta todas las claves públicas, independientemente de quién sea su propietario.

Para inicializar un conjunto de claves ECDH sin procesar con el esquema de acuerdo de `PublicKeyDiscovery` claves, proporcione los siguientes valores:

- Clave privada estática del destinatario

  Debe proporcionar la clave privada codificada en PEM del destinatario (PrivateKeyInfo estructuras PKCS #8), tal como se define en el RFC 5958.

- Especificación de curva

  Identifica la especificación de la curva elíptica en la clave privada especificada. Los pares de claves del remitente y del destinatario deben tener la misma especificación de curva.

  Valores válidos: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

C# / .NET

En el siguiente ejemplo, se crea un anillo de claves ECDH sin procesar con el esquema de acuerdo de `PublicKeyDiscovery` claves. Este conjunto de claves puede descifrar cualquier mensaje en el que la clave pública de la clave privada especificada coincida con la clave pública del destinatario almacenada en el texto cifrado del mensaje.

```
// 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

En el siguiente ejemplo, se crea un anillo de claves ECDH sin procesar con el esquema de acuerdo de claves. `PublicKeyDiscovery` Este conjunto de claves puede descifrar cualquier mensaje en el que la clave pública de la clave privada especificada coincida con la clave pública del destinatario almacenada en el texto cifrado del mensaje.

```
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()
        ).build();

    final IKeyring publicKeyDiscovery  =
 materialProviders.CreateRawEcdhKeyring(rawKeyringInput);
```

```
}
```

Python

En el siguiente ejemplo, se crea un anillo de claves ECDH sin procesar con el esquema de acuerdo de claves. `RawEcdhStaticConfigurationsPublicKeyDiscovery` Este conjunto de claves puede descifrar cualquier mensaje en el que la clave pública de la clave privada especificada coincida con la clave pública del destinatario almacenada en el texto cifrado del mensaje.

```
import boto3
from aws_cryptographic_materialproviders.mpl.models import (
    CreateRawEcdhKeyringInput,
    RawEcdhStaticConfigurationsPublicKeyDiscovery,
    PublicKeyDiscoveryInput,
)
from aws_cryptography_primitives.smithygenerated.aws_cryptography_primitives.models
 import ECDHCurveSpec

# Instantiate the material providers library
mat_prov: AwsCryptographicMaterialProviders = AwsCryptographicMaterialProviders(
    config=MaterialProvidersConfig()
)

# Your get_private_key_bytes must return a PEM-encoded private key
recipient_private_key = get_private_key_bytes()

# Create the raw ECDH discovery keyring
raw_keyring_input = CreateRawEcdhKeyringInput(
    curve_spec = ECDHCurveSpec.ECC_NIST_P256,
    key_agreement_scheme = RawEcdhStaticConfigurationsPublicKeyDiscovery(
        PublicKeyDiscoveryInput(
            recipient_static_private_key = recipient_private_key,
        )
    )
)

keyring = mat_prov.create_raw_ecdh_keyring(raw_keyring_input)
```

Rust

En el siguiente ejemplo, se crea un anillo de claves ECDH sin procesar con el esquema de acuerdo de claves. `discovery_raw_ecdh_static_configuration` Este conjunto de claves

puede descifrar cualquier mensaje en el que la clave pública de la clave privada especificada coincida con la clave pública del destinatario almacenada en el texto cifrado del mensaje.

```
// Instantiate the AWS Encryption SDK client and material providers library
let esdk_config = AwsEncryptionSdkConfig::builder().build()?;
let esdk_client = esdk_client::Client::from_conf(esdk_config)?;

let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;


// Optional: Create your encryption context
let encryption_context = HashMap::from([
    ("encryption".to_string(), "context".to_string()),
    ("is not".to_string(), "secret".to_string()),
    ("but adds".to_string(), "useful metadata".to_string()),
    ("that can help you".to_string(), "be confident that".to_string()),
    ("the data you are handling".to_string(), "is what you think it
 is".to_string()),
]);

// Load keys from UTF-8 encoded PEM files.
let mut file = File::open(Path::new(EXAMPLE_ECC_PRIVATE_KEY_FILENAME_RECIPIENT))?;
let mut private_key_recipient_utf8_bytes = Vec::new();
file.read_to_end(&mut private_key_recipient_utf8_bytes)?;

// Create PublicKeyDiscoveryInput
let discovery_raw_ecdh_static_configuration_input =
    PublicKeyDiscoveryInput::builder()
        // Must be a UTF8 PEM-encoded private key
        .recipient_static_private_key(private_key_recipient_utf8_bytes)
        .build()?;

let discovery_raw_ecdh_static_configuration =

 RawEcdhStaticConfigurations::PublicKeyDiscovery(discovery_raw_ecdh_static_configuration_inp

// Create raw ECDH discovery private key keyring
let discovery_raw_ecdh_keyring = mpl
    .create_raw_ecdh_keyring()
    .curve_spec(ecdh_curve_spec)
    .key_agreement_scheme(discovery_raw_ecdh_static_configuration)
    .send()
```

```
    .await?;
```

Go

```
import (
    "context"

 mpl "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygenerated"
 mpltypes "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygeneratedtypes"
 client "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygenerated"
 esdktypes "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygeneratedtypes"
)

// Instantiate the AWS Encryption SDK client
encryptionClient, err := client.NewClient(esdktypes.AwsEncryptionSdkConfig{})
if err != nil {
    panic(err)
}

// Optional: Create your encryption context
encryptionContext := map[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",
}

// Load keys from UTF-8 encoded PEM files.
privateKeyRecipient, err := os.ReadFile(eccPrivateKeyFileNameRecipient)
if err != nil {
    panic(err)
}

// Instantiate the material providers library
matProv, err := mpl.NewClient(mpltypes.MaterialProvidersConfig{})
if err != nil {
    panic(err)
}
```

```
// Create PublicKeyDiscoveryInput
discoveryRawEcdhStaticConfigurationInput := mpltypes.PublicKeyDiscoveryInput{
    RecipientStaticPrivateKey: privateKeyRecipient,
}

discoveryRawEcdhStaticConfiguration :=
 &mpltypes.RawEcdhStaticConfigurationsMemberPublicKeyDiscovery{
    Value: discoveryRawEcdhStaticConfigurationInput,
}

// Create raw ECDH discovery private key keyring
discoveryRawEcdhKeyringInput := mpltypes.CreateRawEcdhKeyringInput{
    CurveSpec:          ecdhCurveSpec,
    KeyAgreementScheme: discoveryRawEcdhStaticConfiguration,
}

discoveryRawEcdhKeyring, err := matProv.CreateRawEcdhKeyring(context.Background(),
 discoveryRawEcdhKeyringInput)
if err != nil {
    panic(err)
}
```

Conjuntos de claves múltiples

Puede combinar conjuntos de claves en un conjunto de claves múltiple. Un conjunto de claves múltiple es un conjunto de claves que consta de uno o varios conjuntos de claves individuales del mismo tipo o de un tipo distinto. El efecto equivale a utilizar varios conjuntos de claves en una serie. Cuando se utiliza un conjunto de claves múltiple para cifrar datos, cualquiera de las claves de encapsulamiento en cualquiera de los conjuntos de claves puede descifrar dichos datos.

Cuando crea un conjunto de claves múltiple para cifrar datos, designa uno de los conjuntos de claves como conjunto de claves generador. Los conjuntos de claves restantes se conocen como conjuntos de claves secundarios. El conjunto de claves generador genera y cifra la clave de datos de texto no cifrado. A continuación, todas las claves de encapsulamiento de todos los conjuntos de claves secundarios cifran la misma clave de datos en texto no cifrado. El conjunto de claves múltiple devuelve la clave de texto no cifrado y una clave de datos cifrada para cada clave de encapsulación del conjunto de claves múltiple. Si el conjunto de claves generador es un conjunto de claves KMS, la clave generadora del AWS KMS genera y cifra la clave en texto no cifrado. A continuación, todas

las claves adicionales AWS KMS keys del conjunto de AWS KMS claves y todas las claves de empaquetado de todos los anillos secundarios del conjunto de claves múltiples cifran la misma clave de texto sin formato.

Si crea un conjunto de claves múltiples sin un conjunto de claves generador, puede usarlo solo para descifrar datos, pero no para cifrarlos. O bien, si desea utilizar un conjunto de claves múltiples sin un conjunto de claves generador en las operaciones de cifrado, puede especificarlo como un conjunto de claves secundario en otro conjunto de claves múltiples. Un conjunto de claves múltiples sin un anillo de claves generador no puede designarse como el anillo de claves generador en otro conjunto de claves múltiples.

Al descifrar, AWS Encryption SDK utiliza los anillos de claves para intentar descifrar una de las claves de datos cifradas. Los conjuntos de claves se llaman en el orden en que están especificados en el conjunto de claves múltiple. El procesamiento se detiene tan pronto como cualquier clave de cualquier conjunto de claves pueda descifrar una clave de datos cifrada.

A partir de la versión 1.7. x, cuando una clave de datos cifrada se cifra en un anillo de claves AWS Key Management Service (AWS KMS) (o proveedor de claves maestras), AWS Encryption SDK siempre pasa la clave ARN del al parámetro de AWS KMS key`KeyId` la operación de AWS KMS descifrado. Esta es una práctica AWS KMS recomendada que garantiza que se descifra la clave de datos cifrados con la clave de empaquetado que se va a utilizar.

Para ver un ejemplo práctico de un conjunto de claves múltiple, consulte:

- C: multi_keyring.cpp
- C# //.NET: .cs MultiKeyringExample
- JavaScript Node.js: multi_keyring.ts
- JavaScript Navegador: multi_keyring.ts
- Java MultiKeyringExample: .java
- Python: multi_keyring_example.py

Para crear un conjunto de claves múltiple, en primer lugar, instancie los conjunto de claves secundarios. En este ejemplo, utilizamos un anillo de AWS KMS claves y un anillo de claves AES sin procesar, pero puede combinar cualquier conjunto de claves compatible en un conjunto de claves múltiples.

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

En el siguiente ejemplo, se utiliza la `buildClient` función para especificar la política de compromiso predeterminada,. `REQUIRE_ENCRYPT_REQUIRE_DECRYPT` También puede utilizarla `buildClient` para limitar el número de claves de datos cifradas en un mensaje cifrado. Para obtener más información, consulte the section called “Limitar las claves de datos cifrados”.

```
import {
  KmsKeyringBrowser,
  KMS,
  getClient,
  RawAesKeyringWebCrypto,
  RawAesWrappingSuiteIdentifier,
  MultiKeyringWebCrypto,
  buildClient,
  CommitmentPolicy,
  synchronousRandomValues,
} from '@aws-crypto/client-browser'

const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)
```

```
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

En el siguiente ejemplo, se utiliza la `buildClient` función para especificar la política de compromiso predeterminada,`REQUIRE_ENCRYPT_REQUIRE_DECRYPT`. También puede utilizarla `buildClient` para limitar el número de claves de datos cifradas en un mensaje cifrado. Para obtener más información, consulte the section called “Limitar las claves de datos cifrados”.

```
import {
  MultiKeyringNode,
  KmsKeyringNode,
  RawAesKeyringNode,
  RawAesWrappingSuiteIdentifier,
  buildClient,
  CommitmentPolicy,
} from '@aws-crypto/client-node'

const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

// 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);
```

Python

En el siguiente ejemplo, se crea una instancia del AWS Encryption SDK cliente con la política de compromiso predeterminada,. `REQUIRE_ENCRYPT_REQUIRE_DECRYPT`

```
# Create the AWS KMS keyring
kms_client = boto3.client('kms', region_name="us-west-2")

mat_prov: AwsCryptographicMaterialProviders = AwsCryptographicMaterialProviders(
    config=MaterialProvidersConfig()
)

kms_keyring_input: CreateAwsKmsKeyringInput = CreateAwsKmsKeyringInput(
    generator=arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab,
    kms_client=kms_client
)

kms_keyring: IKeyring = mat_prov.create_aws_kms_keyring(
    input=kms_keyring_input
)

# Create Raw AES keyring
key_name_space = "HSM_01"
key_name = "AES_256_012"
```

```
raw_aes_keyring_input: CreateRawAesKeyringInput = CreateRawAesKeyringInput(
    key_namespace=key_name_space,
    key_name=key_name,
    wrapping_key=AESWrappingKey,
    wrapping_alg=AesWrappingAlg.ALG_AES256_GCM_IV12_TAG16
)

raw_aes_keyring: IKeyring = mat_prov.create_raw_aes_keyring(
    input=raw_aes_keyring_input
)
```

Rust

```
// Instantiate the AWS Encryption SDK client
let esdk_config = AwsEncryptionSdkConfig::builder().build()?;
let esdk_client = esdk_client::Client::from_conf(esdk_config)?;

// Create the AWS KMS client
let sdk_config =
 aws_config::load_defaults(aws_config::BehaviorVersion::latest()).await;
let kms_client = aws_sdk_kms::Client::new(&sdk_config);

// Instantiate the material providers library
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;

// Create an AWS KMS keyring
let kms_keyring = mpl
    .create_aws_kms_keyring()
    .kms_key_id(kms_key_id)
    .kms_client(kms_client)
    .send()
    .await?;

// Create a Raw AES keyring
let key_namespace: &str = "my-key-namespace";
let key_name: &str = "my-aes-key-name";

let raw_aes_keyring = mpl
    .create_raw_aes_keyring()
    .key_name(key_name)
    .key_namespace(key_namespace)
    .wrapping_key(aws_smithy_types::Blob::new(AESWrappingKey))
```

```
    .wrapping_alg(AesWrappingAlg::AlgAes256GcmIv12Tag16)
    .send()
    .await?;
```

Go

```
import (
    "context"

 mpl "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygenerated"
 mpltypes "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygeneratedtypes"
 client "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygenerated"
 esdktypes "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygeneratedtypes"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/kms"
)

// Instantiate the AWS Encryption SDK client
encryptionClient, err := client.NewClient(esdktypes.AwsEncryptionSdkConfig{})
if err != nil {
    panic(err)
}

// Create an AWS KMS client
cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    panic(err)
}
kmsClient := kms.NewFromConfig(cfg, func(o *kms.Options) {
    o.Region = KmsKeyRegion
})

// Instantiate the material providers library
matProv, err := mpl.NewClient(mpltypes.MaterialProvidersConfig{})
if err != nil {
    panic(err)
}

// Create an AWS KMS keyring
```

```
awsKmsKeyringInput := mpltypes.CreateAwsKmsKeyringInput{
    KmsClient: kmsClient,
    KmsKeyId:  kmsKeyId,
}
awsKmsKeyring, err := matProv.CreateAwsKmsKeyring(context.Background(),
 awsKmsKeyringInput)
if err != nil {
    panic(err)
}

// Create a Raw AES keyring
var keyNamespace = "my-key-namespace"
var keyName = "my-aes-key-name"

aesKeyRingInput := mpltypes.CreateRawAesKeyringInput{
    KeyName:      keyName,
    KeyNamespace: keyNamespace,
    WrappingKey:  AESWrappingKey,
    WrappingAlg:  mpltypes.AesWrappingAlgAlgAes256GcmIv12Tag16,
}
aesKeyring, err := matProv.CreateRawAesKeyring(context.Background(),
 aesKeyRingInput)
```

A continuación, cree el conjunto de claves múltiple y especifique su conjunto de claves generador, si lo hay. En este ejemplo, creamos un conjunto de claves múltiples en el que el anillo de AWS KMS claves es el generador y el anillo de claves AES es el anillo secundario.

C

En el constructor del conjunto de claves múltiple en C, solo se especifica el conjunto de claves generador.

```
struct aws_cryptosdk_keyring *multi_keyring = aws_cryptosdk_multi_keyring_new(alloc,
 kms_keyring);
```

Para agregar un conjunto de claves secundario a un conjunto de claves múltiple, utilice el método `aws_cryptosdk_multi_keyring_add_child`. Debe llamar al método una vez por cada conjunto de claves secundario que agregue.

```
// Add the Raw AES keyring (C only)
```

```
aws_cryptosdk_multi_keyring_add_child(multi_keyring, aes_keyring);
```

C# / .NET

El constructor `CreateMultiKeyringInput` de .NET permite definir un conjunto de claves generador y conjuntos de claves secundarios. El objeto resultante `CreateMultiKeyringInput` es inmutable.

```
var createMultiKeyringInput = new CreateMultiKeyringInput
{
    Generator = kmsKeyring,
    ChildKeyrings = new List<IKeyring>() {aesKeyring}
};

var multiKeyring = materialProviders.CreateMultiKeyring(createMultiKeyringInput);
```

JavaScript Browser

JavaScript Los llaveros múltiples son inmutables. El constructor de JavaScript llaveros múltiples permite especificar el anillo de claves del generador y varios anillos de claves secundarios.

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

const multiKeyring = new MultiKeyringWebCrypto(generator: kmsKeyring, children:
 [aesKeyring]);
```

JavaScript Node.js

JavaScript los llaveros múltiples son inmutables. El constructor de JavaScript llaveros múltiples permite especificar el anillo de claves del generador y varios anillos de claves secundarios.

```
const multiKeyring = new MultiKeyringNode(generator: kmsKeyring, children:
 [aesKeyring]);
```

Java

El `CreateMultiKeyringInput` constructor de Java permite definir un conjunto de claves generador y conjuntos de claves secundarios. El objeto resultante `createMultiKeyringInput` es inmutable.

```
final CreateMultiKeyringInput createMultiKeyringInput =
 CreateMultiKeyringInput.builder()
```

```
        .generator(awsKmsMrkMultiKeyring)
        .childKeyrings(Collections.singletonList(rawAesKeyring))
        .build();
IKeyring multiKeyring = matProv.CreateMultiKeyring(createMultiKeyringInput);
```

Python

```
multi_keyring_input: CreateMultiKeyringInput = CreateMultiKeyringInput(
    generator=kms_keyring,
    child_keyrings=[raw_aes_keyring]
)

multi_keyring: IKeyring = mat_prov.create_multi_keyring(
    input=multi_keyring_input
)
```

Rust

```
let multi_keyring = mpl
    .create_multi_keyring()
    .generator(kms_keyring.clone())
    .child_keyrings(vec![raw_aes_keyring.clone()])
    .send()
    .await?;
```

Go

```
createMultiKeyringInput := mpltypes.CreateMultiKeyringInput{
  Generator:     awsKmsKeyring,
  ChildKeyrings: []mpltypes.IKeyring{rawAESKeyring},
 }
 multiKeyring, err := matProv.CreateMultiKeyring(context.Background(),
 createMultiKeyringInput)
 if err != nil {
  panic(err)
 }
```

Ahora, puede utilizar el conjunto de claves múltiple para cifrar y descifrar datos.

AWS Encryption SDK lenguajes de programación

AWS Encryption SDK Está disponible para los siguientes lenguajes de programación. Todas las implementaciones de lenguaje son interoperables. Puede cifrar con una implementación de lenguaje y descifrar con otra. La interoperabilidad puede estar sujeta a restricciones de lenguaje. Si es así, estas restricciones se describen en el tema que trata de la implementación del lenguaje. Además, al cifrar y descifrar, debe usar conjuntos de claves compatibles o claves maestras y proveedores de claves maestras. Para obtener información, consulte the section called “Compatibilidad de conjuntos de claves”.

Temas

- SDK de cifrado de AWS para C
- AWS Encryption SDK para.NET
- AWS Encryption SDK para Go
- SDK de cifrado de AWS para Java
- SDK de cifrado de AWS para JavaScript
- SDK de cifrado de AWS para Python
- AWS Encryption SDK para Rust
- AWS Encryption SDK interfaz de línea de comandos

SDK de cifrado de AWS para C

SDK de cifrado de AWS para C Proporciona una biblioteca de cifrado del lado del cliente para los desarrolladores que escriben aplicaciones en C. También sirve como base para las implementaciones AWS Encryption SDK en lenguajes de programación de nivel superior.

Como todas las implementaciones de, SDK de cifrado de AWS para C ofrece funciones avanzadas de AWS Encryption SDK protección de datos. Esto incluye el cifrado de sobre, la información autenticada adicional (AAD) y los conjuntos de algoritmos de clave simétrica autenticados y seguros, tales como AES-GCM de 256 bits con derivación de clave y firma.

Todas las implementaciones específicas del idioma son totalmente interoperables. AWS Encryption SDK Por ejemplo, puede cifrar datos con SDK de cifrado de AWS para C y descifrarlos con cualquier implementación de lenguaje compatible, incluida la AWS CLI de cifrado.

SDK de cifrado de AWS para C Requiere interactuar AWS SDK para C++ con AWS Key Management Service ()AWS KMS. Solo debe usarlo si está usando el conjunto de claves de AWS KMS opcional. Sin embargo, AWS Encryption SDK no requiere AWS KMS ningún otro AWS servicio.

Más información

- Para obtener más información sobre la programación con SDK de cifrado de AWS para C, consulta los ejemplos en C, los ejemplos del aws-encryption-sdk-c repositorio y la documentación de la SDK de cifrado de AWS para C API. GitHub
- Para obtener información sobre cómo utilizar el código SDK de cifrado de AWS para C para cifrar datos y poder descifrarlos en varias partes Regiones de AWS, consulte Cómo descifrar textos cifrados en varias regiones con la letra en C AWS Encryption SDK en el blog de seguridad. AWS

Temas

- Instalación del SDK de cifrado de AWS para C
- Uso del SDK de cifrado de AWS para C
- SDK de cifrado de AWS para C ejemplos

Instalación del SDK de cifrado de AWS para C

Instale la versión más reciente de SDK de cifrado de AWS para C.

> **Note**
>
> Todas las versiones SDK de cifrado de AWS para C anteriores a la 2.0.0 están en end-of-supportfase.
> Puede actualizar de forma segura desde la versión 2.0.x y versiones posteriores a la última versión de SDK de cifrado de AWS para C sin cambios en el código ni en los datos. Sin embargo, nuevas características de seguridad que se introdujeron en la versión 2.0.x no son compatibles con versiones anteriores. Para actualizar desde versiones anteriores a la 1.7.x a la versión 2.0.x y posteriores, primero debe actualizar a la última versión 1.x de SDK de cifrado de AWS para C. Para obtener más información, consulte Migrando su AWS Encryption SDK.

Encontrará instrucciones detalladas para instalarlo y compilarlo SDK de cifrado de AWS para C en el archivo README del aws-encryption-sdk-crepositorio. Incluye instrucciones para crear en las plataformas Amazon Linux, Ubuntu, macOS y Windows.

Antes de empezar, decida si desea utilizar conjuntos de claves de AWS KMS en el AWS Encryption SDK. Si utilizas un AWS KMS llavero, tendrás que instalar el. AWS SDK para C++ Se requiere el AWS SDK para interactuar con AWS Key Management Service(AWS KMS). Cuando utilizas AWS KMS anillos de claves, se AWS Encryption SDK utilizan AWS KMS para generar y proteger las claves de cifrado que protegen tus datos.

No necesita instalarlo AWS SDK para C++ si utiliza otro tipo de anillo de claves, como un anillo de claves AES sin procesar, un anillo de claves RSA sin procesar o un anillo de claves múltiples que no incluye un anillo de claves. AWS KMS Sin embargo, cuando utilice un tipo de conjunto de claves sin procesar, tendrá que generar y proteger sus propias claves de encapsulamiento sin procesar.

Si tiene problemas con la instalación, registre un problema en el repositorio `aws-encryption-sdk-c` o utilice cualquiera de los enlaces de comentarios de esta página.

Uso del SDK de cifrado de AWS para C

En este tema se explican algunas de las funciones del SDK de cifrado de AWS para C que no se admiten en otras implementaciones de lenguajes de programación.

Los ejemplos en esta sección muestran cómo usar la versión 2.0.x y versiones posteriores de SDK de cifrado de AWS para C. Para ver ejemplos que utilizan versiones anteriores, busca tu versión en la lista de versiones del aws-encryption-sdk-c repositorio de GitHub.

Para obtener más información sobre la programación con SDK de cifrado de AWS para C, consulta los ejemplos en C, los ejemplos del aws-encryption-sdk-c repositorio y la documentación de la SDK de cifrado de AWS para C API. GitHub

Véase también: Conjuntos de claves

Temas

- Patrones para cifrar y descifrar datos
- Recuento de referencias

Patrones para cifrar y descifrar datos

Al utilizar el SDK de cifrado de AWS para C, se sigue un patrón similar al siguiente: crear un anillo de claves, crear una CMM que utilice el anillo de claves, crear una sesión que utilice el CMM (y el anillo de claves) y, a continuación, procesar la sesión.

1. Cargue cadenas de error.

   Llame al método `aws_cryptosdk_load_error_strings()` en su código C o C++. Carga información de error que es muy útil para la depuración.

   Solo necesita llamarlo una vez, por ejemplo, en su método `main`.

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

2. Cree un conjunto de claves.

   Configure el conjunto de claves con las claves de encapsulamiento que desee utilizar para cifrar sus claves de datos. En este ejemplo, se utiliza un AWS KMS llavero con otro AWS KMS key, pero se puede utilizar cualquier tipo de llavero en su lugar.

   Para identificar un AWS KMS key en un anillo de claves de cifrado en SDK de cifrado de AWS para C, especifique un ARN de clave o un ARN de alias. En un conjunto de claves de descifrado, debe usar un ARN de clave. Para obtener más información, consulte Identificarse AWS KMS keys en un AWS KMS llavero.

   ```
   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. Crear una sesión.

   En el SDK de cifrado de AWS para C, se utiliza una sesión para cifrar un solo mensaje de texto sin formato o descifrar un solo mensaje de texto cifrado, independientemente de su tamaño. La sesión mantiene el estado del mensaje a lo largo del procesamiento.

   Configure la sesión con un asignador, un conjunto de claves y un modo: `AWS_CRYPTOSDK_ENCRYPT` o `AWS_CRYPTOSDK_DECRYPT`. Si tiene que cambiar el modo de la sesión, utilice el método `aws_cryptosdk_session_reset`.

Al crear una sesión con un llavero, se crea SDK de cifrado de AWS para C automáticamente un gestor de materiales criptográficos (CMM) predeterminado para usted. No es necesario crear, mantener o destruir este objeto.

Por ejemplo, en la sesión siguiente se utiliza el asignador y el conjunto de claves que se ha definido en el paso 1. Cuando se cifran datos, el modo es `AWS_CRYPTOSDK_ENCRYPT`.

```
struct aws_cryptosdk_session * session =
 aws_cryptosdk_session_new_from_keyring_2(allocator, AWS_CRYPTOSDK_ENCRYPT,
 kms_keyring);
```

4. Cifre o descifre los datos.

Para procesar los datos de la sesión, utilice el método `aws_cryptosdk_session_process`. Si el búfer de entrada es lo suficientemente grande como para contener todo el texto no cifrado y el búfer de salida es lo suficientemente grande como para contener todo el texto cifrado, puede llamar a `aws_cryptosdk_session_process_full`. Sin embargo, si tiene que gestionar un streaming de datos, puede llamar a `aws_cryptosdk_session_process` en un bucle. Para ver un ejemplo, consulte file_streaming.cpp. `aws_cryptosdk_session_process_full`Se presenta en AWS Encryption SDK las versiones 1.9. x y 2.2. x.

Cuando la sesión se configura para cifrar los datos, los campos de texto no cifrado describen la entrada y los campos de texto cifrado describen la salida. El campo `plaintext` mantiene el mensaje que desea cifrar y el campo `ciphertext` recibe el mensaje cifrado que devuelve el método de cifrado.

```
/* Encrypting data */
aws_cryptosdk_session_process_full(session,
                                   ciphertext,
                                   ciphertext_buffer_size,
                                   &ciphertext_length,
                                   plaintext,
                                   plaintext_length)
```

Cuando la sesión se configura para descifrar los datos, los campos de texto cifrado describen la entrada y los campos de texto no cifrado describen la salida. El campo `ciphertext` mantiene el mensaje cifrado que ha devuelto el método de cifrado y el campo `plaintext` recibe el mensaje de texto no cifrado que devuelve el método de descifrado.

Para descifrar los datos, llame al método `aws_cryptosdk_session_process_full`.

```
/* Decrypting data */
aws_cryptosdk_session_process_full(session,
                                   plaintext,
                                   plaintext_buffer_size,
                                   &plaintext_length,
                                   ciphertext,
                                   ciphertext_length)
```

Recuento de referencias

Para evitar pérdidas de memoria, libere las referencias a todos los objetos que cree cuando haya terminado con ellos. De lo contrario, acabará con fugas de memoria. El SDK proporciona métodos para facilitar esta tarea.

Cada vez que se crea un objeto principal con uno de los siguientes objetos secundarios, el objeto principal obtiene y mantiene una referencia al objeto secundario, como se indica a continuación:

- Un conjunto de claves, como crear una sesión con un conjunto de claves
- Un administrador de materiales criptográficos (CMM) predeterminado, como crear una sesión o un CMM personalizado con un CMM predeterminado
- Una caché de claves de datos, como crear un CMM de almacenamiento en caché con un conjunto de claves y una caché

A menos que necesite una referencia independiente al objeto secundario, puede liberar la referencia al objeto secundario tan pronto como cree el objeto principal. La referencia restante al objeto secundario se libera cuando se destruye el objeto principal. Este patrón garantiza que mantenga la referencia a cada objeto solo durante el tiempo que la necesite y no tener fugas de memoria debidas a referencias sin liberar.

Solo es responsable de liberar referencias a los objetos secundarios que cree explícitamente. No es responsable de administrar las referencias a ningún objeto que el SDK cree para usted. Si el SDK crea un objeto, como el predeterminado que el método `aws_cryptosdk_caching_cmm_new_from_keyring` agrega a una sesión, el SDK administra la creación y destrucción del objeto y sus referencias.

En el ejemplo siguiente, al crear una sesión con un conjunto de claves, la sesión obtiene una referencia al conjunto de claves y la mantiene hasta que se destruya la sesión. Si no necesita mantener una referencia adicional al conjunto de claves, puede utilizar el método `aws_cryptosdk_keyring_release` para liberar el objeto de conjunto de claves tan pronto como se cree la sesión. Este método reduce el recuento de referencias para el conjunto de claves. La referencia de la sesión al conjunto de claves se libera cuando se llama a `aws_cryptosdk_session_destroy` para destruir la sesión.

```
// 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);
```

En el caso de tareas más complejas, como la reutilización de un conjunto de claves para varias sesiones o la especificación de un conjunto de algoritmos en un CMM, es posible que necesite mantener una referencia independiente al objeto. Si es así, no llame a los métodos de lanzamiento inmediatamente. En su lugar, además de destruir la sesión, libere las referencias cuando ya no use los objetos.

Esta técnica de recuento de referencias también funciona cuando se utiliza una alternativa CMMs, como la CMM de almacenamiento en caché para el almacenamiento en caché de claves de datos. Cuando crea un CMM de almacenamiento en caché a partir de una caché y un conjunto de claves, el CMM de almacenamiento en caché obtiene una referencia a ambos objetos. A menos que los necesite para otra tarea, puede liberar las referencias independientes a la caché y al conjunto de claves tan pronto como se cree el CMM de almacenamiento en caché. Luego, cuando cree una sesión con el CMM de almacenamiento en caché, puede liberar la referencia al CMM de almacenamiento en caché.

Observe que solo es responsable de liberar referencias a objetos que cree explícitamente. Los objetos que los métodos creen en su lugar, como el CMM predeterminado que subyace al CMM de almacenamiento en caché, se administran a través del método.

```
/ 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);
```

SDK de cifrado de AWS para C ejemplos

Los siguientes ejemplos muestran cómo utilizarlos SDK de cifrado de AWS para C para cifrar y descifrar datos.

Los ejemplos en esta sección muestran cómo usar las versiones 2.0.x y posteriores del SDK de cifrado de AWS para C. Para ver ejemplos que utilizan versiones anteriores, busque su versión en la lista de versiones del aws-encryption-sdk-c repositorio de. GitHub

Al instalar y compilar SDK de cifrado de AWS para C, el código fuente de estos y otros ejemplos se incluye en el `examples` subdirectorio y se compila e integra en el `build` directorio. También puedes encontrarlos en el subdirectorio de ejemplos del aws-encryption-sdk-crepositorio en. GitHub

Temas

- Cifrado y descifrado de cadenas

Cifrado y descifrado de cadenas

El siguiente ejemplo muestra cómo utilizar el SDK de cifrado de AWS para C para cifrar y descifrar una cadena.

En este ejemplo se incluye el AWS KMS anillo de claves, un tipo de anillo de claves que utiliza un AWS KMS key in AWS Key Management Service (AWS KMS) para generar y cifrar claves de datos. El ejemplo incluye código escrito en C++. SDK de cifrado de AWS para C Requiere llamar AWS KMS

cuando se utilizan AWS SDK para C++ anillos de claves. AWS KMS Si utilizas un llavero con el que no interactúa AWS KMS, como un llavero AES puro, un llavero RSA puro o un llavero múltiple que no incluye un AWS KMS llavero, no es obligatorio. AWS SDK para C++

Si necesitas ayuda para crear una AWS KMS key, consulta Cómo crear claves en la Guía para desarrolladores.AWS Key Management Service Si necesita ayuda para AWS KMS keys identificarlas en un AWS KMS llavero, consulteIdentificarse AWS KMS keys en un AWS KMS llavero.

Vea la muestra de código completa: string.cpp

Temas

- Cifrado de una cadena
- Descifrado de una cadena

Cifrado de una cadena

En la primera parte de este ejemplo, se utiliza un AWS KMS anillo de claves con otro AWS KMS key para cifrar una cadena de texto sin formato.

Paso 1. Cargue cadenas de error.

Llame al método `aws_cryptosdk_load_error_strings()` en su código C o C++. Carga información de error que es muy útil para la depuración.

Solo necesita llamarlo una vez, por ejemplo, en su método `main`.

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

Paso 2: crear el conjunto de claves.

Cree un anillo de AWS KMS claves para el cifrado. El anillo de claves de este ejemplo está configurado con uno AWS KMS key, pero puede configurar un AWS KMS anillo de claves con varios AWS KMS keys, incluso AWS KMS keys en cuentas diferentes Regiones de AWS y diferentes.

Para identificar un AWS KMS key en un anillo de claves de cifrado en SDK de cifrado de AWS para C, especifique un ARN de clave o un ARN de alias. En un conjunto de claves de descifrado, debe usar un ARN de clave. Para obtener más información, consulte Identificarse AWS KMS keys en un AWS KMS llavero.

Identificarse AWS KMS keys en un AWS KMS llavero

Al crear un conjunto de claves con varios AWS KMS keys, se especifica la que AWS KMS key se utiliza para generar y cifrar la clave de datos de texto sin formato y una matriz opcional de claves adicionales AWS KMS keys que cifran la misma clave de datos de texto sin formato. En este caso, solo se especifica el generador. AWS KMS key

Antes de ejecutar este código, reemplace el ARN de clave de ejemplo por uno válido.

```
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);
```

Paso 3: crear una sesión.

Cree una sesión. Para ello, use el asignador, un enumerador de modos y el conjunto de claves.

Cada sesión requiere un modo: bien `AWS_CRYPTOSDK_ENCRYPT` para cifrar o `AWS_CRYPTOSDK_DECRYPT` para descifrar. Para cambiar el modo de una sesión existente, utilice el método `aws_cryptosdk_session_reset`.

Después de crear una sesión con el conjunto de claves, puede liberar la referencia al conjunto de claves con el método que el SDK proporciona. La sesión conserva una referencia al objeto conjunto de claves durante su vida útil. Las referencias al conjunto de claves y a los objetos de sesión se liberan cuando se destruye la sesión. Esta técnica de recuento de referencias ayuda a evitar fugas de memoria y que los objetos se liberen mientras están en uso.

```
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);
```

Paso 4: establecer el contexto de cifrado.

Un contexto de cifrado son datos autenticados adicionales que son arbitrarios y no son secretos. Al proporcionar un contexto de cifrado al cifrar, este vincula AWS Encryption SDK

criptográficamente el contexto de cifrado al texto cifrado, de modo que se requiere el mismo contexto de cifrado para descifrar los datos. El uso de un contexto de cifrado es opcional, pero es una práctica recomendada que le aconsejamos.

En primer lugar, cree una tabla hash que incluya las cadenas del contexto de cifrado.

```
/* 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)
```

Obtenga un puntero mutable al contexto de cifrado en la sesión. A continuación, utilice la función `aws_cryptosdk_enc_ctx_clone` para copiar el contexto de cifrado en la sesión. Conserve la copia en `my_enc_ctx` para poder validar el valor después de descifrar los datos.

El contexto de cifrado forma parte de la sesión, no es un parámetro transferido a la función de proceso de la sesión. Esto garantiza que se utilice el mismo contexto de cifrado para cada segmento de un mensaje, incluso si se llama varias veces a la función de proceso de la sesión para cifrar todo el mensaje.

```
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)
```

Paso 5: cifrar la cadena.

Para cifrar la cadena de texto no cifrado, utilice el método `aws_cryptosdk_session_process_full` con la sesión en modo de cifrado. Este método, introducido en las versiones 1.9. AWS Encryption SDK x y 2.2. x, está diseñado para el cifrado y el descifrado sin transmisión. Para gestionar el streaming de datos, utilice el comando `aws_cryptosdk_session_process` en un bucle.

A la hora de cifrar, los campos de texto no cifrado son campos de entrada; los campos de texto no cifrado son campos de salida. Cuando se completa el procesamiento, el campo `ciphertext_output` contiene el mensaje cifrado, incluido el texto no cifrado real, las claves de datos cifrados y el contexto de cifrado. Puede descifrar este mensaje cifrado mediante cualquier lenguaje AWS Encryption SDK de programación compatible.

```
/* 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;
}
```

Paso 6: limpiar la sesión.

El paso final destruye la sesión, incluidas las referencias al CMM y al conjunto de claves.

Si lo prefiere, en lugar de destruir la sesión, puede reutilizarla con el mismo conjunto de claves y CMM para descifrar la cadena o para cifrar o descifrar otros mensajes. Para utilizar la sesión para descifrado, utilice el método `aws_cryptosdk_session_reset` para cambiar el modo a `AWS_CRYPTOSDK_DECRYPT`.

Descifrado de una cadena

La segunda parte de este ejemplo descifra un mensaje cifrado que contiene el texto cifrado de la cadena original.

Paso 1: cargar las cadenas de error.

Llame al método `aws_cryptosdk_load_error_strings()` en su código C o C++. Carga información de error que es muy útil para la depuración.

Solo necesita llamarlo una vez, por ejemplo, en su método `main`.

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

Paso 2: crear el conjunto de claves.

Al descifrar los datos AWS KMS, se pasa el mensaje cifrado que ha devuelto la API de cifrado. La API de descifrado no toma ninguna entrada como AWS KMS key entrada. En su lugar, AWS KMS utiliza lo mismo AWS KMS key para descifrar el texto cifrado que utilizó para cifrarlo. Sin embargo, AWS Encryption SDK permite especificar un conjunto de AWS KMS claves sin cifrar ni descifrar. AWS KMS keys

Al descifrar, puede configurar un anillo de claves solo con el AWS KMS keys que desee utilizar para descifrar el mensaje cifrado. Por ejemplo, es posible que desee crear un conjunto de claves que contenga únicamente el AWS KMS key que utilice un rol concreto de su organización. Nunca AWS Encryption SDK utilizará un AWS KMS key a menos que aparezca en el anillo de claves de descifrado. Si el SDK no puede descifrar las claves de datos cifradas con las del AWS KMS keys anillo de claves que usted proporcione, ya sea porque ninguna de las claves del conjunto de AWS KMS keys claves se utilizó para cifrar ninguna de las claves de datos o porque la persona que llama no tiene permiso para usar las claves del anillo de claves para descifrar, la AWS KMS keys llamada de descifrado fallará.

Al especificar un anillo AWS KMS key de claves de descifrado, debe utilizar su clave ARN. ARNsLos alias solo están permitidos en los anillos de claves de cifrado. Si necesita ayuda para identificarlos AWS KMS keys en un AWS KMS anillo de claves, consulte. Identificarse AWS KMS keys en un AWS KMS llavero

En este ejemplo, especificamos un conjunto de claves configurado con el mismo que AWS KMS key se utiliza para cifrar la cadena. Antes de ejecutar este código, reemplace el ARN de clave de ejemplo por uno válido.

```
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);
```

Paso 3: crear una sesión.

Cree una sesión con el asignador y el conjunto de claves. Para configurar la sesión para descifrado, configure la sesión con el modo `AWS_CRYPTOSDK_DECRYPT`.

Después de crear una sesión con un conjunto de claves, puede liberar la referencia al conjunto de claves con el método que el SDK proporciona. La sesión conserva una referencia al objeto de

conjunto de claves durante su ciclo de vida y tanto la sesión como el conjunto de claves se liberan cuando se destruye la sesión. Esta técnica de recuento de referencias ayuda a evitar fugas de memoria y que los objetos se liberen mientras están en uso.

```
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);
```

Paso 4: descifrar la cadena.

Para descifrar la cadena, utilice el método `aws_cryptosdk_session_process_full` con la sesión que está configurada para descifrado. Este método introducido en las versiones 1.9.x y 2.2.x de AWS Encryption SDK está diseñado para el cifrado y el descifrado sin streaming. Para gestionar el streaming de datos, utilice el comando `aws_cryptosdk_session_process` en un bucle.

Al descifrar, los campos de texto cifrado son campos de entrada y los campos de texto no cifrado son campos de salida. El campo `ciphertext_input` mantiene el mensaje cifrado que devolvió el método de cifrado. Cuando el procesamiento está completo, el campo `plaintext_output` contiene la cadena de texto no cifrado (descifrado).

```
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;
}
```

Paso 5: verificar el contexto de cifrado.

Asegúrese de que el contexto de cifrado real (el que se utilizó para descifrar el mensaje) contenga el contexto de cifrado que proporcionó al cifrar el mensaje. El contexto de cifrado real

podría incluir pares adicionales, ya que el administrador de materiales criptográficos (CMM) puede añadir pares al contexto de cifrado proporcionado antes de cifrar el mensaje.

En el SDK de cifrado de AWS para C, no es necesario proporcionar un contexto de cifrado al descifrar, ya que el contexto de cifrado está incluido en el mensaje cifrado que devuelve el SDK. Pero, antes de que devuelva el mensaje de texto no cifrado, la función de descifrado debería verificar que todas las parejas en el contexto de cifrado proporcionado aparezcan en el contexto de cifrado que se utilizó para descifrar el mensaje.

En primer lugar, obtenga un puntero de solo lectura a la tabla hash de la sesión. Esta tabla hash contiene el contexto de cifrado que se utilizó para descifrar el mensaje.

```
const struct aws_hash_table *session_enc_ctx =
 aws_cryptosdk_session_get_enc_ctx_ptr(session);
```

A continuación, ejecute un bucle en el contexto de cifrado en la tabla hash `my_enc_ctx` que copió al realizar el cifrado. Verifique que cada para de la tabla hash `my_enc_ctx` que se utilizó para cifrar aparece en la tabla hash `session_enc_ctx` que se utilizó para descifrar. Si falta alguna clave o dicha clave tiene un valor distinto, detenga el procesamiento y escriba un mensaje de error.

```
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();
    }
}
```

Paso 6: limpiar la sesión.

Después de verificar el contexto de cifrado, puede destruir la sesión o reutilizarla. Si tiene que reconfigurar la sesión, utilice el método `aws_cryptosdk_session_reset`.

```
aws_cryptosdk_session_destroy(session);
```

AWS Encryption SDK para.NET

La AWS Encryption SDK para .NET es una biblioteca de cifrado del lado del cliente para desarrolladores que escriben aplicaciones en C# y otros lenguajes de programación.NET. Es compatible con Windows, macOS y Linux.

> **Note**
>
> La versión 4.0.0 de para.NET se aparta AWS Encryption SDK de la especificación de mensajes. AWS Encryption SDK En consecuencia, los mensajes cifrados con la versión 4.0.0 solo se pueden descifrar con la versión 4.0.0 o posterior de la versión para .NET. AWS Encryption SDK No se pueden descifrar mediante ninguna otra implementación de lenguaje de programación.
> La versión 4.0.1 de .NET escribe AWS Encryption SDK los mensajes de acuerdo con la AWS Encryption SDK especificación de mensajes y es interoperable con otras implementaciones de lenguajes de programación. De forma predeterminada, la versión 4.0.1 puede leer los mensajes cifrados por la versión 4.0.0. Sin embargo, si no desea descifrar los mensajes cifrados con la versión 4.0.0, puede especificar la propiedad de la `NetV4_0_0_RetryPolicy` para impedir que el cliente lea estos mensajes. Para obtener más información, consulte las notas de la versión 4.0.1 en el repositorio de. aws-encryption-sdk GitHub

La AWS Encryption SDK versión para .NET se diferencia de algunas de las implementaciones de otros lenguajes de programación AWS Encryption SDK en los siguientes aspectos:

- No se admite el almacenamiento en caché de claves de datos

  > **Note**
  >
  > Versión 4. x de la versión AWS Encryption SDK para .NET es compatible con el anillo de claves AWS KMS jerárquico, una solución alternativa de almacenamiento en caché de materiales criptográficos.

- No es compatible con el streaming de datos

- Sin registros ni seguimientos de pilas del AWS Encryption SDK para .NET
- Requiere el AWS SDK para .NET

El AWS Encryption SDK para .NET incluye todas las funciones de seguridad introducidas en las versiones 2.0. x y versiones posteriores de las implementaciones en otros idiomas de AWS Encryption SDK. Sin embargo, si utiliza para.NET AWS Encryption SDK para descifrar datos cifrados con una versión anterior a la versión 2.0. x versión de la implementación en otro idioma de la AWS Encryption SDK, es posible que necesite ajustar su política de compromisos. Para obtener más información, consulte ¿Cómo establecer su política de compromiso?.

La AWS Encryption SDK versión para .NET es un producto de AWS Encryption SDK Dafny, un lenguaje de verificación formal en el que se escriben las especificaciones, el código para implementarlas y las pruebas para probarlas. El resultado es una biblioteca que implementa las características del AWS Encryption SDK en una trama que garantiza la corrección funcional.

Más información

- Para ver ejemplos que muestran cómo configurar las opciones del AWS Encryption SDK, como especificar un conjunto de algoritmos alternativo, limitar las claves de datos cifrados y utilizar claves AWS KMS multirregionales, consulte. Configuración del AWS Encryption SDK
- Para obtener más información sobre la AWS Encryption SDK programación con para.NET, consulte el `aws-encryption-sdk-net`directorio del aws-encryption-sdk repositorio en GitHub.

Temas

- Instalación del AWS Encryption SDK para .NET
- Depurar el AWS Encryption SDK para .NET
- AWS Encryption SDK para ver ejemplos de.NET

Instalación del AWS Encryption SDK para .NET

El AWS Encryption SDK para .NET está disponible como `AWS.Cryptography.EncryptionSDK`paquete en NuGet. Para obtener más información sobre la instalación y la creación de AWS Encryption SDK para .NET, consulte el archivo README.md del repositorio. `aws-encryption-sdk-net`

Versión 3.x

Versión 3. x de la AWS Encryption SDK para.NET solo es compatible con .NET Framework 4.5.2 — 4.8 en Windows. Es compatible con .NET Core 3.0+ y .NET 5.0 y versiones posteriores en todos los sistemas operativos compatibles.

Versión 4.x

Versión 4. x de la AWS Encryption SDK para.NET es compatible con .NET 6.0 y .NET Framework net48 y versiones posteriores. Versión 4. x requiere el AWS SDK para .NET v3.

El AWS Encryption SDK sistema para .NET requiere las teclas « SDK para .NET incluso si no se utilizan» AWS Key Management Service (AWS KMS). Se instala con el NuGet paquete. Sin embargo, a menos que utilice AWS KMS claves, AWS Encryption SDK para .NET no se Cuenta de AWS requieren AWS credenciales ni interacción con ningún AWS servicio. Si necesita ayuda para configurar una AWS cuenta, consulteUso del AWS Encryption SDK con AWS KMS.

Depurar el AWS Encryption SDK para .NET

El AWS Encryption SDK para .NET no genera ningún registro. Las excepciones AWS Encryption SDK para .NET generan un mensaje de excepción, pero no hay rastros de pila.

Para ayudarle a depurar, asegúrese de activar el inicio de sesión en la SDK para .NET. Los registros y los mensajes de error del SDK para .NET pueden ayudarle a distinguir los errores que se producen en. NET SDK para .NET de los que se AWS Encryption SDK producen en él. Para obtener ayuda con el SDK para .NET registro, consulte AWSLoggingla Guía para AWS SDK para .NET desarrolladores. (Para ver el tema, amplíe la sección Abrir para ver la sección de contenido de .NET Framework).

AWS Encryption SDK para ver ejemplos de.NET

Los ejemplos siguientes muestran los patrones de codificación básicos que se utilizan al programar con AWS Encryption SDK para.NET. En concreto, se crea una instancia de la biblioteca AWS Encryption SDK y de proveedores de materiales. A continuación, antes de llamar a cada método, se crea una instancia de un objeto que define la entrada del método. Es muy parecido al patrón de codificación que utiliza en la SDK para .NET.

Para ver ejemplos que muestran cómo configurar las opciones de la AWS Encryption SDK, como especificar un conjunto de algoritmos alternativo, limitar las claves de datos cifrados y utilizar claves AWS KMS multirregionales, consulte. Configuración del AWS Encryption SDK

Para ver más ejemplos de programación con AWS Encryption SDK para.NET, consulte los ejemplos en el `aws-encryption-sdk-net` directorio del `aws-encryption-sdk` repositorio en GitHub.

Cifrar datos en el AWS Encryption SDK para .NET

En este ejemplo se muestra el patrón básico de cifrado de datos. Cifra un archivo pequeño con claves de datos que están protegidas por una clave de AWS KMS empaquetado.

Paso 1: Cree una instancia de la biblioteca AWS Encryption SDK y de los proveedores de materiales.

Comience por crear una instancia de la biblioteca AWS Encryption SDK y de los proveedores de materiales. Utilizará los métodos descritos en AWS Encryption SDK para cifrar y descifrar los datos. Utilizará los métodos de la biblioteca de proveedores de materiales para crear los conjuntos de claves que especifican qué claves protegen sus datos.

La forma de instanciar la biblioteca AWS Encryption SDK y la biblioteca de proveedores de materiales difiere entre las versiones 3. x y 4. x de AWS Encryption SDK para .NET. Todos los pasos siguientes son los mismos para ambas versiones 3. x y 4. x de AWS Encryption SDK para .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());
```

Paso 2: crear un objeto de entrada para el conjunto de claves.

Cada método que crea un conjunto de claves tiene una clase de objeto de entrada correspondiente. Por ejemplo, para crear el objeto de entrada para el método `CreateAwsKmsKeyring()`, cree una instancia de la clase `CreateAwsKmsKeyringInput`.

Aunque la entrada de este conjunto de claves no especifica una clave generadora, la única clave KMS especificada por el parámetro `KmsKeyId` es la clave generadora. Genera y cifra la clave de datos que cifra los datos.

Este objeto de entrada requiere un AWS KMS cliente para Región de AWS la clave KMS. Para crear un AWS KMS cliente, cree una instancia de la `AmazonKeyManagementServiceClient` clase en. SDK para .NET Llamar al constructor de `AmazonKeyManagementServiceClient()` sin parámetros crea un cliente con los valores predeterminados.

En un AWS KMS conjunto de claves utilizado para cifrar con .NET, puede identificar las claves de KMS mediante el ID de clave, el ARN de clave, el nombre de alias o el alias ARN. AWS Encryption SDK En un conjunto de AWS KMS claves utilizado para descifrar, debe usar un ARN de clave para identificar cada clave de KMS. Si piensa reutilizar su conjunto de claves de cifrado para descifrar, utilice un identificador ARN de clave para todas las claves de KMS.

```
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
};
```

Paso 3: crear el conjunto de claves.

Para crear el conjunto de claves, llame al método de conjunto de claves con el objeto de entrada del conjunto de claves. En este ejemplo, se utiliza el método `CreateAwsKmsKeyring()`, que solo necesita una clave de KMS.

```
var keyring = materialProviders.CreateAwsKmsKeyring(kmsKeyringInput);
```

Paso 4: definir un contexto de cifrado.

Un contexto de cifrado es un elemento opcional, pero muy recomendable, de las operaciones criptográficas del. AWS Encryption SDK Puede definir uno o varios pares clave-valor no secretos.

> **Note**
>
> Con la versión 4. x AWS Encryption SDK para .NET, puede requerir un contexto de cifrado en todas las solicitudes de cifrado con el contexto de cifrado requerido CMM.

```
// Define the encryption context
var encryptionContext = new Dictionary<string, string>()
{
    {"purpose", "test"}
};
```

Paso 5: crear el objeto de entrada para cifrar.

Antes de llamar al método `Encrypt()`, cree una instancia de la clase `EncryptInput`.

```
string plaintext = File.ReadAllText("C:\\Documents\\CryptoTest\\TestFile.txt");

// Define the encrypt input
var encryptInput = new EncryptInput
{
    Plaintext = plaintext,
    Keyring = keyring,
    EncryptionContext = encryptionContext
};
```

Paso 6: cifrar el texto no cifrado.

Utilice el `Encrypt()` método de AWS Encryption SDK para cifrar el texto sin formato mediante el conjunto de claves que haya definido.

La `EncryptOutput` que devuelve el método `Encrypt()` contiene métodos para obtener el mensaje cifrado (`Ciphertext`), el contexto de cifrado y el conjunto de algoritmos.

```
var encryptOutput = encryptionSdk.Encrypt(encryptInput);
```

Paso 7: obtener el mensaje cifrado.

El `Decrypt()` método AWS Encryption SDK para .NET toma el `Ciphertext` miembro de la instancia. `EncryptOutput`

El miembro `Ciphertext` del objeto `EncryptOutput` es el mensaje cifrado, un objeto portátil que incluye los datos cifrados, las claves de datos cifrados y los metadatos, incluido el contexto de cifrado. Puede almacenar de forma segura el mensaje cifrado durante un período prolongado o enviarlo al método de `Decrypt()` para recuperar el texto no cifrado.

```
var encryptedMessage = encryptOutput.Ciphertext;
```

Descifrar en modo estricto en AWS Encryption SDK para.NET

Las prácticas recomendadas recomiendan especificar las claves que se utilizan para descifrar los datos, una opción conocida como modo estricto. AWS Encryption SDK Utiliza únicamente las claves KMS que especifique en su conjunto de claves para descifrar el texto cifrado. Las claves del conjunto de claves de descifrado deben incluir al menos una de las claves que cifraron los datos.

En este ejemplo se muestra el patrón básico de descifrado en modo estricto con la AWS Encryption SDK para .NET.

Paso 1: Cree una instancia de la biblioteca y de los proveedores de materiales. AWS Encryption SDK

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

Paso 2: crear un objeto de entrada para el conjunto de claves.

Para especificar los parámetros del método de conjunto de claves, cree un objeto de entrada. Cada método de conjunto de claves de .NET tiene un objeto de entrada correspondiente. AWS Encryption SDK Como en este ejemplo se utiliza el método `CreateAwsKmsKeyring()` para crear el conjunto de claves, se crea una instancia de la clase `CreateAwsKmsKeyringInput` para la entrada.

En un conjunto de claves de descifrado, debe usar el ARN de una clave para identificar claves KMS.

```
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
};
```

Paso 3: crear el conjunto de claves.

Para crear el conjunto de claves de descifrado, en este ejemplo se utiliza el método `CreateAwsKmsKeyring()` y el objeto de entrada del conjunto de claves.

```
var keyring = materialProviders.CreateAwsKmsKeyring(kmsKeyringInput);
```

Paso 4: crear el objeto de entrada para descifrar.

Para crear el objeto de entrada para el método `Decrypt()`, cree una instancia de la clase `DecryptInput`.

El parámetro `Ciphertext` del constructor `DecryptInput()` toma el miembro `Ciphertext` del objeto `EncryptOutput` que devolvió el método `Encrypt()`. La propiedad `Ciphertext` representa el mensaje cifrado, que incluye los datos cifrados, las claves de datos cifrados y los metadatos que la AWS Encryption SDK necesita para descifrar el mensaje.

Con la versión 4. x del AWS Encryption SDK para.NET, puede utilizar el `EncryptionContext` parámetro opcional para especificar el contexto de cifrado en el `Decrypt()` método.

Utilice el parámetro `EncryptionContext` para comprobar que el contexto de cifrado utilizado para cifrar está incluido en el contexto de cifrado utilizado para descifrar el texto cifrado. AWS Encryption SDK Añade pares al contexto de cifrado, incluida la firma digital si utilizas un conjunto de algoritmos con firma, como el conjunto de algoritmos predeterminado.

```
var encryptedMessage = encryptOutput.Ciphertext;

var decryptInput = new DecryptInput
{
    Ciphertext = encryptedMessage,
    Keyring = keyring,
    EncryptionContext = encryptionContext // OPTIONAL
};
```

Paso 5: descifrar el texto cifrado.

```
var decryptOutput = encryptionSdk.Decrypt(decryptInput);
```

Paso 6: verificar el contexto de cifrado; versión 3.x

El `Decrypt()` método de la versión 3. x del AWS Encryption SDK para .NET no utiliza un contexto de cifrado. Obtiene los valores del contexto de cifrado de los metadatos en el mensaje cifrado. Sin embargo, antes de devolver o utilizar el texto no cifrado, se recomienda comprobar que el contexto de cifrado que se utilizó para descifrar el texto cifrado incluye el contexto de cifrado que proporcionó al cifrar.

Compruebe que el contexto de cifrado utilizado para cifrar esté incluido en el contexto de cifrado que se utilizó para descifrar el texto cifrado. AWS Encryption SDK Agrega pares al contexto de cifrado, incluida la firma digital si se utiliza un conjunto de algoritmos con firma, como el conjunto de algoritmos predeterminado.

```
// 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");
}
```

Descifrar con un anillo de claves de detección en el para.NET AWS Encryption SDK

En lugar de especificar las claves de KMS para el descifrado, puede proporcionar un conjunto de claves de detección de AWS KMS , que es un conjunto de claves que no especifica ninguna clave de KMS. Un anillo de claves de detección permite AWS Encryption SDK descifrar los datos utilizando la clave KMS que los haya cifrado, siempre que la persona que llama tenga permiso para descifrar la clave. Como práctica recomendada, añada un filtro de detección que limite las claves de KMS que se pueden usar a las Cuentas de AWS de una partición específica, en particular.

AWS Encryption SDK Para .NET, se proporciona un conjunto de claves de detección básico que requiere un AWS KMS cliente y un conjunto de claves de detección múltiple que requiere la especificación de uno o más. Regiones de AWS Tanto el cliente como las regiones limitan las claves KMS que se pueden usar para descifrar el mensaje cifrado. Los objetos de entrada de ambos conjuntos de claves utilizan el filtro de detección recomendado.

El siguiente ejemplo muestra el patrón de descifrado de datos con un conjunto de claves de detección de AWS KMS y un filtro de detección.

Paso 1: crear una instancia de la biblioteca AWS Encryption SDK y de la biblioteca de proveedores de materiales.

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

Paso 2: crear el objeto de entrada para el conjunto de claves.

Para especificar los parámetros del método de conjunto de claves, cree un objeto de entrada. Cada método de conjunto de claves de .NET tiene un objeto de entrada correspondiente. AWS Encryption SDK Como en este ejemplo se utiliza el método `CreateAwsKmsDiscoveryKeyring()` para crear el conjunto de claves, se crea una instancia de la clase `CreateAwsKmsDiscoveryKeyringInput` para la entrada.

```
List<string> accounts = new List<string> { "111122223333" };

var discoveryKeyringInput = new CreateAwsKmsDiscoveryKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    DiscoveryFilter = new DiscoveryFilter()
    {
        AccountIds = accounts,
        Partition = "aws"
    }
};
```

Paso 3: crear el conjunto de claves.

Para crear el conjunto de claves de descifrado, en este ejemplo se utiliza el método `CreateAwsKmsDiscoveryKeyring()` y el objeto de entrada del conjunto de claves.

```
var discoveryKeyring =
 materialProviders.CreateAwsKmsDiscoveryKeyring(discoveryKeyringInput);
```

Paso 4: crear el objeto de entrada para descifrar.

Para crear el objeto de entrada para el método `Decrypt()`, cree una instancia de la clase `DecryptInput`. El valor del parámetro `Ciphertext` es el miembro `Ciphertext` del objeto `EncryptOutput` que devuelve el método `Encrypt()`.

Con la versión 4. x del AWS Encryption SDK para.NET, puede utilizar el `EncryptionContext` parámetro opcional para especificar el contexto de cifrado en el `Decrypt()` método.

Utilice el parámetro `EncryptionContext` para comprobar que el contexto de cifrado utilizado para cifrar está incluido en el contexto de cifrado utilizado para descifrar el texto cifrado. AWS Encryption SDK Añade pares al contexto de cifrado, incluida la firma digital si utilizas un conjunto de algoritmos con firma, como el conjunto de algoritmos predeterminado.

```
var ciphertext = encryptOutput.Ciphertext;

var decryptInput = new DecryptInput
{
    Ciphertext = ciphertext,
    Keyring = discoveryKeyring,
    EncryptionContext = encryptionContext // OPTIONAL

};
var decryptOutput = encryptionSdk.Decrypt(decryptInput);
```

Paso 5: verificar el contexto de cifrado; versión 3.x

El `Decrypt()` método de la versión 3. x del AWS Encryption SDK para .NET no incluye un contexto de cifrado`Decrypt()`. Obtiene los valores del contexto de cifrado de los metadatos en el mensaje cifrado. Sin embargo, antes de devolver o utilizar el texto no cifrado, se recomienda comprobar que el contexto de cifrado que se utilizó para descifrar el texto cifrado incluye el contexto de cifrado que proporcionó al cifrar.

Compruebe que el contexto de cifrado utilizado para cifrar esté incluido en el contexto de cifrado que se utilizó para descifrar el texto cifrado. AWS Encryption SDK Añade pares al contexto de cifrado, incluida la firma digital si se utiliza un conjunto de algoritmos con firma, como el conjunto de algoritmos predeterminado.

```
// 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 Encryption SDK para Go

En este tema se explica cómo instalar y usar AWS Encryption SDK for Go. Para obtener más información sobre la programación con AWS Encryption SDK for Go, consulte el directorio go del aws-encryption-sdk repositorio en GitHub.

AWS Encryption SDK for Go se diferencia de algunas de las implementaciones de otros lenguajes de programación AWS Encryption SDK en los siguientes aspectos:

- No hay soporte para el almacenamiento en caché de claves de datos. Sin embargo, AWS Encryption SDK for Go es compatible con el llavero AWS KMS jerárquico, una solución alternativa de almacenamiento en caché de materiales criptográficos.
- No es compatible con el streaming de datos

El AWS Encryption SDK for Go incluye todas las funciones de seguridad introducidas en las versiones 2.0. x y versiones posteriores de las implementaciones en otros idiomas del AWS Encryption SDK. Sin embargo, si utiliza AWS Encryption SDK for Go para descifrar datos cifrados con una versión anterior a la versión 2.0. x versión de la implementación en otro idioma de la AWS Encryption SDK, es posible que necesite ajustar su política de compromisos. Para obtener más información, consulte ¿Cómo establecer su política de compromiso?.

El AWS Encryption SDK for Go es un producto de Dafny, un lenguaje de verificación formal en el que se escriben las especificaciones, el código para implementarlas y las pruebas para probarlas. AWS Encryption SDK El resultado es una biblioteca que implementa las características del AWS Encryption SDK en una trama que garantiza la corrección funcional.

Más información

- Para ver ejemplos que muestran cómo configurar las opciones del AWS Encryption SDK, como especificar un conjunto de algoritmos alternativo, limitar las claves de datos cifrados y utilizar claves AWS KMS multirregionales, consulte. Configuración del AWS Encryption SDK
- Para ver ejemplos que muestran cómo configurar y usar AWS Encryption SDK for Go, consulta los ejemplos de Go en el aws-encryption-sdk repositorio correspondiente. GitHub

Temas

- Requisitos previos
- Instalación

Requisitos previos

Antes de instalar AWS Encryption SDK for Go, asegúrese de cumplir los siguientes requisitos previos.

Una versión compatible de Go

AWS Encryption SDK Para Go es necesaria la versión 1.23 o posterior.

Para obtener más información sobre la descarga e instalación de Go, consulta Instalación de Go.

Instalación

Instala la última versión de AWS Encryption SDK for Go. Para obtener más información sobre la instalación y creación de AWS Encryption SDK for Go, consulta el archivo README.md en el directorio go del aws-encryption-sdk repositorio en cuestión. GitHub

Para instalar la versión más reciente

- Instala el for Go AWS Encryption SDK

```
go get github.com/aws/aws-encryption-sdk/releases/go/encryption-sdk@latest
```

- Instale la biblioteca de proveedores de material criptográfico (MPL)

```
go get github.com/aws/aws-cryptographic-material-providers-library/releases/go/mpl
```

SDK de cifrado de AWS para Java

En este tema se explica cómo instalar y utilizar el SDK de cifrado de AWS para Java. Para obtener detalles sobre la programación con SDK de cifrado de AWS para Java, consulte el aws-encryption-sdk-javarepositorio en GitHub. Para obtener documentación de la API, consulte la documentación de Javadoc para SDK de cifrado de AWS para Java.

Temas

- Requisitos previos
- Instalación
- SDK de cifrado de AWS para Java ejemplos

Requisitos previos

Antes de instalarlo SDK de cifrado de AWS para Java, asegúrese de cumplir los siguientes requisitos previos.

Un entorno de desarrollo de Java

Necesitará Java 8 o una versión posterior. En el sitio web de Oracle, vaya a la página de descargas de Java SE y, a continuación, descargue e instale el Java SE Development Kit (JDK).

Si utiliza el JDK de Oracle, también debe descargar e instalar los archivos de políticas de jurisdicción de seguridad ilimitada de la extensión de criptografía de Java (JCE).

Bouncy Castle

SDK de cifrado de AWS para Java Requiere Bouncy Castle.

- SDK de cifrado de AWS para Java las versiones 1.6.1 y posteriores utilizan Bouncy Castle para serializar y deserializar objetos criptográficos. Puede utilizar Bouncy Castle o Bouncy Castle FIPS para cumplir este requisito. Si necesita ayuda para instalar y configurar el FIPS de Bouncy Castle, consulte la documentación del FIPS de BC, especialmente las guías de usuario y la política de seguridad. PDFs
- Las versiones anteriores SDK de cifrado de AWS para Java utilizan la API de criptografía de Bouncy Castle para Java. Este requisito solo se cumple con el Bouncy Castle que no es de FIPS.

Si no tienes Bouncy Castle, ve a Descargar Bouncy Castle para Java para descargar el archivo del proveedor correspondiente a tu JDK. También puedes usar Apache Maven para obtener el

artefacto para el proveedor estándar de Bouncy Castle (bcprov-ext-jdk15on) o el artefacto para Bouncy Castle FIPS (bc-fips).

AWS SDK para Java

Versión 3. x de los SDK de cifrado de AWS para Java requiere el AWS SDK for Java 2.x, incluso si no usas AWS KMS llaveros.

Versión 2. x o una versión anterior de SDK de cifrado de AWS para Java no requiere la AWS SDK para Java. Sin embargo, AWS SDK para Java es necesario para usar AWS Key Management Service(AWS KMS) como proveedor de claves maestras. A partir de la SDK de cifrado de AWS para Java versión 2.4.0, SDK de cifrado de AWS para Java es compatible con las versiones 1.x y 2.x de. AWS SDK para Java AWS Encryption SDK el código de la AWS SDK para Java 1.x y la 2.x es interoperable. Por ejemplo, puede cifrar los datos con un AWS Encryption SDK código compatible con la versión AWS SDK para Java 1.x y descifrarlos con un código compatible AWS SDK for Java 2.x (o viceversa). Las versiones anteriores a la 2.4.0 SDK de cifrado de AWS para Java solo admiten la versión 1.x. AWS SDK para Java Para obtener información sobre cómo actualizar su versión de AWS Encryption SDK, consulte. Migración de su AWS Encryption SDK

Al actualizar el SDK de cifrado de AWS para Java código de la versión AWS SDK para Java 1.x a AWS SDK for Java 2.x, sustituya las referencias a la `AWSKMSinterfaz` de la versión AWS SDK para Java 1.x por las referencias a la `KmsClientinterfaz` de. AWS SDK for Java 2.xNo SDK de cifrado de AWS para Java es compatible con la `KmsAsyncClient` interfaz. Además, actualice el código para usar los objetos relacionados al AWS KMS en el espacio de nombres de `kmssdkv2`, en lugar del espacio de nombres de `kms`.

Para instalar AWS SDK para Java, utilice Apache Maven.

- Para importar todo el AWS SDK para Java como una dependencia declárelo en el archivo `pom.xml`.
- Para crear una dependencia solo para el AWS KMS módulo de la AWS SDK para Java versión 1.x, siga las instrucciones para especificar determinados módulos y establezca el `artifactId` valor. `aws-java-sdk-kms`
- Para crear una dependencia solo para el AWS KMS módulo de la versión AWS SDK para Java 2.x, siga las instrucciones para especificar módulos concretos. Establece el `groupId` en `software.amazon.awssdk` y el `artifactId` en `kms`.

Para ver más cambios, consulta las diferencias entre la versión AWS SDK para Java 1.x y la 2.x en la AWS SDK for Java 2.x Guía para desarrolladores.

Los ejemplos de Java de la Guía AWS Encryption SDK para desarrolladores utilizan el. AWS SDK for Java 2.x

Instalación

Instale la versión más reciente de SDK de cifrado de AWS para Java.

> **Note**
>
> Todas las versiones SDK de cifrado de AWS para Java anteriores a la 2.0.0 se encuentran en esta end-of-supportfase.
> Puede actualizar de forma segura desde la versión 2.0.x y versiones posteriores a la última versión de SDK de cifrado de AWS para Java sin cambios en el código ni en los datos. Sin embargo, nuevas características de seguridad que se introdujeron en la versión 2.0.x no son compatibles con versiones anteriores. Para actualizar desde versiones anteriores a la 1.7.x a la versión 2.0.x y posteriores, primero debe actualizar a la última versión 1.x de AWS Encryption SDK. Para obtener más información, consulte Migrando su AWS Encryption SDK.

Puede instalarlo de SDK de cifrado de AWS para Java las siguientes maneras.

Manualmente

Para instalarlo SDK de cifrado de AWS para Java, clone o descargue el aws-encryption-sdk-java GitHubrepositorio.

Con Apache Maven

SDK de cifrado de AWS para Java Está disponible a través de Apache Maven con la siguiente definición de dependencia.

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-encryption-sdk-java</artifactId>
  <version>3.0.0</version>
</dependency>
```

Después de instalar el SDK, comience consultando el ejemplo de código Java de esta guía y el Javadoc que aparece en él. GitHub

SDK de cifrado de AWS para Java ejemplos

Los siguientes ejemplos muestran cómo utilizarlos SDK de cifrado de AWS para Java para cifrar y descifrar datos. Estos ejemplos muestran cómo utilizar la versión 3. x y versiones posteriores de SDK de cifrado de AWS para Java. Versión 3. x de los SDK de cifrado de AWS para Java requiere el AWS SDK for Java 2.x. Versión 3. x de SDK de cifrado de AWS para Java reemplaza a los proveedores de llaves maestras por llaveros. Para ver ejemplos que utilizan versiones anteriores, busca tu versión en la lista de versiones del aws-encryption-sdk-javarepositorio de GitHub.

Temas

- Cifrado y descifrado de cadenas
- Cifrado y descifrado de secuencias de bytes
- Cifrar y descifrar secuencias de bytes con un conjunto de claves múltiples

Cifrado y descifrado de cadenas

En el siguiente ejemplo, se muestra cómo utilizar la versión 3. x de SDK de cifrado de AWS para Java para cifrar y descifrar cadenas. Antes de usar la cadena, conviértala en una matriz de bytes.

En este ejemplo se utiliza un AWS KMS anillo de claves. Al cifrar con un AWS KMS anillo de claves, puede usar un ID de clave, un ARN de clave, un nombre de alias o un ARN de alias para identificar las claves de KMS. Al descifrar, debe utilizar una clave ARN para identificar las claves de KMS.

Cuando se llama al método `encryptData()`, este devuelve un mensaje cifrado (`CryptoResult`) que incluye el texto cifrado, las claves de datos cifradas y el contexto de cifrado. Cuando llama a `getResult` en el objeto `CryptoResult`, se devuelve una versión de cadena codificada en base 64 del mensaje cifrado que puede pasar al método `decryptData()`.

Del mismo modo, cuando llamas`decryptData()`, el `CryptoResult` objeto que devuelve contiene el mensaje de texto sin formato y un identificador. AWS KMS key Antes de que la aplicación devuelva el texto sin formato, compruebe que el AWS KMS key identificador y el contexto de cifrado del mensaje cifrado son los esperados.

```
// 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
 * </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);
  }
}
```

Cifrado y descifrado de secuencias de bytes

En el siguiente ejemplo, se muestra cómo utilizarlos AWS Encryption SDK para cifrar y descifrar secuencias de bytes.

En este ejemplo se utiliza un anillo de claves AES sin procesar.

Al cifrar, en este ejemplo se utiliza el método `AwsCrypto.builder() .withEncryptionAlgorithm()` para especificar un conjunto de algoritmos sin firmas digitales. Al descifrar, para garantizar que el texto cifrado no esté firmado, en este ejemplo se utiliza el método `createUnsignedMessageDecryptingStream()`. El `createUnsignedMessageDecryptingStream()` método falla si encuentra un texto cifrado con una firma digital.

Si está cifrando con el conjunto de algoritmos predeterminado, que incluye firmas digitales, utilice el método `createDecryptingStream()` en su lugar, como se muestra en el siguiente ejemplo.

```
// 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("Exampl
 {
            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");
    }
}
```

Cifrar y descifrar secuencias de bytes con un conjunto de claves múltiples

En el siguiente ejemplo, se muestra cómo utilizarla con un conjunto de claves múltiples. AWS Encryption SDK Cuando se utiliza un conjunto de claves múltiple para cifrar datos, cualquiera de las claves de encapsulamiento en cualquiera de los conjuntos de claves puede descifrar dichos datos. En este ejemplo, se utiliza un AWS KMS llavero y un llavero RSA sin procesar como llaveros secundarios.

En este ejemplo, se cifra con el conjunto de algoritmos predeterminado, que incluye una firma digital. Al transmitir, AWS Encryption SDK publica el texto sin formato después de las comprobaciones de integridad, pero antes de comprobar la firma digital. Para evitar utilizar el texto no cifrado hasta que

se compruebe la firma, en este ejemplo se almacena el texto no cifrado en búfer y se graba en el disco únicamente cuando se han completado el descifrado y la verificación.

```
// 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());

    }
```

```
}
```

SDK de cifrado de AWS para JavaScript

SDK de cifrado de AWS para JavaScript Está diseñado para proporcionar una biblioteca de cifrado del lado del cliente para los desarrolladores que escriben aplicaciones de navegador web JavaScript o aplicaciones de servidor web en Node.js.

Como todas las implementaciones del AWS Encryption SDK, SDK de cifrado de AWS para JavaScript ofrece funciones avanzadas de protección de datos. Esto incluye el cifrado de sobre, la información autenticada adicional (AAD) y los conjuntos de algoritmos de clave simétrica autenticados y seguros, tales como AES-GCM de 256 bits con derivación de clave y firma.

Todas las implementaciones específicas del idioma AWS Encryption SDK están diseñadas para ser interoperables, sujetas a las limitaciones del idioma. Para obtener más información sobre las restricciones lingüísticas, consulte. JavaScript the section called “Compatibilidad”

Más información

- Para obtener más información sobre la programación con SDK de cifrado de AWS para JavaScript, consulte el aws-encryption-sdk-javascriptrepositorio en GitHub.
- Para ver ejemplos de programación, consulte the section called “Ejemplos” los módulos example-browser y example-node del repositorio. aws-encryption-sdk-javascript
- Para ver un ejemplo práctico del uso del código SDK de cifrado de AWS para JavaScript para cifrar datos en una aplicación web, consulte Cómo habilitar el cifrado en un navegador con el archivo Node.js en el SDK de cifrado de AWS para JavaScript blog de seguridad. AWS

Temas

- Compatibilidad del SDK de cifrado de AWS para JavaScript
- Instalación del SDK de cifrado de AWS para JavaScript
- Módulos del SDK de cifrado de AWS para JavaScript
- SDK de cifrado de AWS para JavaScript ejemplos

Compatibilidad del SDK de cifrado de AWS para JavaScript

SDK de cifrado de AWS para JavaScript Está diseñado para ser interoperable con las implementaciones de otros idiomas del. AWS Encryption SDKEn la mayoría de los casos, puede cifrar los datos SDK de cifrado de AWS para JavaScript y descifrarlos con cualquier implementación en otro idioma, incluida la AWS Encryption SDK interfaz de línea de comandos. Y puede utilizarla SDK de cifrado de AWS para JavaScript para descifrar los mensajes cifrados producidos por las implementaciones del en otros idiomas. AWS Encryption SDK

Sin embargo, cuando utilice el SDK de cifrado de AWS para JavaScript, debe tener en cuenta algunos problemas de compatibilidad en la implementación del JavaScript idioma y en los navegadores web.

Además, cuando utilice implementaciones de lenguaje diferentes, asegúrese de configurar proveedores de claves maestras, claves maestras y conjuntos de claves compatibles. Para obtener más información, consulte Compatibilidad de conjuntos de claves.

SDK de cifrado de AWS para JavaScript compatibilidad

La JavaScript implementación de la AWS Encryption SDK se diferencia de las implementaciones de otros lenguajes en los siguientes aspectos:

- La operación de cifrado del SDK de cifrado de AWS para JavaScript no devuelve texto cifrado no enmarcado. Sin embargo, SDK de cifrado de AWS para JavaScript descifrará el texto cifrado enmarcado y no enmarcado devuelto por las implementaciones del. AWS Encryption SDK
- A partir de Node.js versión 12.9.0, Node.js admite las siguientes opciones de encapsulamiento de claves de RSA:
  - OAEP con,, o SHA1 SHA256 SHA384 SHA512
  - OAEP con y con SHA1 MGF1 SHA1
  - PKCS1v15
- Antes de la versión 12.9.0, Node.js admitía las siguientes opciones de encapsulamiento de claves de RSA:
  - OAEP con y con SHA1 MGF1 SHA1
  - PKCS1v15

Compatibilidad del navegador

Algunos navegadores web no admiten operaciones criptográficas básicas que el SDK de cifrado de AWS para JavaScript requiere. Puede compensar algunas de las operaciones faltantes configurando una alternativa para la WebCrypto API que implementa el navegador.

Limitaciones del explorador web

Las siguientes limitaciones son comunes a todos los navegadores web:

- La WebCrypto API no admite el empaquetado de PKCS1v15 claves.
- Los navegadores no admiten claves de 192 bits.

Operaciones criptográficas requeridas

SDK de cifrado de AWS para JavaScript Requiere las siguientes operaciones en los navegadores web. Si un navegador no las admite, es incompatible con el SDK de cifrado de AWS para JavaScript.

- El navegador debe incluir `crypto.getRandomValues()`, que es un método para generar valores criptográficamente aleatorios. Para obtener información sobre las versiones de navegadores web compatibles`crypto.getRandomValues()`, consulte ¿Puedo usar criptomonedas? getRandomValues()? .

Reserva obligatoria

SDK de cifrado de AWS para JavaScript Requiere las siguientes bibliotecas y operaciones en los navegadores web. Si admite un navegador web que no cumple estos requisitos, debe configurar una reserva. De lo contrario, los intentos de SDK de cifrado de AWS para JavaScript utilizarla con el navegador fallarán.

- La WebCrypto API, que realiza operaciones criptográficas básicas en aplicaciones web, no está disponible para todos los navegadores. Para obtener información acerca de las versiones del explorador web que admiten la criptografía web, vea ¿Puedo utilizar criptografía web?.
- Las versiones modernas del navegador web Safari no admiten el cifrado AES-GCM de cero bytes, que sí es obligatorio. AWS Encryption SDK Si el navegador implementa la WebCrypto API, pero no puede usar AES-GCM para cifrar cero bytes, SDK de cifrado de AWS para JavaScript utiliza la biblioteca alternativa solo para el cifrado de cero bytes. Utiliza la API para todas las demás operaciones. WebCrypto

Para configurar una reserva para cualquiera de las limitaciones, agregue las instrucciones siguientes al código. En la función configureFallback, especifique una biblioteca que admita las características que faltan. En el siguiente ejemplo, se utiliza la biblioteca de JavaScript criptografía de Microsoft Research (`msrcrypto`), pero puede sustituirla por una biblioteca compatible. Para ver un ejemplo completo, consulte fallback.ts.

```
import { configureFallback } from '@aws-crypto/client-browser'
configureFallback(msrCrypto)
```

Instalación del SDK de cifrado de AWS para JavaScript

SDK de cifrado de AWS para JavaScript Consiste en una colección de módulos interdependientes. Varios de los módulos son solo colecciones de módulos diseñados para trabajar juntos. Algunos módulos están diseñados para funcionar de forma independiente. Unos cuantos módulos son de uso obligatorio en todas las implementaciones; mientras que otros solo se usan en casos especiales. Para obtener información sobre los módulos del AWS Encryption SDK formulario JavaScript, consulte Módulos del SDK de cifrado de AWS para JavaScript y el `README.md` archivo de cada uno de los módulos del aws-encryption-sdk-javascriptrepositorio en GitHub.

> **Note**
>
> Todas las versiones SDK de cifrado de AWS para JavaScript anteriores a la 2.0.0 están en end-of-supportfase.
> Puede actualizar de forma segura desde la versión 2.0.x y versiones posteriores a la última versión de SDK de cifrado de AWS para JavaScript sin cambios en el código ni en los datos. Sin embargo, nuevas características de seguridad que se introdujeron en la versión 2.0.x no son compatibles con versiones anteriores. Para actualizar desde versiones anteriores a la 1.7.x a la versión 2.0.x y posteriores, primero debe actualizar a la última versión 1.x de SDK de cifrado de AWS para JavaScript. Para obtener más información, consulte Migrando su AWS Encryption SDK.

Para instalar los módulos, utilice el administrador de paquetes npm.

Por ejemplo, para instalar el `client-node` módulo, que incluye todos los módulos que necesita programar con el SDK de cifrado de AWS para JavaScript archivo Node.js, utilice el siguiente comando.

```
npm install @aws-crypto/client-node
```

Para instalar el `client-browser` módulo, que incluye todos los módulos que necesite programar SDK de cifrado de AWS para JavaScript en el navegador, utilice el siguiente comando.

```
npm install @aws-crypto/client-browser
```

Para ver ejemplos prácticos de cómo utilizar el SDK de cifrado de AWS para JavaScript, consulte los ejemplos de los `example-browser` módulos `example-node` y del aws-encryption-sdk-javascriptrepositorio correspondiente GitHub.

Módulos del SDK de cifrado de AWS para JavaScript

Los módulos del SDK de cifrado de AWS para JavaScript facilitan la instalación del código que necesita para sus proyectos.

Módulos para JavaScript Node.js

client-node

Incluye todos los módulos que necesita para programar con Node.js. SDK de cifrado de AWS para JavaScript

caching-materials-manager-node

Exporta las funciones que admiten la función de almacenamiento en caché de claves de datos de SDK de cifrado de AWS para JavaScript Node.js.

decrypt-node

Exporta funciones que descifran y verifican mensajes cifrados que representan datos y flujo de datos. Incluido en el módulo `client-node`.

encrypt-node

Exporta funciones que cifran y firman diferentes tipos de datos. Incluido en el módulo `client-node`.

example-node

Exporta ejemplos prácticos de programación con Node.js. SDK de cifrado de AWS para JavaScript Incluye ejemplos de diferentes tipos de conjuntos de claves y diferentes tipos de datos.

hkdf-node

Exporta una función de derivación de claves (HKDF) basada en HMAC que se utiliza SDK de cifrado de AWS para JavaScript en Node.js en determinados conjuntos de algoritmos. SDK de cifrado de AWS para JavaScript En el navegador, utiliza la función HKDF nativa de la API. WebCrypto

integration-node

Define las pruebas que comprueban que SDK de cifrado de AWS para JavaScript el archivo Node.js es compatible con las implementaciones en otros idiomas del. AWS Encryption SDK

kms-keyring-node

Exporta las funciones que admiten los AWS KMS llaveros en Node.js.

raw-aes-keyring-node

Exporta funciones que admiten conjuntos de claves de AES sin formato en Node.js.

raw-rsa-keyring-node

Exporta funciones que admiten conjuntos de claves de RSA sin formato en Node.js.

Módulos para navegador JavaScript

client-browser

Incluye todos los módulos que necesitas para programar con ellos SDK de cifrado de AWS para JavaScript en el navegador.

caching-materials-manager-browser

Exporta las funciones que admiten la función de almacenamiento JavaScript en caché de claves de datos del navegador.

decrypt-browser

Exporta funciones que descifran y verifican mensajes cifrados que representan datos y flujo de datos.

encrypt-browser

Exporta funciones que cifran y firman diferentes tipos de datos.

example-browser

Ejemplos prácticos de programación con SDK de cifrado de AWS para JavaScript el navegador. Incluye ejemplos de diferentes tipos de conjuntos de claves y diferentes tipos de datos.

integration-browser

Define las pruebas que verifican que el SDK de cifrado de AWS para Java script del navegador es compatible con las implementaciones en otros lenguajes del AWS Encryption SDK.

kms-keyring-browser

Exporta funciones que admiten conjuntos de claves de AWS KMS en el navegador.

raw-aes-keyring-browser

Exporta funciones que admiten conjuntos de claves de AES sin formato en el navegador.

raw-rsa-keyring-browser

Exporta funciones que admiten conjuntos de claves de RSA sin formato en el navegador.

Módulos para todas las implementaciones

cache-material

Admite la característica de almacenamiento en caché de claves de datos. Proporciona código para ensamblar los materiales criptográficos que se almacenan en caché con cada clave de datos.

kms-keyring

Exporta funciones que admiten conjuntos de claves KMS.

material-management

Implementa el administrador de materiales criptográficos (CMM).

raw-keyring

Exporta funciones necesarias para los conjuntos de claves de AES y RSA sin formato.

serialize

Exporta funciones que el SDK utiliza para serializar su salida.

web-crypto-backend

Exporta las funciones que utilizan la WebCrypto API SDK de cifrado de AWS para JavaScript del navegador.

SDK de cifrado de AWS para JavaScript ejemplos

En los siguientes ejemplos se muestra cómo utilizar el SDK de cifrado de AWS para JavaScript para cifrar y descifrar datos.

Puedes encontrar más ejemplos de su uso SDK de cifrado de AWS para JavaScript en los módulos example-node y example-browser del repositorio de. aws-encryption-sdk-javascript GitHub Estos módulos de ejemplo no se instalan al instalar los módulos `client-browser` o `client-node`.

Consulte los ejemplos de código completos: Nodo: kms_simple.ts, Navegador: kms_simple.ts

Temas

- Cifrar datos con un anillo de claves AWS KMS
- Descifrar datos con un anillo de claves AWS KMS

Cifrar datos con un anillo de claves AWS KMS

En el siguiente ejemplo, se muestra cómo utilizar el SDK de cifrado de AWS para JavaScript para cifrar y descifrar una cadena corta o una matriz de bytes.

Este ejemplo incluye un AWS KMS anillo de claves, un tipo de anillo de claves que utiliza un AWS KMS key para generar y cifrar claves de datos. Si necesitas ayuda para crear una AWS KMS key, consulta la sección Creación de claves en la AWS Key Management Service Guía para desarrolladores. Para obtener ayuda para identificarlas AWS KMS keys en un AWS KMS llavero, consulte Identificarse AWS KMS keys en un AWS KMS llavero

Paso 1: Establece la política de compromiso.

A partir de la versión 1.7. x del SDK de cifrado de AWS para JavaScript, puede establecer la política de compromiso al llamar a la nueva `buildClient` función que crea una instancia de un AWS Encryption SDK cliente. La función `buildClient` toma un valor enumerado que representa su política de compromiso. Devuelve funciones actualizadas `encrypt` y `decrypt` que hacen cumplir su política de compromiso al cifrar y descifrar.

En los siguientes ejemplos, se utiliza la `buildClient` función para especificar la política de compromiso predeterminada,. `REQUIRE_ENCRYPT_REQUIRE_DECRYPT` También puede utilizarla `buildClient` para limitar el número de claves de datos cifradas en un mensaje cifrado. Para obtener más información, consulte the section called “Limitar las claves de datos cifrados”.

JavaScript Browser

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

const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)
```

JavaScript Node.js

```
import {
  KmsKeyringNode,
  buildClient,
  CommitmentPolicy,
} from '@aws-crypto/client-node'

const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)
```

Paso 2: crear el conjunto de claves.

Cree un conjunto de AWS KMS claves para el cifrado.

Al cifrar con un AWS KMS anillo de claves, debe especificar una clave generadora, es decir, una AWS KMS key que se utilice para generar la clave de datos en texto plano y cifrarla. También puede especificar cero o más claves adicionales que cifren la misma clave de datos en texto no cifrado. El anillo de claves devuelve la clave de datos en texto plano y una copia cifrada de esa clave de datos para cada AWS KMS key uno de los componentes del conjunto de claves, incluida la clave generadora. Para descifrar los datos, debe descifrar cualquiera de las claves de datos cifradas.

Para especificar el AWS KMS keys anillo de claves de cifrado en SDK de cifrado de AWS para JavaScript, puede utilizar cualquier identificador de clave compatible. AWS KMS En este ejemplo se utiliza una clave generadora, que se identifica mediante su ARN de alias y una clave adicional, que se identifica mediante un ARN de clave.

> **Note**
>
> Si piensa reutilizar el AWS KMS anillo de claves para descifrar, debe utilizar la clave para ARNs identificarlo AWS KMS keys en el anillo de claves.

Antes de ejecutar este código, sustituya los identificadores de ejemplo por AWS KMS key identificadores válidos. Debe tener los permisos necesarios para usar el AWS KMS keys en el conjunto de claves.

JavaScript Browser

Comience por proporcionar sus credenciales al navegador. Los SDK de cifrado de AWS para JavaScript ejemplos utilizan el paquete web. DefinePlugin, que reemplaza las constantes de las credenciales por las credenciales reales. También puede usar cualquier otro método para proporcionar sus credenciales. A continuación, utilice las credenciales para crear un AWS KMS cliente.

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

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

A continuación, especifique AWS KMS keys la clave del generador y la clave adicional. A continuación, cree un AWS KMS conjunto de claves con el AWS KMS cliente y el AWS KMS keys.

```
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 })
```

Paso 3: Defina el contexto de cifrado.

Un contexto de cifrado son datos autenticados adicionales que son arbitrarios y no son secretos. Al proporcionar un contexto de cifrado al cifrar, este vincula AWS Encryption SDK criptográficamente el contexto de cifrado al texto cifrado, de modo que se requiere el mismo contexto de cifrado para descifrar los datos. El uso de un contexto de cifrado es opcional, pero es una práctica recomendada que le aconsejamos.

Cree un objeto simple que incluya los pares de contexto de cifrado. La clave y el valor de cada par deben ser una cadena.

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'
}
```

Paso 4: Cifra los datos.

Para cifrar los datos en texto no cifrado, llame a la función `encrypt`. Introduzca el conjunto de AWS KMS claves, los datos en texto plano y el contexto de cifrado.

La función `encrypt` devuelve un mensaje cifrado (`result`) que contiene los datos cifrados, las claves de datos cifradas y los metadatos importantes, como el contexto de cifrado y la firma.

Puede descifrar este mensaje cifrado utilizando cualquier lenguaje AWS Encryption SDK de programación compatible.

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 })
```

Descifrar datos con un anillo de claves AWS KMS

Puede usarlo SDK de cifrado de AWS para JavaScript para descifrar el mensaje cifrado y recuperar los datos originales.

En este ejemplo, desciframos los datos que hemos cifrado en el ejemplo the section called “Cifrar datos con un anillo de claves AWS KMS”.

Paso 1: Establece la política de compromiso.

A partir de la versión 1.7. x del SDK de cifrado de AWS para JavaScript, puede establecer la política de compromiso al llamar a la nueva `buildClient` función que crea una instancia de un AWS Encryption SDK cliente. La función `buildClient` toma un valor enumerado que representa su política de compromiso. Devuelve funciones actualizadas `encrypt` y `decrypt` que hacen cumplir su política de compromiso al cifrar y descifrar.

En los siguientes ejemplos, se utiliza la `buildClient` función para especificar la política de compromiso predeterminada,. `REQUIRE_ENCRYPT_REQUIRE_DECRYPT` También puede utilizarla `buildClient` para limitar el número de claves de datos cifradas en un mensaje cifrado. Para obtener más información, consulte the section called “Limitar las claves de datos cifrados”.

JavaScript Browser

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

const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)
```

JavaScript Node.js

```
import {
  KmsKeyringNode,
  buildClient,
  CommitmentPolicy,
} from '@aws-crypto/client-node'

const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)
```

Paso 2: crear el conjunto de claves.

Para descifrar los datos, pase el mensaje cifrado (`result`) que la función `encrypt` ha devuelto. El mensaje cifrado contiene los datos cifrados, las claves de datos cifradas y metadatos importantes, como el contexto de cifrado y la firma.

También debe especificar un conjunto de claves de AWS KMS al descifrar. Puede utilizar el mismo conjunto de claves que se utilizó para cifrar los datos u otro conjunto de claves. Para tener éxito, al menos una AWS KMS key de las claves de descifrado debe poder descifrar una de las claves de datos cifradas del mensaje cifrado. Dado que no se generan claves de datos, no es necesario especificar una clave generadora en un conjunto de claves de descifrado. Si lo hace, la clave generadora y las claves adicionales se tratarán de la misma manera.

Para especificar un AWS KMS key anillo de claves de descifrado en SDK de cifrado de AWS para JavaScript, debe utilizar la clave ARN. De lo contrario, no AWS KMS key se reconoce. Si necesita

ayuda para identificarlo AWS KMS keys en un AWS KMS llavero, consulte Identificarse AWS KMS keys en un AWS KMS llavero

> **Note**
>
> Si utiliza el mismo anillo de claves para cifrar y descifrar, utilice la clave ARNs para identificar el AWS KMS keys conjunto de claves.

En este ejemplo, creamos un anillo de claves que incluye solo uno de los del anillo de claves de AWS KMS keys cifrado. Antes de ejecutar este código, reemplace el ARN de clave de ejemplo por uno válido. Debe tener permiso de `kms:Decrypt` sobre la AWS KMS key.

JavaScript Browser

Comience por proporcionar sus credenciales al navegador. Los SDK de cifrado de AWS para JavaScript ejemplos utilizan el paquete web. DefinePlugin, que reemplaza las constantes de las credenciales por las credenciales reales. También puede usar cualquier otro método para proporcionar sus credenciales. A continuación, utilice las credenciales para crear un AWS KMS cliente.

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

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

A continuación, cree un conjunto de AWS KMS claves con el AWS KMS cliente. En este ejemplo, se utiliza solo una de las del AWS KMS keys anillo de claves de cifrado.

```
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 })
```

Paso 3: Descifrar los datos.

A continuación, llame a la función `decrypt`. Pase el conjunto de claves de descifrado que acaba de crear (`keyring`) y el mensaje cifrado que la función `encrypt` ha devuelto (`result`). AWS Encryption SDK Utiliza el anillo de claves para descifrar una de las claves de datos cifradas. Luego, usa la clave de datos en texto no cifrado para descifrar los datos.

Si la llamada se realiza correctamente, el campo `plaintext` contiene los datos en texto no cifrado (descifrado). El campo `messageHeader` contiene metadatos sobre el proceso de descifrado, como el contexto de cifrado que se ha utilizado para descifrar los datos.

JavaScript Browser

```
const { plaintext, messageHeader } = await decrypt(keyring, result)
```

JavaScript Node.js

```
const { plaintext, messageHeader } = await decrypt(keyring, result)
```

Paso 4: Verificar el contexto de cifrado.

El contexto de cifrado utilizado para descifrar los datos se incluye en el encabezado del mensaje (`messageHeader`) que la función `decrypt` devuelve. Antes de que la aplicación devuelva los datos en texto no cifrado, compruebe que el contexto de cifrado que proporcionó durante el cifrado esté incluido en el contexto de cifrado que se utilizó al descifrar. Una discrepancia podría indicar que se han manipulado los datos o que no ha descifrado el texto cifrado correcto.

Al verificar el contexto de cifrado, no requiere una coincidencia exacta. Cuando se utiliza un algoritmo de cifrado con la firma, el administrador de materiales criptográficos (CMM) agrega la clave de firma pública al contexto de cifrado antes de cifrar el mensaje. Pero todos los pares de contexto de cifrado que ha enviado tienen que estar incluidos en el contexto de cifrado devuelto.

Primero, obtenga el contexto de cifrado del encabezado del mensaje. Luego compruebe que cada par clave-valor del contexto de cifrado original (`context`) coincida con un par clave-valor en el contexto de cifrado devuelto (`encryptionContext`).

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')
})
```

Si la comprobación del contexto de cifrado se realiza correctamente, puede devolver los datos en texto no cifrado.

SDK de cifrado de AWS para Python

En este tema se explica cómo instalar y utilizar el SDK de cifrado de AWS para Python. Para obtener detalles sobre la programación con SDK de cifrado de AWS para Python, consulte el aws-encryption-sdk-pythonrepositorio en GitHub. Para obtener documentación de la API, consulte Leer los documentos.

Temas

- Requisitos previos
- Instalación
- SDK de cifrado de AWS para Python código de ejemplo

Requisitos previos

Antes de instalarlo SDK de cifrado de AWS para Python, asegúrese de cumplir los siguientes requisitos previos.

Una versión compatible de Python

SDK de cifrado de AWS para Python Las versiones 3.2.0 y posteriores requieren Python 3.8 o posterior.

> **Note**
>
> La biblioteca de proveedores de material AWS criptográfico (MPL) es una dependencia opcional de la versión SDK de cifrado de AWS para Python 4. x. Si piensa instalar el MPL, debe usar Python 3.11 o posterior.

Las versiones anteriores son AWS Encryption SDK compatibles con Python 2.7 y Python 3.4 y versiones posteriores, pero le recomendamos que utilice la versión más reciente de AWS Encryption SDK.

Para descargar Python, visite el sitio de descargas de Python.

La herramienta de instalación pip para Python

`pip` está incluido en Python 3.6 y versiones posteriores, aunque es posible que desee actualizarlo. Para obtener más información acerca de la actualización o la instalación de `pip`, consulte la sección sobre la instalación en la documentación de `pip`.

Instalación

Instale la versión más reciente de SDK de cifrado de AWS para Python.

> **Note**
>
> Todas las versiones SDK de cifrado de AWS para Python anteriores a la 3.0.0 están en end-of-supportfase.
> Puede actualizar de forma segura desde la versión 2.0.x y versiones posteriores a la última versión de AWS Encryption SDK sin cambios en el código ni en los datos. Sin embargo, nuevas características de seguridad que se introdujeron en la versión 2.0.x no

son compatibles con versiones anteriores. Para actualizar desde versiones anteriores a la 1.7.x a la versión 2.0.x y posteriores, primero debe actualizar a la última versión 1.x de AWS Encryption SDK. Para obtener más información, consulte Migrando su AWS Encryption SDK.

Se utiliza `pip` para instalar el SDK de cifrado de AWS para Python, como se muestra en los siguientes ejemplos.

Para instalar la versión más reciente

```
pip install "aws-encryption-sdk[MPL]"
```

El `[MPL]` sufijo instala la biblioteca de proveedores de material AWS criptográfico (MPL). La MPL contiene estructuras para cifrar y descifrar los datos. La MPL es una dependencia opcional de la introducida en la SDK de cifrado de AWS para Python versión 4. x. Recomendamos encarecidamente instalar el MPL. Sin embargo, si no tiene intención de utilizar el MPL, puede omitir el sufijo. `[MPL]`

Para obtener más información acerca de cómo utilizar pip para instalar y actualizar paquetes, consulte Instalación de paquetes.

SDK de cifrado de AWS para Python Requiere la biblioteca de criptografía (pyca/cryptography) en todas las plataformas. Todas las versiones de `pip` instalan y compilan automáticamente la biblioteca de `cryptography` en Windows. La versión 8.1 de `pip` y las versiones posteriores se instalan y compilan automáticamente `cryptography` en Linux. Si utiliza una versión anterior de `pip` y su entorno Linux no dispone de las herramientas necesarias para crear la biblioteca `cryptography`, tiene que instalarlas. Para obtener más información, consulte Building cryptography on Linux.

Las versiones 1.10.0 y 2.5.0 sitúan la dependencia criptográfica entre las versiones 2.5.0 y SDK de cifrado de AWS para Python 3.3.2. Otras versiones SDK de cifrado de AWS para Python instalan la última versión de la criptografía. Si necesita una versión de criptografía posterior a la 3.3.2, le recomendamos que utilice la última versión principal del SDK de cifrado de AWS para Python.

Para obtener la última versión de desarrollo de SDK de cifrado de AWS para Python, vaya al aws-encryption-sdk-pythonrepositorio de GitHub.

Después de instalar el SDK de cifrado de AWS para Python, comience consultando el código de ejemplo de Python en esta guía.

SDK de cifrado de AWS para Python código de ejemplo

Los siguientes ejemplos muestran cómo utilizar el SDK de cifrado de AWS para Python para cifrar y descifrar datos.

Los ejemplos de esta sección muestran cómo utilizar la versión 4. x de SDK de cifrado de AWS para Python con la dependencia opcional de la biblioteca de proveedores de material criptográfico (`aws-cryptographic-material-providers`). Para ver ejemplos que utilizan versiones anteriores o instalaciones sin la biblioteca de proveedores de materiales (MPL), busca tu versión en la lista de versiones del aws-encryption-sdk-pythonrepositorio de. GitHub

Cuando usas la versión 4. x SDK de cifrado de AWS para Python con la MPL, utiliza anillos de claves para realizar el cifrado de sobres. AWS Encryption SDK Proporciona anillos de claves que son compatibles con los proveedores de claves maestras que utilizaba en versiones anteriores. Para obtener más información, consulte the section called “Compatibilidad de conjuntos de claves”. Para ver ejemplos sobre la migración de proveedores de claves maestras a conjuntos de claves, consulte los ejemplos de migración en el repositorio de; `aws-encryption-sdk-python` GitHub

Temas

- Cifrado y descifrado de cadenas
- Cifrado y descifrado de secuencias de bytes

Cifrado y descifrado de cadenas

El siguiente ejemplo muestra cómo utilizarlos AWS Encryption SDK para cifrar y descifrar cadenas. En este ejemplo, se utiliza un AWS KMS anillo de claves con una clave KMS de cifrado simétrico.

En este ejemplo, se crea una instancia del AWS Encryption SDK cliente con la política de compromiso predeterminada,. `REQUIRE_ENCRYPT_REQUIRE_DECRYPT` Para obtener más información, consulte the section called “Establecer su política de compromiso”.

```
# Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
This example sets up the KMS Keyring

The AWS KMS keyring uses symmetric encryption KMS keys to generate, encrypt and
decrypt data keys. This example creates a KMS Keyring and then encrypts a custom input
 EXAMPLE_DATA
```

```
with an encryption context. This example also includes some sanity checks for
 demonstration:
1. Ciphertext and plaintext data are not the same
2. Encryption context is correct in the decrypted message header
3. Decrypted plaintext value matches EXAMPLE_DATA
These sanity checks are for demonstration in the example only. You do not need these in
 your code.

AWS KMS keyrings can be used independently or in a multi-keyring with other keyrings
of the same or a different type.

"""

import boto3
from aws_cryptographic_material_providers.mpl import AwsCryptographicMaterialProviders
from aws_cryptographic_material_providers.mpl.config import MaterialProvidersConfig
from aws_cryptographic_material_providers.mpl.models import CreateAwsKmsKeyringInput
from aws_cryptographic_material_providers.mpl.references import IKeyring
from typing import Dict  # noqa pylint: disable=wrong-import-order

import aws_encryption_sdk
from aws_encryption_sdk import CommitmentPolicy

EXAMPLE_DATA: bytes = b"Hello World"


def encrypt_and_decrypt_with_keyring(
    kms_key_id: str
):
    """Demonstrate an encrypt/decrypt cycle using an AWS KMS keyring.

    Usage: encrypt_and_decrypt_with_keyring(kms_key_id)
    :param kms_key_id: KMS Key identifier for the KMS key you want to use for
 encryption and
    decryption of your data keys.
    :type kms_key_id: string

    """
    # 1. Instantiate the encryption SDK client.
    # 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 were to build the client as
    # `client = aws_encryption_sdk.EncryptionSDKClient()`.
    client = aws_encryption_sdk.EncryptionSDKClient(
        commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
    )

    # 2. Create a boto3 client for KMS.
    kms_client = boto3.client('kms', region_name="us-west-2")

    # 3. Optional: create encryption context.
    # Remember that your encryption context is NOT SECRET.
    encryption_context: Dict[str, str] = {
        "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",
    }

    # 4. Create your keyring
    mat_prov: AwsCryptographicMaterialProviders = AwsCryptographicMaterialProviders(
        config=MaterialProvidersConfig()
    )

    keyring_input: CreateAwsKmsKeyringInput = CreateAwsKmsKeyringInput(
        kms_key_id=kms_key_id,
        kms_client=kms_client
    )

    kms_keyring: IKeyring = mat_prov.create_aws_kms_keyring(
        input=keyring_input
    )

    # 5. Encrypt the data with the encryptionContext.
    ciphertext, _ = client.encrypt(
        source=EXAMPLE_DATA,
        keyring=kms_keyring,
        encryption_context=encryption_context
    )

    # 6. Demonstrate that the ciphertext and plaintext are different.
```

```
    # (This is an example for demonstration; you do not need to do this in your own
 code.)
    assert ciphertext != EXAMPLE_DATA, \
        "Ciphertext and plaintext data are the same. Invalid encryption"

    # 7. Decrypt your encrypted data using the same keyring you used on encrypt.
    plaintext_bytes, _ = client.decrypt(
        source=ciphertext,
        keyring=kms_keyring,
        # Provide the encryption context that was supplied to the encrypt method
        encryption_context=encryption_context,
    )

    # 8. Demonstrate that the decrypted plaintext is identical to the original
 plaintext.
    # (This is an example for demonstration; you do not need to do this in your own
 code.)
    assert plaintext_bytes == EXAMPLE_DATA, \
        "Decrypted plaintext should be identical to the original plaintext. Invalid
 decryption"
```

Cifrado y descifrado de secuencias de bytes

El siguiente ejemplo muestra cómo utilizarla AWS Encryption SDK para cifrar y descifrar secuencias de bytes. En este ejemplo se utiliza un anillo de claves AES sin procesar.

En este ejemplo, se crea una instancia del AWS Encryption SDK cliente con la política de compromiso predeterminada,. `REQUIRE_ENCRYPT_REQUIRE_DECRYPT` Para obtener más información, consulte the section called “Establecer su política de compromiso”.

```
# Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
This example demonstrates file streaming for encryption and decryption.

File streaming is useful when the plaintext or ciphertext file/data is too large to
 load into
memory. Therefore, the AWS Encryption SDK allows users to stream the data, instead of
 loading it
all at once in memory. In this example, we demonstrate file streaming for encryption
 and decryption
using a Raw AES keyring. However, you can use any keyring with streaming.
```

```
This example creates a Raw AES Keyring and then encrypts an input stream from the file
`plaintext_filename` with an encryption context to an output (encrypted) file
 `ciphertext_filename`.
It then decrypts the ciphertext from `ciphertext_filename` to a new file
 `decrypted_filename`.
This example also includes some sanity checks for demonstration:
1. Ciphertext and plaintext data are not the same
2. Encryption context is correct in the decrypted message header
3. Decrypted plaintext value matches EXAMPLE_DATA
These sanity checks are for demonstration in the example only. You do not need these in
 your code.

See raw_aes_keyring_example.py in the same directory for another raw AES keyring
 example
in the AWS Encryption SDK for Python.
"""
import filecmp
import secrets

from aws_cryptographic_material_providers.mpl import AwsCryptographicMaterialProviders
from aws_cryptographic_material_providers.mpl.config import MaterialProvidersConfig
from aws_cryptographic_material_providers.mpl.models import AesWrappingAlg,
 CreateRawAesKeyringInput
from aws_cryptographic_material_providers.mpl.references import IKeyring
from typing import Dict  # noqa pylint: disable=wrong-import-order

import aws_encryption_sdk
from aws_encryption_sdk import CommitmentPolicy


def encrypt_and_decrypt_with_keyring(
    plaintext_filename: str,
    ciphertext_filename: str,
    decrypted_filename: str
):
    """Demonstrate a streaming encrypt/decrypt cycle.

    Usage: encrypt_and_decrypt_with_keyring(plaintext_filename
                                            ciphertext_filename
                                            decrypted_filename)
    :param plaintext_filename: filename of the plaintext data
    :type plaintext_filename: string
    :param ciphertext_filename: filename of the ciphertext data
    :type ciphertext_filename: string
```

```
    :param decrypted_filename: filename of the decrypted data
    :type decrypted_filename: string
    """
    # 1. Instantiate the encryption SDK client.
    # 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 were to build the client as
    # `client = aws_encryption_sdk.EncryptionSDKClient()`.
    client = aws_encryption_sdk.EncryptionSDKClient(
        commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
    )

    # 2. The key namespace and key name are defined by you.
    # and are used by the Raw AES keyring to determine
    # whether it should attempt to decrypt an encrypted data key.
    key_name_space = "Some managed raw keys"
    key_name = "My 256-bit AES wrapping key"

    # 3. Optional: create encryption context.
    # Remember that your encryption context is NOT SECRET.
    encryption_context: Dict[str, str] = {
        "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",
    }

    # 4. Generate a 256-bit AES key to use with your keyring.
    # In practice, you should get this key from a secure key management system such as
an HSM.

    # Here, the input to secrets.token_bytes() = 32 bytes = 256 bits
    static_key = secrets.token_bytes(32)

    # 5. Create a Raw AES keyring
    # We choose to use a raw AES keyring, but any keyring can be used with streaming.
    mat_prov: AwsCryptographicMaterialProviders = AwsCryptographicMaterialProviders(
        config=MaterialProvidersConfig()
```

```
    )

    keyring_input: CreateRawAesKeyringInput = CreateRawAesKeyringInput(
        key_namespace=key_name_space,
        key_name=key_name,
        wrapping_key=static_key,
        wrapping_alg=AesWrappingAlg.ALG_AES256_GCM_IV12_TAG16
    )

    raw_aes_keyring: IKeyring = mat_prov.create_raw_aes_keyring(
        input=keyring_input
    )

    # 6. Encrypt the data stream with the encryptionContext
    with open(plaintext_filename, 'rb') as pt_file, open(ciphertext_filename, 'wb') as
ct_file:
        with client.stream(
            mode='e',
            source=pt_file,
            keyring=raw_aes_keyring,
            encryption_context=encryption_context
        ) as encryptor:
            for chunk in encryptor:
                ct_file.write(chunk)

    # 7. Demonstrate that the ciphertext and plaintext are different.
    # (This is an example for demonstration; you do not need to do this in your own
code.)
    assert not filecmp.cmp(plaintext_filename, ciphertext_filename), \
        "Ciphertext and plaintext data are the same. Invalid encryption"

    # 8. Decrypt your encrypted data stream using the same keyring you used on
encrypt.
    with open(ciphertext_filename, 'rb') as ct_file, open(decrypted_filename, 'wb') as
pt_file:
        with client.stream(
            mode='d',
            source=ct_file,
            keyring=raw_aes_keyring,
            encryption_context=encryption_context
        ) as decryptor:
            for chunk in decryptor:
                pt_file.write(chunk)
```

```
    # 10. Demonstrate that the decrypted plaintext is identical to the original
plaintext.
    # (This is an example for demonstration; you do not need to do this in your own
code.)
    assert filecmp.cmp(plaintext_filename, decrypted_filename), \
        "Decrypted plaintext should be identical to the original plaintext. Invalid
decryption"
```

AWS Encryption SDK para Rust

En este tema se explica cómo instalar y usar AWS Encryption SDK for Rust. Para obtener más información sobre la programación AWS Encryption SDK para Rust, consulte el directorio Rust del aws-encryption-sdk repositorio en GitHub.

La AWS Encryption SDK versión para Rust se diferencia de algunas de las implementaciones de otros lenguajes de programación AWS Encryption SDK en los siguientes aspectos:

- No hay soporte para el almacenamiento en caché de claves de datos. Sin embargo, AWS Encryption SDK para Rust es compatible con el anillo de claves AWS KMS jerárquico, una solución alternativa de almacenamiento en caché de materiales criptográficos.
- No es compatible con el streaming de datos

La versión AWS Encryption SDK para Rust incluye todas las funciones de seguridad introducidas en las versiones 2.0. x y versiones posteriores de las implementaciones en otros idiomas del AWS Encryption SDK. Sin embargo, si está utilizando Rust AWS Encryption SDK para descifrar datos cifrados con una versión anterior a la 2.0. x versión de la implementación en otro idioma de la AWS Encryption SDK, es posible que deba ajustar su política de compromisos. Para obtener más información, consulte ¿Cómo establecer su política de compromiso?.

El AWS Encryption SDK de Rust es un producto de Dafny, un lenguaje de verificación formal en el que se escriben las especificaciones, el código para implementarlas y las pruebas para probarlas. AWS Encryption SDK El resultado es una biblioteca que implementa las características del AWS Encryption SDK en una trama que garantiza la corrección funcional.

Más información

- Para ver ejemplos que muestran cómo configurar las opciones del AWS Encryption SDK, como especificar un conjunto de algoritmos alternativo, limitar las claves de datos cifrados y utilizar claves AWS KMS multirregionales, consulte. Configuración del AWS Encryption SDK

- Para ver ejemplos que muestran cómo configurar y usar AWS Encryption SDK para Rust, consulta los ejemplos de Rust en el aws-encryption-sdk repositorio de Rust. GitHub

Temas

- Requisitos previos
- Instalación
- AWS Encryption SDK para el código de ejemplo de Rust

Requisitos previos

Antes de instalar el AWS Encryption SDK para Rust, asegúrate de cumplir los siguientes requisitos previos.

Instale Rust y Cargo

Instala la versión estable actual de Rust usando rustup.

Para obtener más información sobre la descarga e instalación de rustup, consulta los procedimientos de instalación en The Cargo Book.

Instalación

El AWS Encryption SDK de Rust está disponible en forma de `aws-esdk`caja en Crates.io. Para obtener más información sobre cómo instalar y compilar AWS Encryption SDK para Rust, consulta el archivo README.md en el repositorio de Rust. aws-encryption-sdk GitHub

Puedes instalar el AWS Encryption SDK para Rust de las siguientes maneras.

Manualmente

Para instalar el repositorio AWS Encryption SDK para Rust, clona o descarga el aws-encryption-sdk GitHub repositorio.

Usando Crates.io

Ejecute el siguiente comando Cargo en el directorio de su proyecto:

```
cargo add aws-esdk
```

O añade la siguiente línea a tu Cargo.toml:

```
aws-esdk = "<version>"
```

AWS Encryption SDK para el código de ejemplo de Rust

Los siguientes ejemplos muestran los patrones de codificación básicos que se utilizan al programar con Rust. AWS Encryption SDK En concreto, se crea una instancia de la biblioteca AWS Encryption SDK y de proveedores de materiales. A continuación, antes de llamar a cada método, se crea una instancia del objeto que define la entrada del método.

Para ver ejemplos que muestran cómo configurar las opciones del AWS Encryption SDK, como especificar un conjunto de algoritmos alternativo y limitar las claves de datos cifrados, consulta los ejemplos de Rust en el aws-encryption-sdk repositorio de. GitHub

Cifrar y descifrar datos en el caso de Rust AWS Encryption SDK

Este ejemplo muestra el patrón básico para cifrar y descifrar datos. Cifra un archivo pequeño con claves de datos que están protegidas por una AWS KMS clave de empaquetado.

Paso 1: Instanciar el. AWS Encryption SDK

Utilizará los métodos descritos en AWS Encryption SDK para cifrar y descifrar los datos.

```
let esdk_config = AwsEncryptionSdkConfig::builder().build()?;
let esdk_client = esdk_client::Client::from_conf(esdk_config)?;
```

Paso 2: Crea un AWS KMS cliente.

```
let sdk_config =
 aws_config::load_defaults(aws_config::BehaviorVersion::latest()).await;
let kms_client = aws_sdk_kms::Client::new(&sdk_config);
```

Opcional: cree su contexto de cifrado.

```
let encryption_context = HashMap::from([
    ("encryption".to_string(), "context".to_string()),
    ("is not".to_string(), "secret".to_string()),
```

```
    ("but adds".to_string(), "useful metadata".to_string()),
    ("that can help you".to_string(), "be confident that".to_string()),
    ("the data you are handling".to_string(), "is what you think it
 is".to_string()),
]);
```

Paso 3: Cree una instancia de la biblioteca de proveedores de materiales.

Utilizará los métodos de la biblioteca de proveedores de materiales para crear los conjuntos de claves que especifican qué claves protegen sus datos.

```
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;
```

Paso 4: Crea un AWS KMS llavero.

Para crear el conjunto de claves, llame al método de conjunto de claves con el objeto de entrada del conjunto de claves. En este ejemplo, se utiliza el `create_aws_kms_keyring()` método y se especifica una clave KMS.

```
let kms_keyring = mpl
    .create_aws_kms_keyring()
    .kms_key_id(kms_key_id)
    .kms_client(kms_client)
    .send()
    .await?;
```

Paso 5: Cifra el texto sin formato.

```
let plaintext = example_data.as_bytes();

let encryption_response = esdk_client.encrypt()
    .plaintext(plaintext)
    .keyring(kms_keyring.clone())
    .encryption_context(encryption_context.clone())
    .send()
    .await?;

let ciphertext = encryption_response
                    .ciphertext
                    .expect("Unable to unwrap ciphertext from encryption response");
```

Paso 6: Descifra tus datos cifrados con el mismo conjunto de claves que utilizaste para cifrar.

```
let decryption_response = esdk_client.decrypt()
    .ciphertext(ciphertext)
    .keyring(kms_keyring)
    // Provide the encryption context that was supplied to the encrypt method
    .encryption_context(encryption_context)
    .send()
    .await?;

let decrypted_plaintext = decryption_response
                            .plaintext
                            .expect("Unable to unwrap plaintext from decryption
 response");
```

AWS Encryption SDK interfaz de línea de comandos

La interfaz de línea de AWS Encryption SDK comandos (CLI de AWS cifrado) le permite utilizarla AWS Encryption SDK para cifrar y descifrar datos de forma interactiva en la línea de comandos y en los scripts. No se requieren conocimientos de criptografía o programación.

> **Note**
>
> Las versiones de la CLI de AWS cifrado anteriores a la 4.0.0 están en end-of-supportfase. Puede actualizar de forma segura desde la versión 2.1.x y versiones posteriores a la última versión de la CLI de cifrado de AWS sin cambios en el código ni en los datos. Sin embargo, se introdujeron nuevas características de seguridad en la versión 2.1.x que no son compatibles con versiones anteriores. Para actualizar desde la versión 1.7. x o anterior, primero debe actualizar a la última versión 1. versión x de la CLI AWS de cifrado. Para obtener más información, consulte Migrando su AWS Encryption SDK.
> Las nuevas funciones de seguridad se publicaron originalmente en las versiones 1.7 de AWS Encryption CLI. x y 2.0. x. Sin embargo, AWS Encryption CLI versión 1.8. x reemplaza a la versión 1.7. x y CLI de AWS cifrado 2.1. x reemplaza a 2.0. x. Para obtener más información, consulte el aviso de seguridad correspondiente en el aws-encryption-sdk-clirepositorio de GitHub.

Como todas las implementaciones de AWS Encryption SDK, la CLI de AWS cifrado ofrece funciones avanzadas de protección de datos. Esto incluye el cifrado de sobre, los datos autenticados adicionales (AAD) y los conjuntos de algoritmos de clave simétrica autenticados y seguros, tales como AES-GCM de 256 bits con derivación de clave compromiso clave y firma.

La CLI de AWS cifrado se basa en Linux, macOS SDK de cifrado de AWS para Pythony Windows y es compatible con ellos. Puede ejecutar comandos y scripts para cifrar y descifrar los datos en el shell que prefiera en Linux o macOS, en una ventana de línea de comandos (cmd.exe) en Windows y en una PowerShell consola de cualquier sistema.

Todas las implementaciones específicas del idioma AWS Encryption SDK, incluida la AWS CLI de cifrado, son interoperables. Por ejemplo, puede cifrar datos con la SDK de cifrado de AWS para Javay descifrarlos con la AWS CLI de cifrado.

En este tema se presenta la CLI de AWS cifrado, se explica cómo instalarla y utilizarla y se proporcionan varios ejemplos que le ayudarán a empezar. Para comenzar rápidamente, consulte Cómo cifrar y descifrar sus datos con la AWS CLI de cifrado en el blog de AWS seguridad. Para obtener información más detallada, consulte Read The Docs y únase a nosotros en el desarrollo de la CLI de AWS cifrado en el aws-encryption-sdk-clirepositorio de GitHub.

Desempeño

La CLI de AWS cifrado se basa en SDK de cifrado de AWS para Python. Cada vez que ejecuta la CLI, se inicia una nueva instancia del tiempo de ejecución de Python. Para mejorar el rendimiento, siempre que sea posible, utilice un solo comando en lugar de una serie de comandos independientes. Por ejemplo, ejecute un comando que procese los archivos de un directorio de forma repetida en lugar de ejecutar comandos independientes para cada archivo.

Temas

- Instalación de la interfaz de línea de AWS Encryption SDK comandos
- Cómo utilizar la CLI AWS de cifrado
- Ejemplos de la CLI AWS de cifrado
- AWS Encryption SDK Referencia de parámetros y sintaxis de CLI
- Versiones de la CLI AWS de cifrado

Instalación de la interfaz de línea de AWS Encryption SDK comandos

En este tema se explica cómo instalar la CLI de AWS cifrado. Para obtener información detallada, consulte el aws-encryption-sdk-clirepositorio en GitHub y lea los documentos.

Temas

- Instalación de los requisitos previos
- Instalación y actualización de la CLI AWS de cifrado

Instalación de los requisitos previos

La CLI de AWS cifrado se basa en SDK de cifrado de AWS para Python. Para instalar la CLI de AWS cifrado, necesita Python y `pip` la herramienta de administración de paquetes de Python. Python y `pip` están disponibles en todas las plataformas con soporte.

Instale los siguientes requisitos previos antes de instalar la CLI de AWS cifrado:

Python

Las versiones 4.2.0 y posteriores de la CLI de AWS cifrado requieren Python 3.8 o posterior.

Las versiones anteriores de la CLI de AWS cifrado admiten Python 2.7 y 3.4 y versiones posteriores, pero le recomendamos que utilice la versión más reciente de la CLI de AWS cifrado.

Python está incluido en la mayoría de las instalaciones de Linux y macOS, pero es necesario actualizar a la versión Python 3.6 o posterior. Le recomendamos que utilice la última versión de Python. Tendrá que instalar Python en Windows, si todavía no está de forma predeterminada. Para descargar e instalar Python, visite el sitio de descargas de Python.

Para determinar si Python está instalado, escriba lo siguiente en la línea de comandos:

```
python
```

Para comprobar la versión de Python, utilice el parámetro `-V` (V mayúscula).

```
python -V
```

En Windows, después de instalar Python, añada la ruta al archivo del `Python.exe` al valor de la variable de entorno Path.

De forma predeterminada, Python se instala en el directorio de todos los usuarios o en un directorio de perfil de usuario (`$home` o `%userprofile%`) en el subdirectorio `AppData\Local\Programs\Python`. Para encontrar la ubicación del archivo `Python.exe` en el sistema, consulte una de las siguientes claves del registro. Puede utilizarla PowerShell para buscar en el registro.

```
PS C:\> dir HKLM:\Software\Python\PythonCore\version\InstallPath
# -or-
PS C:\> dir HKCU:\Software\Python\PythonCore\version\InstallPath
```

pip

`pip` es el administrador de paquetes de Python. Para instalar la CLI de AWS cifrado y sus dependencias, necesita la `pip` versión 8.1 o una versión posterior. Para obtener ayuda con la instalación o actualización de `pip`, consulte la sección de instalación en la documentación de `pip`.

En las instalaciones de Linux, las versiones `pip` anteriores a la 8.1 no pueden crear la biblioteca de criptografía que requiere la CLI de AWS cifrado. Si decide no actualizar su versión de `pip`, puede instalar las herramientas de compilación por separado. Para obtener más información, consulte Building cryptography on Linux.

AWS Command Line Interface

El AWS Command Line Interface (AWS CLI) solo es necesario si utiliza AWS KMS keys in AWS Key Management Service (AWS KMS) con la CLI de AWS cifrado. Si utiliza un proveedor de claves maestras diferente, no AWS CLI es obligatorio.

Para utilizarla AWS KMS keys con la CLI de AWS cifrado, debe instalar y configurar la AWS CLI. La configuración hace que las credenciales que se utilizan para autenticarse AWS KMS estén disponibles para la CLI de AWS cifrado.

Instalación y actualización de la CLI AWS de cifrado

Instale la última versión de la CLI de AWS cifrado. Cuando se instala la CLI de AWS cifrado, se instalan automáticamente las bibliotecas que la CLI necesita, incluidas la SDK de cifrado de AWS para Pythonbiblioteca de criptografía de Python y la. `pip` AWS SDK para Python (Boto3)

> **Note**
>
> Las versiones de la CLI de AWS cifrado anteriores a la 4.0.0 están en end-of-supportfase. Puede actualizar de forma segura desde la versión 2.1.x y versiones posteriores a la última versión de la CLI de cifrado de AWS sin cambios en el código ni en los datos. Sin embargo, se introdujeron nuevas características de seguridad en la versión 2.1.x que no son compatibles con versiones anteriores. Para actualizar desde la versión 1.7. x o anterior, primero debe actualizar a la última versión 1. versión x de la CLI AWS de cifrado. Para obtener más información, consulte Migrando su AWS Encryption SDK.
> Las nuevas funciones de seguridad se publicaron originalmente en las versiones 1.7 de AWS Encryption CLI. x y 2.0. x. Sin embargo, AWS Encryption CLI versión 1.8. x reemplaza a la versión 1.7. x y CLI de AWS cifrado 2.1. x reemplaza a 2.0. x. Para obtener más información, consulte el aviso de seguridad correspondiente en el aws-encryption-sdk-clirepositorio de GitHub.

Para instalar la versión más reciente de la CLI de AWS cifrado

```
pip install aws-encryption-sdk-cli
```

Para actualizar a la última versión de la CLI de AWS cifrado

```
pip install --upgrade aws-encryption-sdk-cli
```

Para encontrar los números de versión de su CLI de AWS cifrado y AWS Encryption SDK

```
aws-encryption-cli --version
```

El resultado muestra los números de versión de ambas bibliotecas.

```
aws-encryption-sdk-cli/2.1.0 aws-encryption-sdk/2.0.0
```

Para actualizar a la última versión de la CLI de AWS cifrado

```
pip install --upgrade aws-encryption-sdk-cli
```

Al instalar la CLI de AWS cifrado, también se instala la versión más reciente de AWS SDK para Python (Boto3), si aún no está instalada. Si Boto3 está instalado, el instalador verifica la versión de Boto3 y la actualiza si es necesario.

Para encontrar la versión de Boto3 que tiene instalada

```
pip show boto3
```

Para actualizar a la versión más reciente de Boto3

```
pip install --upgrade boto3
```

Para instalar la versión de la CLI de AWS cifrado actualmente en desarrollo, consulte el aws-encryption-sdk-clirepositorio en GitHub.

Para obtener más información acerca de cómo utilizar `pip` para instalar y actualizar paquetes de Python, consulte la documentación de pip.

Cómo utilizar la CLI AWS de cifrado

En este tema se explica cómo utilizar los parámetros de la CLI de AWS cifrado. Para ver ejemplos, consulte Ejemplos de la CLI AWS de cifrado. Para ver la documentación completa, consulte Leer los documentos. La sintaxis que se muestra en estos ejemplos corresponde a la versión 2.1 de la CLI de AWS cifrado. x y versiones posteriores.

> **Note**
>
> Las versiones de la CLI de AWS cifrado anteriores a la 4.0.0 están en end-of-supportfase. Puede actualizar de forma segura desde la versión 2.1.x y versiones posteriores a la última versión de la CLI de cifrado de AWS sin cambios en el código ni en los datos. Sin embargo, se introdujeron nuevas características de seguridad en la versión 2.1.x que no son compatibles con versiones anteriores. Para actualizar desde la versión 1.7. x o anterior, primero debe actualizar a la última versión 1. versión x de la CLI AWS de cifrado. Para obtener más información, consulte Migrando su AWS Encryption SDK.
> Las nuevas funciones de seguridad se publicaron originalmente en las versiones 1.7 de AWS Encryption CLI. x y 2.0. x. Sin embargo, AWS Encryption CLI versión 1.8. x reemplaza a la versión 1.7. x y CLI de AWS cifrado 2.1. x reemplaza a 2.0. x. Para obtener más información,

consulte el aviso de seguridad correspondiente en el aws-encryption-sdk-clirepositorio de GitHub.

Para ver un ejemplo que muestra cómo usar la característica de seguridad que limita las claves de datos cifrados, consulte Limitar las claves de datos cifrados.

Para ver un ejemplo que muestra cómo utilizar las claves AWS KMS multirregionales, consulteUso de varias regiones AWS KMS keys.

Temas

- Cómo cifrar y descifrar datos
- Cómo especificar las claves de encapsulamiento
- Cómo proporcionar la entrada
- Cómo especificar la ubicación de salida
- Cómo utilizar un contexto de cifrado
- Cómo especificar una política de compromiso
- Cómo almacenar parámetros en un archivo de configuración

Cómo cifrar y descifrar datos

La CLI de AWS cifrado utiliza las funciones de AWS Encryption SDK para facilitar el cifrado y el descifrado de datos de forma segura.

> **Note**
>
> El parámetro `--master-keys` está obsoleto en la versión 1.8.x de la CLI de cifrado de AWS y se eliminó en la versión 2.1.x. En su lugar, utilice el parámetro `--wrapping-keys`. A partir de la versión 2.1.x, el parámetro `--wrapping-keys` es obligatorio para cifrar y descifrar. Para obtener más información, consulte AWS Encryption SDK Referencia de parámetros y sintaxis de CLI.

- Al cifrar datos en la CLI de AWS cifrado, se especifican los datos de texto sin formato y una clave de empaquetado (o clave maestra), como una AWS KMS key in AWS Key Management Service (AWS KMS). Si utiliza un proveedor de claves maestras personalizadas, también debe

especificarlo. También debe especificar las ubicaciones de salida del mensaje cifrado y de los metadatos relativos a la operación de cifrado. El contexto de cifrado es opcional, pero se recomienda.

En la versión 1.8.x, se requiere el parámetro `--commitment-policy` cuando usa el parámetro `--wrapping-keys`; de lo contrario, no será válido. En la versión 2.1.x, el parámetro `--commitment-policy` es opcional, pero se recomienda.

```
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
```

La CLI de AWS cifrado cifra los datos con una clave de datos única. Luego, cifra la clave de datos con las claves de encapsulamiento que especifique. Devuelve un mensaje cifrado y los metadatos relativos a la operación. El mensaje cifrado contiene los datos cifrados (texto cifrado) y una copia cifrada de la clave de datos. No tiene que preocuparse por almacenar, administrar ni perder la clave de datos.

- Al descifrar datos, en el mensaje cifrado se transmiten el contexto de cifrado opcional y las ubicaciones de salida del texto no cifrado y de los metadatos. También debe especificar las claves de empaquetado que la CLI de AWS cifrado puede usar para descifrar el mensaje o decirle a la CLI de AWS encriptación que puede usar cualquier clave de empaquetado que cifre el mensaje.

  A partir de la versión 1.8.x, el parámetro `--wrapping-keys` es opcional al descifrar, pero se recomienda. A partir de la versión 2.1.x, el parámetro `--wrapping-keys` es obligatorio para cifrar y descifrar.

  Al descifrar, puede utilizar el atributo key del parámetro `--wrapping-keys` para especificar las claves de encapsulamiento que descifran los datos. Especificar una clave de AWS KMS empaquetado al descifrar es opcional, pero es una práctica recomendada ya que evita que utilice una clave que no tenía intención de usar. Si utiliza un proveedor de claves maestras personalizadas, debe especificar el proveedor y la clave de encapsulamiento.

Si no utiliza el atributo clave, debe establecer el atributo de detección del `--wrapping-keys` parámetro en`true`, lo que permite a la CLI de AWS cifrado descifrar mediante cualquier clave de empaquetado que haya cifrado el mensaje.

Como práctica recomendada, utilice el parámetro `--max-encrypted-data-keys` para evitar descifrar un mensaje mal formado con un número excesivo de claves de datos cifradas. Especifique el número esperado de claves de datos cifrados (una por cada clave de encapsulamiento utilizada en el cifrado) o un máximo razonable (por ejemplo, 5). Para obtener más información, consulte Limitar las claves de datos cifrados.

El parámetro `--buffer` devuelve el texto no cifrado solo después de procesar todas las entradas, incluida la verificación de la firma digital, si existe alguna.

El parámetro `--decrypt-unsigned` descifra el texto cifrado y garantiza que los mensajes no estén firmados antes del descifrado. Utilice este parámetro si utilizó el parámetro `--algorithm` y seleccionó un conjunto de algoritmos sin firma digital para cifrar los datos. Si el texto cifrado está firmado, se produce un error en el descifrado.

Puede utilizar `--decrypt` o `--decrypt-unsigned` para el descifrado, pero no ambos.

```
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
```

La CLI de AWS cifrado utiliza la clave de empaquetado para descifrar la clave de datos del mensaje cifrado. A continuación, utiliza la clave de datos para descifrar los datos. Devuelve los datos en texto no cifrado y los metadatos relativos a la operación.

Cómo especificar las claves de encapsulamiento

Al cifrar datos en la CLI de AWS cifrado, debe especificar al menos una clave de empaquetado (o clave maestra). Puede usar AWS KMS keys in AWS Key Management Service (AWS KMS),

empaquetar claves de un proveedor de claves maestras personalizado, o ambas. El proveedor de claves maestras personalizadas puede ser cualquiera que sea compatible con Python.

Para especificar las claves de encapsulamiento en las versiones 1.8.x y posteriores, utilice el parámetro `--wrapping-keys` (`-w`). El valor de este parámetro es una colección de atributos con el formato `attribute=value`. Los atributos que se utilizan dependen del proveedor de claves maestras y del comando.

- AWS KMS. En los comandos de cifrado, debe especificar un parámetro `--wrapping-keys` con un atributo key. A partir de la versión 2.1.x, el parámetro `--wrapping-keys` también es obligatorio para descifrar comandos. Al descifrar, el parámetro `--wrapping-keys` requiere un atributo key o un atributo discovery con un valor de `true` (pero no ambos). Otros atributos son opcionales.
- Proveedor de claves maestras personalizadas. Debe especificar un parámetro `--wrapping-keys` en cada comando. El valor del parámetro debe tener los atributos key y provider.

Puede incluir varios parámetros `--wrapping-keys` y varios atributos key en el mismo comando.

Atributos de parámetro de clave de encapsulamiento

El valor del parámetro `--wrapping-keys` se compone de los siguientes atributos y sus valores. Todos los comandos de cifrado requieren un parámetro `--wrapping-keys` (o parámetro `--master-keys`). A partir de la versión 2.1.x, el parámetro `--wrapping-keys` también es obligatorio para descifrar.

Si el nombre o valor de un atributo contiene espacios o caracteres especiales, incluya tanto el nombre como el valor entre comillas. Por ejemplo, `--wrapping-keys key=12345 "provider=my cool provider"`.

Key: especifique una clave de encapsulamiento

Use el atributo key para identificar una clave de encapsulamiento. Al cifrar, el valor puede ser cualquier identificador de clave que el proveedor de claves maestras pueda reconocer.

```
--wrapping-keys key=1234abcd-12ab-34cd-56ef-1234567890ab
```

En un comando de cifrado, debe incluir al menos un atributo key y su valor. Para cifrar su clave de datos en varias claves de encapsulamiento, utilice varios atributos key.

```
aws-encryption-cli --encrypt --wrapping-keys
 key=1234abcd-12ab-34cd-56ef-1234567890ab key=1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d
```

En los comandos de cifrado que utilizan AWS KMS keys, el valor de la clave puede ser el ID de la clave, el ARN de la clave, un nombre de alias o el ARN del alias. Por ejemplo, este comando de cifrado utiliza el ARN de un alias en el valor del atributo key. Para obtener más información sobre los identificadores clave de un AWS KMS key, consulte los identificadores clave en la Guía para desarrolladores.AWS Key Management Service

```
aws-encryption-cli --encrypt --wrapping-keys key=arn:aws:kms:us-
west-2:111122223333:alias/ExampleAlias
```

En los comandos de descifrado que utilizan un proveedor de claves maestras personalizadas, son obligatorios los atributos key y provider.

```
\\ Custom master key provider
aws-encryption-cli --decrypt --wrapping-keys provider='myProvider' key='100101'
```

En los comandos de descifrado que se utilizan AWS KMS, puede utilizar el atributo clave para especificar el que se va AWS KMS keys a utilizar para el descifrado, o el atributo de descubrimiento con un valor de`true`, que permite a la AWS CLI de cifrado utilizar cualquiera de los AWS KMS key que se hayan utilizado para cifrar el mensaje. Si especifica una AWS KMS key, debe ser una de las claves de empaquetado utilizadas para cifrar el mensaje.

Especificar la clave de encapsulamiento es una práctica recomendada de AWS Encryption SDK. Garantiza que utilice lo que pretende AWS KMS key utilizar.

En un comando de descifrado, el valor del atributo key debe ser un ARN de clave.

```
\\ AWS KMS key
aws-encryption-cli --decrypt --wrapping-keys key=arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

Descubrimiento: utilice cualquiera AWS KMS key al descifrar

Si no necesita limitar el uso AWS KMS keys al descifrar, puede usar el atributo de descubrimiento con un valor de. `true` Un valor de `true` permite que la CLI de AWS cifrado descifre mediante cualquiera AWS KMS key que haya cifrado el mensaje. Si no especifica un atributo discovery,

discovery será `false` (predeterminado). El atributo de descubrimiento solo es válido en los comandos de descifrado y solo cuando el mensaje se cifró con. AWS KMS keys

El atributo discovery con un valor de `true` es una alternativa al uso del atributo key para especificar AWS KMS keys. Al descifrar un mensaje cifrado con AWS KMS keys, cada `--wrapping-keys` parámetro debe tener un atributo clave o un atributo de descubrimiento con un valor de`true`, pero no ambos.

Cuando el descubrimiento es cierto, se recomienda utilizar los atributos discovery-partition y discovery-account para limitar el AWS KMS keys uso a los que usted especifique. Cuentas de AWS En el siguiente ejemplo, los atributos de descubrimiento permiten que la CLI de AWS cifrado utilice cualquiera AWS KMS key de los especificados Cuentas de AWS.

```
aws-encryption-cli --decrypt --wrapping-keys \
    discovery=true \
    discovery-partition=aws \
    discovery-account=111122223333 \
    discovery-account=444455556666
```

Provider: especifique el proveedor de claves maestras

El atributo provider identifica el proveedor de claves maestras. El valor predeterminado es `aws-kms`, que representa a AWS KMS. Si utiliza otro proveedor de claves maestras, el atributo provider es obligatorio.

```
--wrapping-keys key=12345 provider=my_custom_provider
```

Para obtener más información sobre cómo utilizar proveedores de claves maestras personalizadas (distintos de AWS KMS), consulte el tema Advanced Configuration del archivo README del repositorio de la CLI de cifrado de AWS.

Región: especifique una Región de AWS

Utilice el atributo de región para especificar el Región de AWS de un AWS KMS key. Este atributo es válido solo en los comandos de cifrado y únicamente cuando el proveedor de clave maestra es AWS KMS.

```
--encrypt --wrapping-keys key=alias/primary-key region=us-east-2
```

AWS Los comandos CLI de cifrado utilizan lo Región de AWS que se especifica en el valor del atributo clave si incluye una región, como un ARN. Si el valor clave especifica un Región de AWS, se omite el atributo de la región.

El atributo region tiene prioridad sobre las demás especificaciones de la región. Si no utiliza un atributo de región, los comandos de CLI de AWS cifrado utilizan el Región de AWS especificado en el perfil AWS CLI designado, si lo hubiera, o en el perfil predeterminado.

Profile: permite especificar un perfil con nombre

Utilice el atributo profile para especificar un perfil con nombre de la AWS CLI . Los perfiles con nombre pueden contener credenciales y una Región de AWS. Este atributo solo es válido cuando el proveedor de claves maestras es AWS KMS.

```
--wrapping-keys key=alias/primary-key profile=admin-1
```

Puede utilizar el atributo profile para especificar unas credenciales alternativas en los comandos de cifrado y descifrado. En un comando de cifrado, la CLI de AWS cifrado utiliza Región de AWS el del perfil nombrado solo cuando el valor de la clave no incluye una región y no hay ningún atributo de región. En un comando de descifrado, se ignora Región de AWS el perfil del nombre.

Cómo especificar varias claves de encapsulamiento

Puede especificar varias claves de encapsulamiento (o claves maestras) en cada comando.

Si especifica más de una clave de encapsulamiento, la primera de ellas genera y cifra la clave de datos que se utiliza para cifrar sus datos. Las otras claves de encapsulamiento cifran la misma clave de datos. El mensaje cifrado obtenido contiene los datos cifrados ("texto cifrado") y una colección de claves de datos cifradas, cada una de ellas cifrada mediante cada clave de encapsulamiento. Cualquiera de las claves de encapsulamiento permite descifrar una clave de datos y luego los datos.

Hay dos formas de especificar varias claves de encapsulamiento:

- Incluir varios atributos key en el valor del parámetro `--wrapping-keys`.

```
$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
```

- Incluir varios parámetros `--wrapping-keys` en el mismo comando. Utilice esta sintaxis cuando los valores de los atributos que vaya a especificar no deban aplicarse a todas las claves de encapsulamiento del comando.

```
--wrapping-keys region=us-east-2 key=alias/test_key \
--wrapping-keys region=us-west-1 key=alias/test_key
```

El atributo de descubrimiento con un valor de `true` permite a la CLI de AWS cifrado utilizar cualquier elemento AWS KMS key que haya cifrado el mensaje. Si usa varios parámetros `--wrapping-keys` en el mismo comando, el uso de `discovery=true` en cualquier parámetro `--wrapping-keys` anula de manera efectiva los límites del atributo key en otros parámetros `--wrapping-keys`.

Por ejemplo, en el siguiente comando, el atributo clave del primer `--wrapping-keys` parámetro limita la CLI de AWS cifrado a lo especificado AWS KMS key. Sin embargo, el atributo de descubrimiento del segundo `--wrapping-keys` parámetro permite a la CLI de AWS cifrado utilizar cualquiera AWS KMS key de las cuentas especificadas para descifrar el mensaje.

```
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
```

Cómo proporcionar la entrada

La operación de cifrado de la CLI de AWS cifrado toma datos de texto sin formato como entrada y devuelve un mensaje cifrado. La operación de descifrado toma un mensaje cifrado como entrada y devuelve datos en texto no cifrado.

El `--input` parámetro (`-i`), que indica a la CLI de AWS cifrado dónde encontrar la entrada, es obligatorio en todos los comandos de la CLI de AWS cifrado.

Puede proporcionar la entrada de cualquiera de las siguientes formas:

- Utilice un archivo.

```
--input myData.txt
```

- Utilice un patrón de nombre de archivo.

```
--input testdir/*.xml
```

- Utilice un directorio o patrón de nombre de directorio. Cuando la entrada es un directorio, el parámetro `--recursive` (`-r`, `-R`) es obligatorio.

```
--input testdir --recursive
```

- Canalice la entrada al comando (stdin). Utilice el valor `-` para el parámetro `--input`. El parámetro `--input` siempre es obligatorio.

```
echo 'Hello World' | aws-encryption-cli --encrypt --input -
```

Cómo especificar la ubicación de salida

El `--output` parámetro indica a la CLI de AWS cifrado dónde escribir los resultados de la operación de cifrado o descifrado. Se requiere en todos los comandos de CLI de AWS cifrado. La CLI de cifrado de AWS crea un nuevo archivo de salida por cada archivo de entrada de la operación.

Si ya existe un archivo de salida, de forma predeterminada, la CLI de AWS cifrado imprime una advertencia y, a continuación, sobrescribe el archivo. Para evitar la sobrescritura, utilice el parámetro `--interactive`, que solicita confirmación antes de sobrescribir, o el parámetro `--no-overwrite`, que omite la entrada si el resultado generara una sobrescritura. Para suprimir la advertencia de sobrescritura, utilice `--quiet`. Para capturar los errores y las advertencias de la CLI de AWS cifrado, utilice el operador de `2>&1` redirección para escribirlos en el flujo de salida.

> **Note**
>
> Los comandos que sobrescriben archivos de salida comienzan por eliminar el archivo de salida. Aunque el comando no se ejecute correctamente, el archivo de salida podría sobrescribirse igualmente.

Puede indicar la ubicación de salida de varias maneras.

- Especifique el nombre de archivo. Si especifica una ruta al archivo, todos los directorios de la ruta deben existir antes de que se ejecute el comando.

```
--output myEncryptedData.txt
```

- Especifique un directorio. El directorio de salida debe existir antes de que se ejecute el comando.

  Si la entrada contiene subdirectorios, el comando los reproduce en el directorio especificado.

  ```
  --output Test
  ```

  Cuando la ubicación de salida es un directorio (sin nombres de archivo), la CLI de AWS cifrado crea nombres de archivos de salida en función de los nombres de los archivos de entrada más un sufijo. Las operaciones de cifrado anexan `.encrypted` al nombre del archivo de entrada y las operaciones de descifrado anexan `.decrypted`. Para cambiar el sufijo, utilice el parámetro `--suffix`.

  Por ejemplo, si cifra `file.txt`, el comando de cifrado crea `file.txt.encrypted`. Si descifra `file.txt.encrypted`, el comando de descifrado crea `file.txt.encrypted.decrypted`.

- Escriba en la línea de comandos (stdout). Escriba el valor `-` para el parámetro `--output`. Puede utilizar `--output -` para canalizar el resultado a otro comando o programa.

  ```
  --output -
  ```

Cómo utilizar un contexto de cifrado

La CLI de AWS cifrado le permite proporcionar un contexto de cifrado en los comandos de cifrado y descifrado. No es obligatorio, pero es una práctica criptográfica recomendada que le aconsejamos.

Un contexto de cifrado es un tipo de información autenticada adicional que es arbitraria y no es secreta. En la CLI de cifrado de AWS , el contexto de cifrado consta de una colección de pares `name=value`. Puede utilizar cualquier contenido en los pares; esto incluye información sobre los archivos, datos que le ayuden a encontrar la operación de cifrado en los registros o datos que se requieran para sus concesiones o políticas.

En un comando de cifrado

El contexto de cifrado que se especifica en un comando de cifrado, junto con cualquier par adicional que agregue el CMM, está vinculado criptográficamente a los datos cifrados. También se incluye (en

texto no cifrado) en el mensaje cifrado que devuelve el comando. Si utiliza una AWS KMS key, es posible que el contexto de cifrado también aparezca en texto plano en los registros y registros de auditoría, por ejemplo. AWS CloudTrail

En el siguiente ejemplo se muestra un contexto de cifrado con tres pares de `name=value`.

```
--encryption-context purpose=test dept=IT class=confidential
```

En un comando de descifrado

En un comando de descifrado, el contexto de cifrado ayuda a confirmar que se descifre el mensaje cifrado acertado.

No es preciso proporcionar un contexto de cifrado en un comando de descifrado, aunque sí se haya utilizado al realizar el cifrado. Sin embargo, si lo hace, la CLI de AWS cifrado verifica que todos los elementos del contexto de cifrado del comando decrypt coincidan con un elemento del contexto de cifrado del mensaje cifrado. Si cualquier elemento no coincide, el comando de descifrado genera un error.

Por ejemplo, el siguiente comando descifra el mensaje cifrado solo si su contexto de cifrado incluye `dept=IT`.

```
aws-encryption-cli --decrypt --encryption-context dept=IT ...
```

Un contexto de cifrado es una parte importante de la estrategia de seguridad. Sin embargo, al elegir un contexto de cifrado, es importante recordar que sus valores no son secretos. No incluya datos confidenciales en el contexto de cifrado.

Para especificar un contexto de cifrado

- En un comando de cifrado, utilice el parámetro `--encryption-context` con uno o varios pares `name=value`. Utilice un espacio para separar cada par del siguiente.

  ```
  --encryption-context name=value [name=value] ...
  ```

- En un comando de descifrado, el valor del parámetro `--encryption-context` puede incluir pares `name=value`, elementos `name` (sin valores) o una combinación de ambos.

  ```
  --encryption-context name[=value] [name] [name=value] ...
  ```

Si `name` o `value` de un par `name=value` incluye espacios o caracteres especiales, incluya el par completo entre comillas.

```
--encryption-context "department=software engineering" "Región de AWS=us-west-2"
```

Por ejemplo, este comando de cifrado incluye un contexto de cifrado con dos pares, `purpose=test` y `dept=23`.

```
aws-encryption-cli --encrypt --encryption-context purpose=test dept=23 ...
```

Estos comandos de descifrado se ejecutarían correctamente. El contexto de cifrado de cada comando es un subconjunto del contexto de cifrado original.

```
\\ 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 ...
```

Sin embargo, estos comandos de descifrado generarían un error. El contexto de cifrado del mensaje cifrado no contiene los elementos especificados.

```
aws-encryption-cli --decrypt --encryption-context dept=Finance ...
aws-encryption-cli --decrypt --encryption-context scope ...
```

Cómo especificar una política de compromiso

Para establecer la política de compromiso del comando, utilice el parámetro `--commitment-policy`. Este parámetro se introduce en la versión 1.8.x. Es válido en los comandos de cifrado y descifrado. La política de compromiso que establezca solo es válida para el comando en el que aparece. Si no establece una política de compromiso para un comando, la CLI de AWS cifrado utiliza el valor predeterminado.

Por ejemplo, el siguiente valor de parámetro establece la política de compromiso en `require-encrypt-allow-decrypt`, que siempre cifra con un compromiso clave, pero descifra un texto cifrado que se haya cifrado con o sin compromiso clave.

```
--commitment-policy require-encrypt-allow-decrypt
```

Cómo almacenar parámetros en un archivo de configuración

Puede ahorrar tiempo y evitar errores de escritura guardando los parámetros y valores de la CLI de AWS cifrado utilizados con frecuencia en los archivos de configuración.

Un archivo de configuración es un archivo de texto que contiene parámetros y valores para un comando CLI de AWS cifrado. Cuando se hace referencia a un archivo de configuración en un comando de la CLI de cifrado de AWS , la referencia se sustituye por los parámetros y los valores del archivo de configuración. El efecto es el mismo que si se hubiese escrito el contenido del archivo en la línea de comandos. Un archivo de configuración puede tener cualquier nombre y puede estar ubicado en cualquier directorio al que el usuario actual tenga acceso.

En el siguiente ejemplo de archivo de configuración, `key.conf`, se especifican dos AWS KMS keys en diferentes regiones.

```
--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
```

Para utilizar el archivo de configuración en un comando, el nombre de archivo debe llevar el símbolo de arroba (@) como prefijo. En una PowerShell consola, utilice un carácter de comilla invertida para evitar el signo at (`@).

En el comando de este ejemplo se utiliza el archivo `key.conf` en un comando de cifrado.

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
```

Reglas de los archivos de configuración

A continuación, se enumeran las reglas para utilizar archivos de configuración:

- Puede incluir varios parámetros en cada archivo de configuración y enumerarlos en cualquier orden. Enumere cada parámetro con sus valores (si procede) en una línea independiente.
- Utilice # para agregar un comentario a la totalidad o parte de una línea.
- Puede incluir referencias a otros archivos de configuración. No utilices una tilde para escapar del @ letrero, ni siquiera para entrar. PowerShell
- Si utiliza comillas en un archivo de configuración, el texto entrecomillado no puede abarcar varias líneas.

Por ejemplo, este es el contenido de un archivo `encrypt.conf` de muestra.

```
# 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
```

También puede incluir varios archivos de configuración en un comando. En el comando de este ejemplo se utilizan los archivos de configuración `encrypt.conf` y `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
```

Siguiente: Pruebe los ejemplos de CLI de cifrado de AWS

Ejemplos de la CLI AWS de cifrado

Utilice los siguientes ejemplos para probar la CLI de AWS cifrado en la plataforma que prefiera. Para obtener ayuda con las claves maestras y otros parámetros, consulte Cómo utilizar la CLI AWS

de cifrado. Para obtener rápidamente información de referencia, consulte AWS Encryption SDK Referencia de parámetros y sintaxis de CLI.

> **Note**
>
> Los siguientes ejemplos utilizan la sintaxis de la versión 2.1 de la CLI de AWS cifrado. x. Las nuevas funciones de seguridad se publicaron originalmente en las versiones 1.7 de AWS Encryption CLI. x y 2.0. x. Sin embargo, AWS Encryption CLI versión 1.8. x reemplaza a la versión 1.7. x y CLI de AWS cifrado 2.1. x reemplaza a 2.0. x. Para obtener más información, consulte el aviso de seguridad correspondiente en el aws-encryption-sdk-clirepositorio de GitHub.

Para ver un ejemplo que muestra cómo usar la característica de seguridad que limita las claves de datos cifrados, consulte Limitar las claves de datos cifrados.

Para ver un ejemplo que muestra cómo utilizar las claves AWS KMS multirregionales, consulteUso de varias regiones AWS KMS keys.

Temas

- Cifrado de un archivo
- Descifrado de un archivo
- Cifrado de todos los archivos de un directorio
- Descifrado de todos los archivos de un directorio
- Cifrado y descifrado en la línea de comandos
- Uso de varias claves maestras
- Cifrado y descifrado en scripts
- Uso del almacenamiento en caché de claves de datos

Cifrado de un archivo

En este ejemplo, se utiliza la CLI de AWS cifrado para cifrar el contenido del `hello.txt` archivo, que contiene la cadena «Hello World».

Al ejecutar un comando de cifrado en un archivo, la CLI de AWS cifrado obtiene el contenido del archivo, genera una clave de datos única, cifra el contenido del archivo con la clave de datos y, a continuación, escribe el mensaje cifrado en un archivo nuevo.

El primer comando guarda la clave ARN de an AWS KMS key en la `$keyArn` variable. Al cifrar con un AWS KMS key, puede identificarlo mediante un identificador de clave, un ARN de clave, un nombre de alias o un ARN de alias. Para obtener más información sobre los identificadores clave de un AWS KMS key, consulta los identificadores clave en la Guía para desarrolladores.AWS Key Management Service

El segundo comando cifra el contenido del archivo. El comando utiliza el parámetro `--encrypt` para especificar la operación y el parámetro `--input` para indicar el archivo que se ha de cifrar. El `--wrapping-keys`parámetro y su atributo clave obligatorio indican al comando que utilice el AWS KMS key representado por la clave ARN.

El comando utiliza el parámetro `--metadata-output` para especificar un archivo de texto para los metadatos acerca de la operación de cifrado. El comando aplica la práctica recomendada de utilizar el parámetro `--encryption-context` para especificar un contexto de cifrado.

Este comando también usa el parámetro `--commitment-policy` para establecer la política de compromiso de forma explícita. En la versión 1.8.x, se requiere este parámetro cuando usa el parámetro `--wrapping-keys`. En la versión 2.1.x, el parámetro `--commitment-policy` es opcional, pero se recomienda.

El valor del parámetro `--output`, un punto (.), indica al comando que escriba el archivo de salida en el directorio actual.

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 .
```

Cuando el comando de cifrado se ejecuta correctamente, no devuelve ningún resultado. Para determinar si el comando se ha ejecutado correctamente, consulte el valor booleano de la variable `$?`. Si el comando se ejecuta correctamente, el valor de `$?` es `0` (Bash) o `True` (). PowerShell Cuando se produce un error en el comando, el valor de `$?` es distinto de cero (Bash) o (). `False` PowerShell

Bash

```
$ echo $?
0
```

PowerShell

```
PS C:\> $?
True
```

También puede utilizar un comando de lista de directorios para comprobar que el comando de cifrado ha creado un archivo nuevo, `hello.txt.encrypted`. Como el comando encrypt no especificó un nombre de archivo para el resultado, la CLI de AWS cifrado escribió el resultado en un archivo con el mismo nombre que el archivo de entrada más un `.encrypted` sufijo. Si desea utilizar un sufijo diferente o suprimir el sufijo, utilice el parámetro `--suffix`.

El archivo `hello.txt.encrypted` contiene un mensaje cifrado que incluye el texto cifrado del archivo `hello.txt`, una copia cifrada de la clave de datos y metadatos adicionales, incluido el contexto de cifrado.

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
```

Descifrado de un archivo

En este ejemplo, se utiliza la CLI de AWS cifrado para descifrar el contenido del `Hello.txt.encrypted` archivo que se cifró en el ejemplo anterior.

El comando de descifrado utiliza el parámetro `--decrypt` para indicar la operación y el parámetro `--input` para identificar el archivo que se ha de descifrar. El valor del parámetro `--output` es un punto que representa el directorio actual.

El parámetro `--wrapping-keys` con un atributo key especifica la clave de encapsulamiento que se utiliza para descifrar el mensaje cifrado. En los comandos de descifrado con AWS KMS keys, el valor del atributo clave debe ser un ARN clave. El parámetro `--wrapping-keys` es obligatorio en un comando de descifrado. Si usa AWS KMS keys, puede usar el atributo key para especificar AWS KMS keys para descifrar o el atributo discovery con un valor de `true` (pero no ambos). Si utiliza un proveedor de claves maestras personalizado, los atributos key y provider son obligatorios.

El parámetro `--commitment-policy` es opcional a partir de la versión 2.1.x, pero se recomienda. Su uso explícito deja clara su intención, incluso si especifica el valor predeterminado, `require-encrypt-require-decrypt`.

El parámetro `--encryption-context` es opcional en el comando de descifrado, incluso cuando se proporciona un contexto de cifrado en el comando de cifrado. En este caso, el comando de descifrado utiliza el mismo contexto de cifrado que se proporcionó en el comando de cifrado. Antes de descifrar, la CLI de AWS cifrado verifica que el contexto de cifrado del mensaje cifrado incluya un `purpose=test` par. En caso contrario, el comando de descifrado genera un error.

El parámetro `--metadata-output` especifica un archivo para los metadatos relativos a la operación de descifrado. El valor del parámetro `--output`, un punto (.), escribe el archivo de salida en el directorio actual.

Como práctica recomendada, utilice el parámetro `--max-encrypted-data-keys` para evitar descifrar un mensaje mal formado con un número excesivo de claves de datos cifradas. Especifique el número esperado de claves de datos cifrados (una por cada clave de encapsulamiento utilizada en el cifrado) o un máximo razonable (por ejemplo, 5). Para obtener más información, consulte Limitar las claves de datos cifrados.

El `--buffer` devuelve el texto no cifrado solo después de procesar todas las entradas, incluida la verificación de la firma digital, si existe alguna.

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 .
```

Cuando el comando de descifrado se ejecuta correctamente, no devuelve ningún resultado. Para determinar si el comando se ha ejecutado correctamente, obtenga el valor de la variable `$?`. También puede utilizar un comando de lista de directorios para comprobar que el comando de descifrado ha creado un archivo nuevo con el sufijo `.decrypted`. Para ver el contenido de texto no cifrado, utilice un comando para obtener el contenido del archivo (por ejemplo, `cat` o 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
```

Cifrado de todos los archivos de un directorio

En este ejemplo, se utiliza la CLI de AWS cifrado para cifrar el contenido de todos los archivos de un directorio.

Cuando un comando afecta a varios archivos, la CLI de AWS cifrado procesa cada archivo de forma individual. Obtiene el contenido del archivo, obtiene una clave de datos única para el archivo a partir

de la clave maestra, cifra el contenido del archivo con la clave de datos y escribe los resultados en un archivo nuevo en el directorio de salida. Por lo tanto, los archivos de salida se pueden descifrar de manera independiente.

Esta lista del directorio `TestDir` muestra los archivos de texto no cifrado que deseamos cifrar.

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
```

El primer comando guarda el nombre de recurso de Amazon (ARN) de an AWS KMS key en la `$keyArn` variable.

El segundo comando cifra el contenido de los archivos en el directorio `TestDir` y escribe los archivos con el contenido cifrado en el directorio `TestEnc`. Si el directorio `TestEnc` no existe, el comando genera un error. Dado que la ubicación de entrada es un directorio, el parámetro `--recursive` es obligatorio.

El parámetro `--wrapping-keys` y su atributo key obligatorio especifican la clave de encapsulamiento para utilizar. El comando de cifrado incluye un contexto de cifrado, `dept=IT`. Cuando especifique un contexto de cifrado en un comando que cifre varios archivos, se utilizará el mismo contexto de cifrado para todos ellos.

El comando también tiene un `--metadata-output` parámetro para indicar a la CLI de AWS cifrado dónde escribir los metadatos sobre las operaciones de cifrado. La CLI de AWS cifrado escribe un registro de metadatos para cada archivo que se cifró.

El `--commitment-policy parameter` es opcional a partir de la versión 2.1.x, pero se recomienda. Si el comando o el script fallan porque no pueden descifrar un texto cifrado, la configuración explícita de la política de compromiso puede ayudarle a detectar el problema rápidamente.

Cuando se completa el comando, la CLI de AWS cifrado escribe los archivos cifrados en el `TestEnc` directorio, pero no devuelve ningún resultado.

El último comando muestra los archivos en el directorio `TestEnc`. Hay un archivo de salida de contenido cifrado por cada archivo de entrada de contenido de texto no cifrado. Dado que el comando no ha especificado un sufijo alternativo, el comando de cifrado ha agregado `.encrypted` al nombre de cada archivo de entrada.

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
```

Descifrado de todos los archivos de un directorio

Este ejemplo descifra todos los archivos de un directorio. Comienza por los archivos del directorio `TestEnc` que se han cifrado en el ejemplo anterior.

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
```

Este comando de descifrado descifra todos los archivos del TestEnc directorio y escribe los archivos de texto sin formato en el directorio. TestDec El `--wrapping-keys` parámetro con un atributo clave y un valor de ARN clave indica a la AWS CLI de cifrado qué se debe utilizar AWS KMS keys para

descifrar los archivos. El comando usa el `--interactive` parámetro para indicar a la CLI de AWS cifrado que le pregunte antes de sobrescribir un archivo con el mismo nombre.

Este comando también utiliza el contexto de cifrado que se proporcionó al cifrar los archivos. Al descifrar varios archivos, la CLI de AWS cifrado comprueba el contexto de cifrado de cada archivo. Si se produce un error en la comprobación del contexto de AWS cifrado de algún archivo, la CLI de cifrado rechaza el archivo, escribe una advertencia, registra el error en los metadatos y, a continuación, continúa comprobando los archivos restantes. Si la CLI de AWS cifrado no puede descifrar un archivo por cualquier otro motivo, el comando de descifrado completo fallará inmediatamente.

En este ejemplo, los mensajes cifrados de todos los archivos de entrada contienen el elemento de contexto de cifrado `dept=IT`. Sin embargo, si va a descifrar mensajes con diferentes contextos de cifrado, puede verificar parte del contexto de cifrado. Por ejemplo, si algunos mensajes tuviesen el contexto de cifrado `dept=finance` y otros tuviesen `dept=IT`, podría verificar que el contexto de cifrado siempre contiene el nombre `dept`, sin especificar su valor. Si desea mayor precisión, podría descifrar los archivos mediante comandos independientes.

El comando de descifrado no devuelve ningún resultado, pero puede utilizar un comando de lista de directorios para comprobar que ha creado nuevos archivos con el sufijo `.decrypted`. Para ver el contenido de texto no cifrado, utilice un comando para obtener el contenido del archivo.

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
```

Cifrado y descifrado en la línea de comandos

Estos ejemplos muestran cómo canalizar la entrada a los comandos (stdin) y escribir el resultado en la línea de comandos (stdout). Explican cómo representar stdin y stdout en un comando y cómo utilizar las herramientas de codificación Base64 integradas para evitar que el shell interprete incorrectamente los caracteres no ASCII.

En este ejemplo se canaliza una cadena de texto no cifrado a un comando de cifrado y se guarda el mensaje cifrado en una variable. A continuación, se canaliza el mensaje cifrado de la variable a un comando de descifrado que escribe su resultado en la canalización (stdout).

El ejemplo se compone de tres comandos:

- El primer comando guarda la clave ARN de an AWS KMS key en la `$keyArn` variable.

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'
```

- El segundo comando envía la cadena `Hello World` al comando de cifrado y guarda el resultado en la variable `$encrypted`.

  Los parámetros `--input` y `--output` son obligatorios en todos los comandos de la CLI de cifrado de AWS . Para indicar que la entrada se va a canalizar al comando (stdin), utilice un guion (`-`) para el valor del parámetro `--input`. Para enviar el resultado a la línea de comandos (stdout), utilice un guion para el valor del parámetro `--output`.

  El parámetro `--encode` codifica en Base64 el resultado antes de devolverlo. Esto impide que el shell interprete incorrectamente los caracteres no ASCII del mensaje cifrado.

  Dado que este comando es una mera prueba de concepto, omitimos el contexto de cifrado y suprimimos los metadatos (`-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 `
                                                         --input - --output - --
  encode `
  ```

```
                                                    --wrapping-keys key=
$keyArn
```

- El tercer comando canaliza el mensaje cifrado de la variable `$encrypted` al comando de descifrado.

  Este comando de descifrado utiliza `--input -` para indicar que la entrada procede de la canalización (stdin) y `--output -` para enviar el resultado a la canalización (stdout). (El parámetro de entrada toma la ubicación de la entrada, no los bytes de entrada reales, por lo que no puede utilizar la variable `$encrypted` como valor del parámetro `--input`.)

  En este ejemplo, se utiliza el atributo de descubrimiento del `--wrapping-keys` parámetro para permitir que la CLI de AWS cifrado lo utilice AWS KMS key para descifrar los datos. No especifica una política de compromiso, por lo que utiliza el valor predeterminado para las versiones 2.1.x y posteriores, `require-encrypt-require-decrypt`.

  Debido a que el resultado se ha cifrado y, a continuación, codificado, el comando de descifrado utiliza el parámetro `--decode` para descodificar entrada codificada en Base64 antes de descifrarlo. También puede utilizar el parámetro `--decode` para descodificar la entrada codificada en Base64 antes de cifrarla.

  También en este caso, el comando omite el contexto de cifrado y elimina los metadatos (-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
  ```

También puede realizar las operaciones de cifrado y descifrado en un solo comando sin que intervenga la variable.

Como en el ejemplo anterior, los parámetros `--input` y `--output` tienen un valor `-`; el comando usa el parámetro `--encode` para codificar el resultado y el parámetro `--decode` para descodificar la entrada.

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
```

Uso de varias claves maestras

En este ejemplo, se muestra cómo utilizar varias claves maestras al cifrar y descifrar datos en la AWS CLI de cifrado.

Cuando se utilizan varias claves maestras para cifrar los datos, cualquiera de ellas se puede utilizar para descifrarlos. Esta estrategia garantiza que se puedan descifrar los datos, aunque una de las claves maestras no esté disponible. Si va a almacenar los datos cifrados en varios Regiones de AWS, esta estrategia le permite utilizar una clave maestra en la misma región para descifrar los datos.

Cuando se realiza el cifrado con varias claves maestras, la primera de ellas desempeña una función especial. Genera la clave de datos que se utiliza para cifrar los datos. Las demás claves maestras

cifran la clave de datos en texto no cifrado. El mensaje cifrado resultante incluye los datos cifrados y una colección de claves de datos cifradas, una por cada clave maestra. Aunque la primera clave maestra fue la que generó la clave de datos, cualquiera de ellas puede descifrar una de las claves de datos con el fin de utilizarla para descifrar los datos.

Cifrado con tres claves maestras

El comando de este ejemplo utiliza tres claves maestras para cifrar el archivo `Finance.log`, una en cada una de las tres Regiones de AWS.

Escribe el mensaje cifrado en el directorio `Archive`. El comando utiliza el parámetro `--suffix` sin ningún valor para suprimir el sufijo, de tal forma que los nombres de los archivos de entrada y salida sean iguales.

El comando utiliza el parámetro `--wrapping-keys` con tres atributos key. También puede utilizar varios parámetros `--wrapping-keys` en el mismo comando.

Para cifrar el archivo de registro, la CLI de AWS cifrado solicita a la primera clave de empaquetado de la lista`$key1`,, que genere la clave de datos que utiliza para cifrar los datos. A continuación, utiliza cada una de las demás claves de encapsulamiento para cifrar una copia en texto no cifrado de la misma clave de datos. El mensaje cifrado del archivo de salida incluye las tres claves de datos cifradas.

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
```

Este comando descifra la copia cifrada del archivo `Finance.log` y la escribe en un archivo `Finance.log.clear` del directorio `Finance`. Para descifrar los datos cifrados en tres AWS KMS keys, puede especificar las mismas tres AWS KMS keys o cualquier subconjunto de ellas. En este ejemplo se especifica solo uno de los AWS KMS keys.

Para indicar a la CLI de AWS cifrado qué AWS KMS keys debe utilizar para descifrar los datos, utilice el atributo clave del `--wrapping-keys` parámetro. Al descifrar con AWS KMS keys, el valor del atributo clave debe ser un ARN clave.

Debe tener permiso para llamar a la API de descifrado en el AWS KMS keys lugar que especifique. Para obtener más información, consulte Autenticación y control de acceso de AWS KMS.

Como práctica recomendada, en este ejemplo se utiliza el parámetro `--max-encrypted-data-keys` para evitar descifrar un mensaje mal formado con un número excesivo de claves de datos cifradas. Aunque en este ejemplo se utiliza solo una clave de encapsulamiento para el descifrado, el mensaje cifrado tiene tres (3) claves de datos cifrados, una para cada una de las tres claves de encapsulamiento utilizadas al cifrar. Especifique el número esperado de claves de datos cifrados o un valor máximo razonable, como 5. Si especifica un valor máximo inferior a 3, se produce un error en el comando. Para obtener más información, consulte Limitar las claves de datos cifrados.

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
```

Cifrado y descifrado en scripts

En este ejemplo, se muestra cómo utilizar la CLI de AWS cifrado en los scripts. Puede escribir scripts que se limiten a cifrar y descifrar datos o scripts que cifren o descifren los datos dentro de un proceso de administración de datos.

En este ejemplo, el script obtiene una colección de archivos de registro, los comprime, los cifra y, una vez cifrados, los copia en un bucket de Amazon S3. Este script procesa cada archivo por separado, de forma que podrá descifrarlos y expandirlos por separado.

Cuando comprima y cifre archivos, asegúrese de comprimirlos antes de cifrarlos. Los datos cifrados correctamente no se pueden comprimir.

> **Warning**
>
> Tenga cuidado al comprimir datos que incluyan secretos e información confidencial que algún tercero malintencionado podría tratar de controlar. El tamaño final de los datos comprimidos podría, sin pretenderlo, revelar información confidencial sobre su contenido.

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

# 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"
            }

            # Step 2: Encrypt
            if (-not (Test-Path "$ZipDirectory\$filename.zip"))
            {
                Write-Error "Cannot find zipped file: $ZipDirectory\$filename.zip"
            }
            else
            {
                # 2>&1 captures command output
```

```
                $err = (aws-encryption-cli -e -i "$ZipDirectory\$filename.zip" `
                                           -o $EncryptDirectory `
                                           -m key=$wrappingKeyID provider=
$masterKeyProvider `
                                           -c $EncryptionContext `
                                           --metadata-output $MetadataDirectory `
                                           -v) 2>&1

                # Check error status
                if ($? -eq $false)
                {
                    # Write the error
                    $err
                }
                elseif (Test-Path "$EncryptDirectory\$fileName.zip.encrypted")
                {
                    # Step 3: Write to S3 bucket
                    if ($S3BucketFolder)
                    {
                        Write-S3Object -BucketName $S3Bucket -File
 "$EncryptDirectory\$fileName.zip.encrypted" -Key "$S3BucketFolder/
$fileName.zip.encrypted"

                    }
                    else
                    {
                        Write-S3Object -BucketName $S3Bucket -File
 "$EncryptDirectory\$fileName.zip.encrypted"
                    }
                }
            }
        }
    }
}
```

Uso del almacenamiento en caché de claves de datos

En este ejemplo, se utiliza el almacenamiento en caché de claves de datos en un comando que cifra gran cantidad de archivos.

De forma predeterminada, la CLI de AWS cifrado (y otras versiones de AWS Encryption SDK) generan una clave de datos única para cada archivo que cifra. Aunque el uso de una clave de datos

única para cada operación es una práctica criptográfica recomendada, una reutilización limitada de las claves de datos es aceptable en algunas situaciones. Si está estudiando la posibilidad de utilizar el almacenamiento en caché de claves de datos, consulte a un ingeniero de seguridad para conocer los requisitos de seguridad de la aplicación y determinar los umbrales de seguridad que sean adecuados para su caso.

En este ejemplo, el almacenamiento en caché de claves de datos acelera la operación de cifrado porque reduce la frecuencia de las solicitudes al proveedor de claves maestras.

El comando de este ejemplo cifra un gran directorio con varios subdirectorios que contienen un total aproximado de 800 archivos de registro de pequeño tamaño. El primer comando guarda el ARN de la AWS KMS key en una variable `keyARN`. El segundo comando cifra todos los archivos del directorio de entrada (de forma recursiva) y los escribe en un directorio de archivo. El comando utiliza el parámetro `--suffix` para especificar el sufijo `.archive`.

El parámetro `--caching` permite el almacenamiento en caché de claves de datos. El atributo capacity, que limita el número de claves de datos de la caché, se establece en 1, ya que el procesamiento de archivos en serie nunca utiliza más de una clave de datos a la vez. El atributo max_age, que determina durante cuánto tiempo se puede utilizar la clave de datos almacenada en la memoria caché, se establece en 10 segundos.

El atributo opcional max_messages_encrypted se establece en 10 mensajes, de tal forma que la misma clave de datos nunca se utilice para cifrar más de 10 archivos. Limitar la cantidad de archivos cifrados mediante cada clave de datos reduce la cantidad de archivos que se verían afectadas en el caso poco probable de que se filtrase una clave de datos.

Para ejecutar este comando en los archivos de registro generados por el sistema operativo, es posible que necesite permisos de administrador (`sudo` en Linux; Ejecutar como administrador en 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
```

Para probar el efecto del almacenamiento en caché de las claves de datos, en este ejemplo se utiliza el cmdlet Measure-Command in. PowerShell Cuando se ejecuta este ejemplo sin almacenar las claves de datos en caché, tarda unos 25 segundos en completarse. Este proceso genera una nueva clave de datos para cada archivo del directorio.

```
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
```

El almacenamiento en caché de claves de datos agiliza el proceso, aunque se limite cada clave de datos para un máximo de 10 archivos. Ahora, el comando tarda menos de 12 segundos en completarse y reduce el número de llamadas al proveedor de claves maestras a 1/10 del valor original.

```
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        : 0.00328146222222222
TotalMinutes      : 0.196887733333333
TotalSeconds      : 11.813264
TotalMilliseconds : 11813.264
```

Si se elimina la restricción `max_messages_encrypted`, todos los archivos se cifrarán con la misma clave de datos. Este cambio aumenta el riesgo inherente a la reutilización de claves de datos sin aportar mucha más rapidez al proceso. Sin embargo, reduce a una sola el número de llamadas al proveedor de claves maestras.

```
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 Referencia de parámetros y sintaxis de CLI

En este tema se proporcionan diagramas de sintaxis y descripciones breves de los parámetros para ayudarlo a usar la interfaz de línea de comandos (CLI) del AWS Encryption SDK . Para obtener ayuda con las claves de encapsulamiento y otros parámetros, consulte Cómo utilizar la CLI AWS de cifrado. Para ver ejemplos, consulte Ejemplos de la CLI AWS de cifrado. Para ver la documentación completa, consulte Leer los documentos.

Temas

- AWS Sintaxis de CLI de cifrado
- AWS Parámetros de línea de comandos de la CLI de cifrado
- Parámetros avanzados

AWS Sintaxis de CLI de cifrado

Estos diagramas de sintaxis de la CLI de AWS cifrado muestran la sintaxis de cada tarea que se realiza con la CLI de AWS cifrado. Representan la sintaxis recomendada en la versión 2.1 de la CLI de AWS cifrado. x y versiones posteriores.

Las nuevas funciones de seguridad se publicaron originalmente en las versiones 1.7 de AWS Encryption CLI. x y 2.0. x. Sin embargo, AWS Encryption CLI versión 1.8. x reemplaza a la versión 1.7. x y CLI de AWS cifrado 2.1. x reemplaza a 2.0. x. Para obtener más información, consulte el aviso de seguridad correspondiente en el aws-encryption-sdk-clirepositorio de GitHub.

> **Note**
>
> A menos que se indique lo contrario en la descripción del parámetro, cada parámetro o atributo solo se puede usar una vez en cada comando.
> Si utiliza un atributo que un parámetro no admite, la CLI de AWS cifrado ignora ese atributo no admitido sin que aparezca ninguna advertencia o error.

Obtención de ayuda

Para obtener la sintaxis CLI de AWS cifrado completa con descripciones de parámetros, utilice `--help` o `-h`.

```
aws-encryption-cli (--help | -h)
```

Obtención de la versión

Para obtener el número de versión de la instalación de la CLI de AWS cifrado, utilice `--version`. Asegúrese de incluir la versión cuando haga preguntas, informe de problemas o comparta consejos sobre el uso de la CLI de AWS cifrado.

```
aws-encryption-cli --version
```

Cifrar datos

En el siguiente diagrama de sintaxis se muestran los parámetros que se utilizan en un comando de cifrado encrypt.

```
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]
```

Descifrado de datos

En el siguiente diagrama de sintaxis se muestran los parámetros que se utilizan en un comando de descifrado decrypt.

En la versión 1.8.x, el parámetro `--wrapping-keys` es opcional al descifrar, pero se recomienda. A partir de la versión 2.1.x, el parámetro `--wrapping-keys` es obligatorio para cifrar y descifrar. Para AWS KMS keys, puede usar el atributo key para especificar las claves de encapsulamiento (práctica recomendada) o establecer el atributo discovery en `true`, lo que no limita las claves de encapsulamiento que puede usar la CLI de cifrado de AWS .

```
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]
```

Uso de archivos de configuración

Puede hacer referencia a archivos de configuración que contengan los parámetros y sus valores. Esto equivale a escribir los parámetros y sus valores en el comando. Para ver un ejemplo, consulta Cómo almacenar parámetros en un archivo de configuración.

```
aws-encryption-cli @<configuration_file>

# In a PowerShell console, use a backtick to escape the @.
aws-encryption-cli `@<configuration_file>
```

AWS Parámetros de línea de comandos de la CLI de cifrado

Esta lista proporciona una descripción básica de los parámetros del comando CLI de AWS cifrado. Para obtener una descripción completa, consulte la aws-encryption-sdk-clidocumentación.

--encrypt (-e)

Cifra los datos de entrada. Todos los comandos deben tener un parámetro `--encrypt`, `--decrypt` o `--decrypt-unsigned`

--decrypt (-d)

Descifra los datos de entrada. Todos los comandos deben tener un parámetro `--encrypt`, `--decrypt` o `--decrypt-unsigned`.

--decrypt-unsigned [Introducido en las versiones 1.9.x y 2.2.x]

El parámetro `--decrypt-unsigned` descifra el texto cifrado y garantiza que los mensajes no estén firmados antes del descifrado. Utilice este parámetro si utilizó el parámetro `--algorithm` y seleccionó un conjunto de algoritmos sin firma digital para cifrar los datos. Si el texto cifrado está firmado, se produce un error en el descifrado.

Puede utilizar `--decrypt` o `--decrypt-unsigned` para el descifrado, pero no ambos.

--wrapping-keys (-w) [Introducido en la versión 1.8.x]

Especifica las claves de encapsulamiento (o claves maestras) utilizadas en las operaciones de cifrado y descifrado. Puede utilizar múltiples parámetros `--wrapping-keys` en cada comando.

A partir de la versión 2.1.x, el parámetro `--wrapping-keys` es obligatorio para cifrar y descifrar comandos. En la versión 1.8.x, los comandos de cifrado requieren un parámetro `--wrapping-`

`keys` o `--master-keys`. En los comandos de descifrado de la versión 1.8.x, un parámetro `--wrapping-keys` es opcional, pero se recomienda.

Al utilizar un proveedor de claves maestras personalizadas, los comandos de cifrado y descifrado requieren los atributos key y provider. Cuando se utilizan AWS KMS keys, los comandos de cifrado requieren un atributo clave. Los comandos de descifrado requieren un atributo key o un atributo discovery con un valor de `true` (pero no ambos). Utilizar el atributo key al descifrar es una práctica recomendada de AWS Encryption SDK . Esto es especialmente importante si va a descifrar lotes de mensajes desconocidos, como los que se encuentran en un bucket de Amazon S3 o en una cola de Amazon SQS.

Para ver un ejemplo que muestra cómo utilizar claves AWS KMS multirregionales como claves de empaquetado, consulte. Uso de varias regiones AWS KMS keys

Atributos: el valor del parámetro `--wrapping-keys` se compone de los siguientes atributos. El formato es `attribute_name=value`.

clave

Identifica la clave de encapsulamiento utilizada en la operación. Se expresa en formato de par key=ID. Puede especificar varios atributos key en cada valor del parámetro `--wrapping-keys`.

- Comandos de cifrado: todos los comandos de cifrado requieren el atributo key. Cuando se utiliza un comando AWS KMS key in a encrypt, el valor del atributo clave puede ser un identificador de clave, un ARN de clave, un nombre de alias o un ARN de alias. Para obtener una descripción de los identificadores AWS KMS clave, consulte los identificadores clave en la Guía para desarrolladores.AWS Key Management Service
- Descifrar comandos: al descifrar con AWS KMS keys, el parámetro `--wrapping-keys` requiere un atributo key con un valor de ARN clave o un atributo discovery con un valor de `true` (pero no ambos). Utilizar el atributo key es una práctica recomendada de AWS Encryption SDK . Al descifrar con un proveedor de claves maestras personalizado, se requiere el atributo key.

> **Note**
>
> Para especificar una clave de AWS KMS empaquetado en un comando de descifrado, el valor del atributo clave debe ser un ARN de clave. Si utiliza un

identificador de clave, un nombre de alias o un alias ARN, la AWS CLI de cifrado no reconoce la clave de empaquetado.

Puede especificar varios atributos key en cada valor del parámetro `--wrapping-keys`. Sin embargo, todos los atributos provider, region y profile en un parámetro `--wrapping-keys` se aplican a todas las claves de encapsulamiento del valor del parámetro. Para especificar claves de encapsulamiento con valores de atributos diferentes, utilice varios parámetros `--wrapping-keys` en el comando.

discovery

Permite que la CLI de AWS cifrado utilice cualquiera AWS KMS key para descifrar el mensaje. El valor de discovery puede ser `true` o `false`. El valor predeterminado es `false`. El atributo discovery es válido solo en los comandos de cifrado y únicamente cuando el proveedor de clave maestra es AWS KMS.

Al descifrar con AWS KMS keys, el `--wrapping-keys` parámetro requiere un atributo clave o un atributo de detección con un valor de `true` (pero no ambos). Si usa el atributo key, puede usar un atributo discovery con un valor de `false` para rechazar explícitamente la detección.

- `False`(predeterminado): cuando no se especifica el atributo de descubrimiento o su valor sí lo está`false`, la CLI de AWS cifrado descifra el mensaje utilizando únicamente lo AWS KMS keys especificado en el atributo clave del `--wrapping-keys` parámetro. Si no especifica un atributo key cuando discovery es `false`, se produce un error en el comando de descifrado. Este valor es compatible con una práctica recomendada de CLI de AWS cifrado.
- `True`— Cuando el valor del atributo de descubrimiento es`true`, la CLI de AWS cifrado AWS KMS keys obtiene los metadatos del mensaje cifrado y los utiliza AWS KMS keys para descifrar el mensaje. El atributo de descubrimiento con un valor de `true` se comporta como las versiones de la CLI de AWS cifrado anteriores a la versión 1.8. x que no permitía especificar una clave de empaquetado al descifrar. Sin embargo, su intención de utilizarla AWS KMS key es explícita. Si especifica un atributo key cuando discovery es `true`, se produce un error en el comando de descifrado.

  El `true` valor puede provocar que la CLI de AWS cifrado se utilice AWS KMS keys en diferentes Cuentas de AWS regiones o que intente utilizar AWS KMS keys algo que el usuario no esté autorizado a utilizar.

Cuando lo es el descubrimiento`true`, se recomienda utilizar los atributos discovery-partition y discovery-account para limitar el AWS KMS keys uso a los que especifiques. Cuentas de AWS

discovery-account

Limita los que AWS KMS keys se utilizan para descifrar a los especificados. Cuenta de AWS El único valor válido para este atributo es un identificador de Cuenta de AWS.

Este atributo es opcional y válido solo en los comandos de descifrado en los AWS KMS keys que el atributo de descubrimiento esté establecido en `true` y el atributo de partición de descubrimiento esté especificado.

Cada atributo de cuenta de descubrimiento solo tiene un Cuenta de AWS identificador, pero puede especificar varios atributos de cuenta de descubrimiento en el mismo parámetro. `--wrapping-keys` Todas las cuentas especificadas en un parámetro `--wrapping-keys` determinado deben estar en la partición AWS especificada.

discovery-partition

Especifica la AWS partición de las cuentas en el atributo discovery-account. Su valor debe ser una AWS partición, como `awsaws-cn`, o. `aws-gov-cloud` Para obtener información, consulte Nombres de recurso de Amazon en el Referencia general de AWS.

Este atributo es obligatorio cuando utiliza el atributo discovery-account. Solo puede especificar un atributo de discovery-partition en cada parámetro `--wrapping keys`. Para especificar Cuentas de AWS en varias particiones, utilice un `--wrapping-keys` parámetro adicional.

proveedor

Identifica el proveedor de claves maestras. Se expresa en formato de par provider=ID. El valor predeterminado, aws-kms, representa. AWS KMS Este atributo solo es necesario cuando el proveedor de la clave maestra no lo es. AWS KMS

region

Identifica el Región de AWS de un AWS KMS key. Este atributo solo es válido para AWS KMS keys. Se utiliza solo cuando el identificador key no especifica una región; de lo contrario, se omite. Cuando se usa, anula la región predeterminada en el perfil con nombre de AWS CLI.

profile

Identifica un perfil AWS CLI con nombre. Este atributo solo es válido para AWS KMS keys. La región del perfil se utiliza solamente cuando el identificador key no especifica ninguna región y no hay un atributo region en el comando.

--input (-i)

Especifica la ubicación de los datos que se van a cifrar o descifrar. Este parámetro es obligatorio. El valor puede ser una ruta a un archivo o directorio o un patrón de nombres de archivo. Si va a canalizar la entrada al comando (stdin), utilice `-`.

Si la entrada no existe, el comando se completa correctamente sin generar ningún error o advertencia.

--recursive (-r, -R)

Realiza la operación en los archivos del directorio de entrada y sus subdirectorios. Este parámetro es obligatorio cuando el valor de `--input` es un directorio.

--decode

Decodifica la entrada codificada en Base64.

Si va a descifrar un mensaje que se ha cifrado y, a continuación, codificado, debe descodificarlo antes de descifrarlo. Este parámetro se encarga de realizar esta tarea.

Por ejemplo, si ha utilizado el parámetro `--encode` en un comando de cifrado, utilice el parámetro `--decode` en el comando de descifrado correspondiente. También puede utilizar este parámetro para descodificar la entrada codificada en Base64 antes de cifrarla.

--output (-o)

Especifica un destino para el resultado. Este parámetro es obligatorio. El valor puede ser un nombre de archivo, un directorio existente o bien `-`, que escribe el resultado en la línea de comandos (stdout).

Si el directorio de salida especificado no existe, el comando genera un error. Si la entrada contiene subdirectorios, la CLI de AWS cifrado reproduce los subdirectorios en el directorio de salida que especifique.

De forma predeterminada, la CLI de AWS cifrado sobrescribe los archivos con el mismo nombre. Para cambiar ese comportamiento, utilice los parámetros `--interactive` o `--no-overwrite`. Para suprimir la advertencia de sobrescritura, utilice el parámetro `--quiet`.

> **Note**
>
> Si un comando que sobrescribiría un archivo no se ejecuta correctamente, el archivo de salida se elimina.

--interactive

Pregunta antes de sobrescribir el archivo.

--no-overwrite

No sobrescribe los archivos. En cambio, si el archivo de salida existe, la CLI de AWS cifrado omite la entrada correspondiente.

--suffix

Especifica un sufijo de nombre de archivo personalizado para los archivos que crea la CLI de AWS cifrado. Para indicar que no se use ningún sufijo, utilice el parámetro sin ningún valor (`--suffix`).

De forma predeterminada, cuando el parámetro `--output` no especifica un nombre de archivo, el nombre del archivo de salida tiene el mismo nombre que el archivo de entrada, más el sufijo. El sufijo para los comandos de cifrado es `.encrypted`. El sufijo para los comandos de descifrado es `.decrypted`.

--encode

Aplica al resultado la codificación en Base64 (de binario a texto). La codificación impide que el programa del host del shell interprete incorrectamente los caracteres no ASCII del texto de salida.

Utilice este parámetro cuando escriba la salida cifrada en stdout (`--output -`), especialmente en una PowerShell consola, incluso cuando esté canalizando la salida a otro comando o guardándola en una variable.

--metadata-output

Especifica una ubicación para los metadatos relativos a las operaciones criptográficas. Escriba la ruta y el nombre de archivo. Si el directorio no existe, el comando genera un error. Para escribir los metadatos en la línea de comandos (stdout), utilice `-`.

No se puede escribir en stdout el resultado del comando (`--output`) y los metadatos de salida (`--metadata-output`) en el mismo comando. Además, cuando el valor de `--input` o `--output` es un directorio (sin los nombres de archivo), no puede escribir la salida de metadatos en el mismo directorio ni en ningún subdirectorio de ese directorio.

Si especifica un archivo existente, de forma predeterminada, la CLI de AWS cifrado anexa nuevos registros de metadatos a cualquier contenido del archivo. Esta característica le permite crear

un único archivo que contiene los metadatos de todas las operaciones criptográficas. Para sobrescribir el contenido en un archivo existente, utilice el parámetro `--overwrite-metadata`.

La CLI de AWS cifrado devuelve un registro de metadatos con formato JSON para cada operación de cifrado o descifrado que realice el comando. Cada registro de metadatos incluye las rutas completas a los archivos de entrada y salida, el contexto de cifrado, el conjunto de algoritmos y otra información valiosa que puede utilizar para revisar la operación y verificar que cumpla los estándares de seguridad.

--overwrite-metadata

Sobrescribe el contenido en el archivo de salida de metadatos. De forma predeterminada, el parámetro `--metadata-output` adjunta los metadatos a todo el contenido existente del archivo.

--suppress-metadata (-S)

Suprime los metadatos relativos a la operación de cifrado y descifrado.

--commitment-policy

Especifica la política de compromiso para los comandos de cifrado y descifrado. La política de compromiso determina si su mensaje está cifrado o descifrado con la característica de seguridad de compromiso clave.

El parámetro `--commitment-policy` se introduce en la versión 1.8.x. Es válido en los comandos de cifrado y descifrado.

En la versión 1.8. x, la CLI de AWS cifrado utiliza la política de `forbid-encrypt-allow-decrypt` compromiso para todas las operaciones de cifrado y descifrado. Cuando se utiliza el parámetro `--wrapping-keys` en un comando de cifrado o descifrado, se requiere un parámetro `--commitment-policy` con el valor `forbid-encrypt-allow-decrypt`. Si no utiliza el parámetro `--wrapping-keys`, el parámetro `--commitment-policy` no es válido. Establecer este valor de forma explícita evita que la política de compromiso se modifique automáticamente a `require-encrypt-require-decrypt` cuando sube de categoría a la versión 2.1.x.

A partir de la versión 2.1.x, se admiten todos los valores de la política de compromiso. El parámetro `--commitment-policy` es opcional y el valor predeterminado es `require-encrypt-require-decrypt`.

Este parámetro tiene los siguientes valores:

- `forbid-encrypt-allow-decrypt`: no se puede cifrar con un compromiso clave. Puede descifrar textos cifrados, cifrados con o sin compromiso clave.

  En la versión 1.8.x, este es el único valor válido. La CLI de AWS cifrado utiliza la política de `forbid-encrypt-allow-decrypt` compromiso para todas las operaciones de cifrado y descifrado.
- `require-encrypt-allow-decrypt`: cifra solo con un compromiso clave. Descifra con y sin compromiso clave. Este valor se introduce en la versión 2.1.x.
- `require-encrypt-require-decrypt` (predeterminado): cifra y descifra solo con el compromiso clave. Este valor se introduce en la versión 2.1.x. Es el valor predeterminado en las versiones 2.1.x y posteriores. Con este valor, la CLI de AWS cifrado no descifrará ningún texto cifrado que se haya cifrado con versiones anteriores de. AWS Encryption SDK

Para obtener información detallada sobre cómo establecer su política de compromiso, consulte Migrando su AWS Encryption SDK.

--encryption-context (-c)

Especifica un contexto de cifrado para la operación. Este parámetro no es obligatorio, pero se recomienda.

- En un comando `--encrypt`, escriba uno o más pares `name=value`. Utilice espacios para separar los pares.
- En un comando `--decrypt`, ingrese pares de `name=value`, elementos de `name` sin valores o ambos.

Si `name` o `value` de un par `name=value` incluye espacios o caracteres especiales, incluya el par completo entre comillas. Por ejemplo, `--encryption-context "department=software development"`.

--buffer (-b) [Introducido en la versión 1.9.x y 2.2.x]

Devuelve el texto no cifrado solo después de procesar todas las entradas, incluida la verificación de la firma digital, si existe alguna.

-- max-encrypted-data-keys [Introducido en la versión 1.9. x y 2.2. x]

Especifica el número máximo de claves de datos cifrados en un mensaje cifrado. Este parámetro es opcional.

Los valores válidos son de 1 a 65 535. Si omite este parámetro, la CLI de AWS cifrado no aplica ningún máximo. Un mensaje cifrado puede contener hasta 65 535 (2^16 - 1) claves de datos cifrados.

Puede usar este parámetro en los comandos de cifrado para evitar un mensaje con un formato incorrecto. Puede usarlo en los comandos de descifrado para detectar mensajes malintencionados y evitar el descifrado de mensajes con numerosas claves de datos cifradas que no puede descifrar. Para obtener información detallada y un ejemplo, consulte Limitar las claves de datos cifrados.

--help (-h)

Imprime la sintaxis y la utilización en la línea de comandos.

--version

Obtiene la versión de la CLI de AWS cifrado.

-v | -vv | -vvv | -vvvv

Muestra información detallada, advertencias y mensajes de depuración. Los detalles del resultado aumentan en función del número de veces que se repita la `v` en el parámetro. La configuración más detallada (`-vvvv`) devuelve datos de nivel de depuración de la AWS CLI de cifrado y de todos los componentes que utiliza.

--quiet (-q)

Suprime mensajes de advertencia, como el mensaje que aparece al sobrescribir un archivo de salida.

--master-keys (-m) [Obsoleto]

> **Note**
>
> El parámetro --master-keys está obsoleto en la versión 1.8.x y se eliminó en la versión 2.1.x. En su lugar, utilice el parámetro --wrapping-keys.

Especifica las claves maestras utilizadas en las operaciones de cifrado y descifrado. Puede utilizar varios parámetros de clave maestra en cada comando.

El parámetro `--master-keys` es obligatorio en los comandos de cifrado. Es obligatorio en los comandos de descifrado solo cuando utiliza un proveedor de claves maestras (no AWS KMS) personalizadas.

Atributos: el valor del parámetro `--master-keys` se compone de los siguientes atributos. El formato es `attribute_name=value`.

clave

Identifica la clave de encapsulamiento utilizada en la operación. Se expresa en formato de par key=ID. El atributo key es obligatorio en todos los comandos de cifrado.

Cuando se utiliza un comando AWS KMS key in a encrypt, el valor del atributo clave puede ser un identificador de clave, un ARN de clave, un nombre de alias o un ARN de alias. Para obtener más información sobre los identificadores de AWS KMS clave, consulte los identificadores de clave en la Guía para desarrolladores.AWS Key Management Service

El atributo key es obligatorio en los comandos de descifrado cuando el proveedor de claves maestras no es AWS KMS. El atributo key no se permite en los comandos que descifran datos que se han cifrado en un AWS KMS key.

Puede especificar varios atributos key en cada valor del parámetro `--master-keys`. Sin embargo, todos los atributos provider, region y profile se aplican a todas las claves maestras del valor del parámetro. Para especificar claves maestras con valores de atributos diferentes, utilice varios parámetros `--master-keys` en el comando.

proveedor

Identifica el proveedor de claves maestras. Se expresa en formato de par provider=ID. El valor predeterminado, aws-kms, representa. AWS KMS Este atributo solo es necesario cuando el proveedor de la clave maestra no lo es. AWS KMS

region

Identifica el Región de AWS de un AWS KMS key. Este atributo solo es válido para AWS KMS keys. Se utiliza solo cuando el identificador key no especifica una región; de lo contrario, se omite. Cuando se usa, anula la región predeterminada en el perfil con nombre de AWS CLI.

profile

Identifica un perfil AWS CLI con nombre. Este atributo solo es válido para AWS KMS keys. La región del perfil se utiliza solamente cuando el identificador key no especifica ninguna región y no hay un atributo region en el comando.

Parámetros avanzados

--algorithm

Especifica un conjunto de algoritmos alternativo. Este parámetro es opcional y solamente es válido en los comandos de cifrado.

Si omite este parámetro, la CLI de AWS cifrado utiliza uno de los conjuntos de algoritmos predeterminados para los AWS Encryption SDK introducidos en la versión 1.8. x. Ambos algoritmos predeterminados utilizan AES-GCM con un HKDF, una firma ECDSA y una clave de cifrado de 256 bits. Uno usa el compromiso clave; el otro, no. La elección del conjunto de algoritmos predeterminado viene determinada por la política de compromiso del comando.

Los conjuntos de algoritmos se recomiendan para la mayoría de las operaciones de cifrado. Para obtener una lista de valores válidos, consulte los valores para el parámetro `algorithm` en Leer los documentos.

--frame-length

Crea un resultado con la longitud de trama especificada. Este parámetro es opcional y solamente es válido en los comandos de cifrado.

Especifique el valor en bytes. Los valores válidos son 0 y 1 – 2^31 - 1. Un valor de 0 indica datos sin trama. El valor predeterminado es 4096 (bytes).

> **Note**
>
> Siempre que sea posible, utilice datos con trama. AWS Encryption SDK Admite datos no enmarcados solo para uso heredado. Algunas implementaciones lingüísticas del aún AWS Encryption SDK pueden generar texto cifrado sin marcos. Todas las implementaciones de idiomas compatibles pueden descifrar texto cifrado con trama y sin trama.

--max-length

Indica el tamaño máximo en bytes de la trama (o la longitud máxima del contenido para los mensajes sin trama) que se ha de leer en los mensajes cifrados. Este parámetro es opcional y solamente es válido en los comandos de descifrado. Se ha diseñado para protegerle contra el descifrado de texto cifrado malintencionado de tamaño excesivamente grande.

Especifique el valor en bytes. Si omite este parámetro, AWS Encryption SDK no limita el tamaño del marco al descifrar.

--caching

Habilita la característica de almacenamiento en caché de claves de datos, que reutiliza las claves de datos en lugar de generar una nueva para cada archivo de entrada. Este parámetro admite una situación avanzada. Asegúrese de leer la documentación Almacenamiento en caché de claves de datos antes de usar esta característica.

El parámetro `--caching` tiene los siguientes atributos.

capacity (obligatorio)

Determina el número máximo de entradas de la caché.

El valor mínimo es 1. No hay ningún valor máximo.

max_age (obligatorio)

Determina durante cuánto tiempo se usarán las entradas de la caché, en segundos, a partir del momento en que se agregaron a la caché.

Escriba un valor mayor que 0. No hay ningún valor máximo.

max_messages_encrypted (opcional)

Determina el número máximo de mensajes que se pueden cifrar con una misma entrada almacenada en caché.

Los valores válidos son 1 – 2^32. El valor predeterminado es 2^32 (mensajes).

max_bytes_encrypted (opcional)

Determina el número máximo de bytes se pueden cifrar con una misma entrada almacenada en caché.

Los valores válidos son 0 y 1 – 2^63 - 1. El valor predeterminado es 2^63 - 1 (mensajes). El valor 0 permite utilizar almacenamiento en caché de clave de datos solo al cifrar cadenas de mensaje vacías.

Versiones de la CLI AWS de cifrado

Le recomendamos que utilice la última versión de la CLI de AWS cifrado.

> **Note**
>
> Las versiones de la CLI de AWS cifrado anteriores a la 4.0.0 están en end-of-supportfase. Puede actualizar de forma segura desde la versión 2.1.x y versiones posteriores a la última versión de la CLI de cifrado de AWS sin cambios en el código ni en los datos. Sin embargo, se introdujeron nuevas características de seguridad en la versión 2.1.x que no son compatibles con versiones anteriores. Para actualizar desde la versión 1.7. x o anterior, primero debe actualizar a la última versión 1. versión x de la CLI AWS de cifrado. Para obtener más información, consulte Migrando su AWS Encryption SDK.
> Las nuevas funciones de seguridad se publicaron originalmente en las versiones 1.7 de AWS Encryption CLI. x y 2.0. x. Sin embargo, AWS Encryption CLI versión 1.8. x reemplaza a la versión 1.7. x y CLI de AWS cifrado 2.1. x reemplaza a 2.0. x. Para obtener más información, consulte el aviso de seguridad correspondiente en el aws-encryption-sdk-clirepositorio de GitHub.

Para obtener información sobre las versiones más importantes del AWS Encryption SDK, consulteVersiones del AWS Encryption SDK.

¿Qué versión utilizo?

Si es la primera vez que utiliza la CLI de AWS cifrado, utilice la versión más reciente.

Para descifrar los datos cifrados por una versión AWS Encryption SDK anterior a la 1.7. x, migre primero a la última versión de la CLI de AWS cifrado. Realice todos los cambios recomendados antes de subir de categoría a la versión 2.1.x o posterior. Para obtener más información, consulte Migrando su AWS Encryption SDK.

Más información

- Para obtener información detallada sobre los cambios y directrices para migrar a estas nuevas versiones, consulte Migrando su AWS Encryption SDK.
- Para obtener descripciones de los nuevos parámetros y atributos de la CLI de AWS cifrado, consulteAWS Encryption SDK Referencia de parámetros y sintaxis de CLI.

Las siguientes listas describen el cambio en la CLI de AWS cifrado en las versiones 1.8. x y 2.1. x.

Versión 1.8. x cambios en la CLI AWS de cifrado

- Hace obsoleto el parámetro `--master-keys`. En su lugar, utilice el parámetro `--wrapping-keys`.
- Añade el parámetro `--wrapping-keys` (`-w`). Es compatible con todos los atributos del parámetro `--master-keys`. También añade los siguientes atributos opcionales, que solo son válidos cuando se descifra con AWS KMS keys.
  - discovery
  - discovery-partition
  - discovery-account

  Para los proveedores de claves maestras personalizadas, los comandos `--encrypt` y `--decrypt` requieren un parámetro `--wrapping-keys` o un parámetro `--master-keys` (pero no ambos). Además, un `--encrypt` comando AWS KMS keys que requiera un `--wrapping-keys` parámetro o un `--master-keys` parámetro (pero no ambos).

  En un `--decrypt` comando con AWS KMS keys, el `--wrapping-keys` parámetro es opcional, pero se recomienda porque es obligatorio en la versión 2.1. x. Si lo usa, debe especificar el atributo key o el atributo discovery con un valor de `true` (pero no ambos).
- Añade el parámetro `--commitment-policy`. El único valor válido es `forbid-encrypt-allow-decrypt`. La política de compromiso de `forbid-encrypt-allow-decrypt` se utiliza en todos los comandos de cifrado y descifrado.

  En la versión 1.8.x, cuando utiliza el parámetro `--wrapping-keys`, se requiere un parámetro `--commitment-policy` con el valor `forbid-encrypt-allow-decrypt`. Si se establece este valor de forma explícita, se evita que la política de compromiso cambie automáticamente a `require-encrypt-require-decrypt` cuando sube de categoría a la versión 2.1.x.

Versión 2.1. x cambios en la CLI AWS de cifrado

- Elimina el parámetro `--master-keys`. En su lugar, utilice el parámetro `--wrapping-keys`.
- El parámetro `--wrapping-keys` es obligatorio en todos los comandos de cifrado y descifrado. Debe especificar un atributo key o un atributo discovery con un valor de `true` (pero no ambos).
- El parámetro `--commitment-policy` admite los siguientes valores. Para obtener más información, consulte Establecer su política de compromiso.
  - `forbid-encrypt-allow-decrypt`

  - `require-encrypt-allow-decrypt`
  - `require-encrypt-require decrypt` (predeterminado)
- El parámetro `--commitment-policy` es opcional en la versión 2.1.x. El valor predeterminado es `require-encrypt-require-decrypt`.

Versión 1.9. x y 2.2. x cambios en la CLI AWS de cifrado

- Añade el parámetro `--decrypt-unsigned`. Para obtener más información, consulte Versión 2.2.x.
- Añade el parámetro `--buffer`. Para obtener más información, consulte Versión 2.2.x.
- Añade el parámetro `--max-encrypted-data-keys`. Para obtener más información, consulte Limitar las claves de datos cifrados.

Versión 3.0. x cambios en la CLI AWS de cifrado

- Añade compatibilidad con claves AWS KMS multirregionales. Para obtener información, consulte Uso de varias regiones AWS KMS keys.

Almacenamiento en caché de claves de datos

Con el almacenamiento en caché de claves de datos, se guardan claves de datos y sus materiales criptográficos relacionados en una caché. Al cifrar o descifrar datos, AWS Encryption SDK busca una clave de datos coincidente en la memoria caché. Si encuentra una coincidencia, utiliza la clave de datos almacenada en caché en lugar de generar una nueva. El almacenamiento en caché de claves de datos puede mejorar el rendimiento, reducir los costos y ayudarle a mantenerse dentro de los límites de servicio, aunque cambie la escala de la aplicación.

La aplicación puede beneficiarse del almacenamiento en caché de claves de datos si:

- puede reutilizar claves de datos;
- genera numerosas claves de datos;
- las operaciones criptográficas son inaceptablemente lentas, costosas o limitadas o realizan un uso intensivo de los recursos.

El almacenamiento en caché puede reducir el uso de servicios criptográficos, como (). AWS Key Management Service AWS KMS Si estás alcanzando tu AWS KMS requests-per-secondlímite, el almacenamiento en caché puede ayudarte. Tu aplicación puede usar claves almacenadas en caché para atender algunas de tus solicitudes de claves de datos en lugar de realizar llamadas. AWS KMS(También puede crear un caso en el AWS Centro de asistencia para aumentar el límite de su cuenta).

AWS Encryption SDK Esto le ayuda a crear y administrar su caché de claves de datos. Proporciona una caché local y un administrador de materiales criptográficos de almacenamiento (CMM de almacenamiento en caché) que interactúa con la caché y aplica los umbrales de seguridad establecidos. Juntos, estos componentes le ayudan a beneficiarse de la eficacia de reutilizar las claves de datos y, al mismo tiempo, mantiene la seguridad del sistema.

El almacenamiento en caché de las claves de datos es una función opcional AWS Encryption SDK que debe utilizar con precaución. De forma predeterminada, AWS Encryption SDK genera una nueva clave de datos para cada operación de cifrado. Esta técnica es compatible con las prácticas recomendadas criptográficas, que desaconsejan una reutilización excesiva de las claves de datos. En general, conviene utilizar el almacenamiento en caché de claves de datos exclusivamente cuando sea imprescindible para satisfacer sus objetivos de rendimiento. A continuación, utilice umbrales de seguridad del almacenamiento en caché de claves de datos para asegurarse de usar la cantidad mínima de almacenamiento en caché requerida para satisfacer sus objetivos de costo y rendimiento.

Versión 3. x SDK de cifrado de AWS para Java solo admite la CMM de almacenamiento en caché con la interfaz antigua de proveedores de claves maestras, no con la interfaz de anillo de claves. Sin embargo, la versión 4. x del AWS Encryption SDK para .NET, versión 3. x de SDK de cifrado de AWS para Java, versión 4. x de la SDK de cifrado de AWS para Python, versión 1. x AWS Encryption SDK para Rust y la versión 0.1. x o versiones posteriores de AWS Encryption SDK for Go admiten el llavero AWS KMS jerárquico, una solución alternativa de almacenamiento en caché de materiales criptográficos. El contenido cifrado con el anillo de claves AWS KMS jerárquico solo se puede descifrar con el anillo de claves jerárquico. AWS KMS

Para consultar una descripción detallada de lo que esto conlleva para la seguridad, consulte AWS Encryption SDK: cómo decidir si el almacenamiento en caché de las claves de datos es adecuado para su aplicación en el blog sobre seguridad de AWS .

Temas

- Cómo utilizar el almacenamiento en caché de claves de datos
- Configuración de los umbrales de seguridad de la caché
- Información detallada sobre el almacenamiento en caché de claves de datos
- Ejemplo de almacenamiento en caché de claves de datos

Cómo utilizar el almacenamiento en caché de claves de datos

En este tema se muestra cómo usar el almacenamiento en caché de claves de datos en las aplicaciones. Recorremos el proceso paso a paso. A continuación, combinaremos todo el procedimiento en un ejemplo sencillo que utiliza el almacenamiento en caché de claves de datos en una operación para cifrar una cadena.

Los ejemplos en esta sección muestran cómo usar la versión 2.0.x y posterior del AWS Encryption SDK. Para ver ejemplos que utilizan versiones anteriores, busca tu versión en la lista de versiones del GitHub repositorio de tu lenguaje de programación.

Para ver ejemplos completos y comprobados del uso del almacenamiento en caché de claves de datos en el AWS Encryption SDK, consulta:

- C/C++: caching_cmm.cpp
- Java: SimpleDataKeyCachingExample .java
- JavaScript Navegador: caching_cmm.ts
- JavaScript Node.js: caching_cmm.ts

- Python: data_key_caching_basic.py

La AWS Encryption SDK para .NET no admite el almacenamiento en caché de claves de datos.

Temas

- Uso del almacenamiento en caché de claves de datos: Step-by-step
- Ejemplo de almacenamiento en caché de claves de datos: cifrado de una cadena

Uso del almacenamiento en caché de claves de datos: Step-by-step

Estas step-by-step instrucciones le muestran cómo crear los componentes que necesita para implementar el almacenamiento en caché de claves de datos.

- Crear una caché de clave de datos. En estos ejemplos, utilizamos la caché local que AWS Encryption SDK proporciona. Limitamos la caché a 10 claves de datos.

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

En el siguiente ejemplo se utiliza la versión 2. x del SDK de cifrado de AWS para Java. Versión 3. x del SDK de cifrado de AWS para Java CMM de almacenamiento en caché de claves de datos está en desuso. Con la versión 3. x, también puede utilizar el llavero AWS KMS jerárquico, una solución alternativa de almacenamiento en caché de materiales criptográficos.

```
// 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)
```

- Cree un proveedor de claves maestras (Java y Python) o un anillo de claves (C y JavaScript). En estos ejemplos se utiliza un proveedor de claves maestras AWS Key Management Service (AWS KMS) o un conjunto de AWS KMS claves compatible.

  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

  En el siguiente ejemplo se utiliza la versión 2. x del SDK de cifrado de AWS para Java. Versión 3. x del SDK de cifrado de AWS para Java CMM de almacenamiento en caché de claves

de datos está en desuso. Con la versión 3. x, también puede utilizar el llavero AWS KMS jerárquico, una solución alternativa de almacenamiento en caché de materiales criptográficos.

```
// 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

En el navegador, debe insertar las credenciales de forma segura. En este ejemplo se definen las credenciales de un paquete web (kms.webpack.config) que resuelve las credenciales en tiempo de ejecución. Crea una instancia de proveedor de AWS KMS clientes a partir de un AWS KMS cliente y las credenciales. Luego, cuando crea el anillo de claves, pasa el proveedor del cliente al constructor junto con el 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])
```

- Crear un administrador de materiales criptográficos de almacenamiento (CMM de almacenamiento en caché).

  Asocie el CMM de almacenamiento en caché con la caché y con el proveedor de claves maestras o conjunto de claves. A continuación, establezca los umbrales de seguridad de la caché en el CMM de almacenamiento en caché.

  C

  En el SDK de cifrado de AWS para C, puede crear una CMM de almacenamiento en caché a partir de una CMM subyacente, como la CMM predeterminada, o desde un anillo de claves. En este ejemplo se crea el CMM de almacenamiento en caché a partir de un conjunto de claves.

  Después de crear el CMM de almacenamiento en caché, puede liberar sus referencias al conjunto de claves y a la caché. Para obtener más información, consulte the section called "Recuento de referencias".

  ```
  // 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

En el siguiente ejemplo, se utiliza la versión 2. x del SDK de cifrado de AWS para Java. Versión 3. x of the SDK de cifrado de AWS para Java no admite el almacenamiento en caché de claves de datos, pero sí es compatible con el anillo de claves AWS KMS jerárquico, una solución alternativa de almacenamiento en caché de materiales criptográficos.

```
/*
 * 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
)
```

Es todo lo que tiene que hacer. A continuación, deje que AWS Encryption SDK administren la memoria caché por usted o añada su propia lógica de administración de la memoria caché.

Cuando desee utilizar el almacenamiento de claves de datos en una llamada para cifrar o descifrar datos, especifique el CMM de almacenamiento en caché en lugar del proveedor de claves maestras u otro CMM.

> **Note**
>
> Si va a cifrar flujos de datos o datos de tamaño desconocido, asegúrese de especificar el tamaño de los datos en la solicitud. AWS Encryption SDK No utiliza el almacenamiento en caché de claves de datos al cifrar datos de tamaño desconocido.

C

En el SDK de cifrado de AWS para C, se crea una sesión con la CMM de almacenamiento en caché y, a continuación, se procesa la sesión.

De forma predeterminada, cuando el tamaño del mensaje es desconocido e ilimitado, no almacena en caché las claves de AWS Encryption SDK datos. Para permitir el almacenamiento en caché cuando no se conoce el tamaño exacto de los datos, utilice el método `aws_cryptosdk_session_set_message_bound` para establecer un tamaño máximo para el mensaje. Defina el límite en un tamaño mayor que el tamaño estimado del mensaje. Si el tamaño real del mensaje supera el límite, la operación de cifrado genera un error.

```
/* 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

En el siguiente ejemplo, se utiliza la versión 2. x del SDK de cifrado de AWS para Java. Versión 3. x del SDK de cifrado de AWS para Java CMM de almacenamiento en caché de claves de datos está en desuso. Con la versión 3. x, también puede utilizar el llavero AWS KMS jerárquico, una solución alternativa de almacenamiento en caché de materiales criptográficos.

```
// 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

Si utiliza el CMM de almacenamiento en caché SDK de cifrado de AWS para JavaScript para Node.js, el `encrypt` método requiere la longitud del texto sin formato. Si no la proporciona, la clave de datos no se almacena en caché. Si proporciona una longitud, pero los datos en texto no cifrado que proporciona superan esa longitud, se producirá un error en la operación de cifrado. Si no sabe la longitud exacta del texto no cifrado, como cuando transmite datos, proporcione el valor esperado más grande.

```
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
```

```
)
```

Ejemplo de almacenamiento en caché de claves de datos: cifrado de una cadena

Este ejemplo de código sencillo utiliza el almacenamiento en caché de claves de datos al cifrar una cadena. Combina el código del step-by-step procedimiento en un código de prueba que se puede ejecutar.

El ejemplo crea una caché local y un proveedor de claves maestras o un conjunto de claves para un AWS KMS key. A continuación, se utiliza la caché local y un proveedor de claves maestras o conjunto de claves para crear un CMM de almacenamiento en caché con los umbrales de seguridad adecuados. En Java y Python, la solicitud de cifrado especifica el CMM de almacenamiento en caché, los datos en texto no cifrado que hay que cifrar y un contexto de cifrado. En C, el CMM de almacenamiento en caché se especifica en la sesión y la sesión se proporciona a la solicitud de cifrado.

Para ejecutar estos ejemplos, debe proporcionar el Nombre de recurso de Amazon (ARN) de un AWS KMS key. Compruebe de tenga permiso para utilizar la AWS KMS key para generar una clave de datos.

Para obtener ejemplos reales más detallados de cómo crear y usar una caché de claves de datos, consulte Ejemplo de almacenamiento en caché de claves de datos.

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

En el siguiente ejemplo se utiliza la versión 2. x del SDK de cifrado de AWS para Java. Versión 3. x del SDK de cifrado de AWS para Java CMM de almacenamiento en caché de claves de datos está en desuso. Con la versión 3. x, también puede utilizar el llavero AWS KMS jerárquico, una solución alternativa de almacenamiento en caché de materiales criptográficos.

```
// 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.kmssdkv2.KmsMasterKey;
import com.amazonaws.encryptionsdk.kmssdkv2.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 testCachingCMMExample() {
  /* 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 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 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
```

Configuración de los umbrales de seguridad de la caché

Al implementar el almacenamiento en caché de claves de datos, es preciso configurar los umbrales de seguridad que el CMM de almacenamiento en caché debe aplicar.

Los umbrales de seguridad ayudan a limitar el tiempo durante el cual se utiliza cada clave de datos almacenada en caché, así como la cantidad de datos que se protegen con cada una de ellas. El CMM de almacenamiento en caché devuelve claves de datos almacenadas en caché solo cuando la entrada se ajusta a todos los umbrales de seguridad. Si la entrada de la caché supera cualquier umbral, no se utilizará para la operación actual y se expulsará de la caché lo antes posible. El primer uso de cada clave de datos (antes de almacenarla en caché) no se cuenta al calcular estos umbrales.

Por regla general, utilice la cantidad mínima de almacenamiento en caché que se requiera para satisfacer sus objetivos de costo y rendimiento.

El AWS Encryption SDK único que almacena en caché las claves de datos que se cifran mediante una función de derivación de claves. Además, establece límites máximos para algunos de los valores de los umbrales. Estas restricciones garantizan que las claves de datos no se reutilicen más allá de sus límites criptográficos. Sin embargo, dado que las claves de datos en texto no cifrado se almacenan en caché (en memoria, de forma predeterminada), es importante intentar minimizar el tiempo durante el que se conservan. Además, es importante tratar de limitar los datos que podrían verse expuestas en caso de filtrarse una clave.

Para ver ejemplos de cómo configurar los umbrales de seguridad de la memoria caché, consulte AWS Encryption SDK: Cómo decidir si el almacenamiento en caché de claves de datos es adecuado para su aplicación en el blog de seguridad. AWS

> **Note**
>
> El CMM de almacenamiento en caché aplica todos los umbrales siguientes. Si no se especifica un valor opcional, el CMM de almacenamiento en caché utiliza el predeterminado. Para deshabilitar el almacenamiento en caché de claves de datos temporalmente, las implementaciones de Java y Python del AWS Encryption SDK proporcionan una caché de materiales criptográficos nula (caché nula). La caché nula devuelve un error por cada solicitud `GET` y no responde a las solicitudes `PUT`. Le recomendamos que utilice la caché nula en lugar de establecer la capacidad de caché o los umbrales de seguridad en 0. Para obtener más información, consulte la caché nula en Java y Python.

Antigüedad máxima (obligatoria)

Determina cuánto tiempo se puede utilizar una entrada almacenada en la caché, a partir del momento en que se agregó. Este valor es obligatorio. Escriba un valor mayor que 0. Esto AWS Encryption SDK no limita el valor máximo de edad.

Todas las implementaciones lingüísticas del AWS Encryption SDK definen la edad máxima en segundos, excepto el SDK de cifrado de AWS para JavaScript, que utiliza milisegundos.

Utilice el intervalo más breve que permita que la aplicación continúe beneficiándose de la caché. Puede utilizar el umbral de antigüedad máxima como política de rotación de claves. Utilícelo para limitar la reutilización de claves de datos, minimizar la exposición de materiales criptográficas y expulsar las claves de datos cuyas políticas podrían haber cambiado mientras estaban almacenadas en la caché.

Número máximo de mensajes cifrados (opcional)

Especifica el número máximo de mensajes que una clave de datos almacenada en caché puede cifrar. Este valor es opcional. Escriba un valor comprendido entre 1 y 2^32 mensajes. El valor predeterminado es 2^32 mensajes.

Establezca el número de mensajes protegidos por cada clave almacenada en caché de modo que sea lo bastante grande para obtener un valor reutilizado pero lo bastante pequeño para limitar el número de mensajes que podrían verse expuestos en caso de filtrarse una clave.

Número máximo de bytes cifrados (opcional)

Especifica el número máximo de bytes que una clave de datos almacenada en caché puede cifrar. Este valor es opcional. Escriba un valor comprendido entre 0 y 2^63 - 1. El valor

predeterminado es 2^63 - 1. El valor 0 permite utilizar almacenamiento en caché de clave de datos solo al cifrar cadenas de mensaje vacías.

Los bytes de la solicitud actual se incluyen al evaluar este umbral. Si la suma de bytes procesados más bytes actuales supera este umbral, la clave de datos almacenada en caché se expulsa de la caché, aunque se haya utilizado en una solicitud menor.

Información detallada sobre el almacenamiento en caché de claves de datos

La mayoría de las aplicaciones pueden utilizar la implementación predeterminada de almacenamiento en caché de claves de datos sin necesidad de escribir código personalizado. En esta sección se describen la implementación predeterminada y algunos detalles sobre las opciones.

Temas

- Funcionamiento del almacenamiento en caché de claves de datos
- Creación de una caché de materiales criptográficos
- Creación de un administrador de materiales criptográficos de almacenamiento en caché
- ¿Qué es una entrada en la caché de claves de datos?
- Contexto de cifrado: cómo seleccionar las entradas de la caché
- ¿Usa mi aplicación claves de datos almacenadas en caché?

Funcionamiento del almacenamiento en caché de claves de datos

Cuando se utiliza el almacenamiento en caché de claves de datos en una solicitud de cifrado o descifrado de datos, el AWS Encryption SDK busca primero en la caché si hay alguna clave de datos que coincida con la solicitud. Si encuentra una coincidencia válida, utiliza la clave de datos almacenada en caché para cifrar los datos. De lo contrario, genera una nueva clave de datos, tal y como lo haría sin la caché.

El almacenamiento en caché de claves de datos no se utiliza para datos de tamaño desconocido, como, por ejemplo, los datos transmitidos en streaming. Esto permite que el CMM de almacenamiento en caché aplique correctamente el umbral de bytes máximos. Para evitar este comportamiento, agregue el tamaño del mensaje a la solicitud de cifrado.

Además de una caché, el almacenamiento en caché de claves de datos utiliza un administrador de materiales criptográficos de almacenamiento (almacenamiento en caché CMM). El CMM de almacenamiento en caché es un administrador de materiales criptográficos (CMM) especializado que interactúa con una caché y con un CMM subyacente. (Cuando especifica un proveedor de claves maestras o un conjunto de claves, el AWS Encryption SDK le crea un CMM predeterminado). El CMM almacenamiento almacena en caché las claves de datos que el CMM subyacente devuelve. El CMM de almacenamiento en caché también aplica los umbrales de seguridad de caché establecidos.

Para evitar que se seleccione una clave de datos incorrecta de la caché, todo almacenamiento en caché compatible CMMs requiere que las siguientes propiedades de los materiales criptográficos almacenados en caché coincidan con la solicitud de materiales.

- Conjunto de algoritmos
- Contexto de cifrado (incluso cuando está vacío)
- Nombre de partición (una cadena que identifica el CMM de almacenamiento en caché)
- (Descifrado solo) Claves de datos cifradas

> **Note**
>
> Almacena en AWS Encryption SDK caché las claves de datos solo cuando el conjunto de algoritmos utiliza una función de derivación de claves.

Los siguientes flujos de trabajo muestran cómo se procesa una solicitud de cifrado de datos con y sin almacenamiento en caché de las claves de datos. Muestran cómo se utilizan en el proceso los componentes de almacenamiento en caché que crea, lo que incluye la caché y el CMM de almacenamiento en caché.

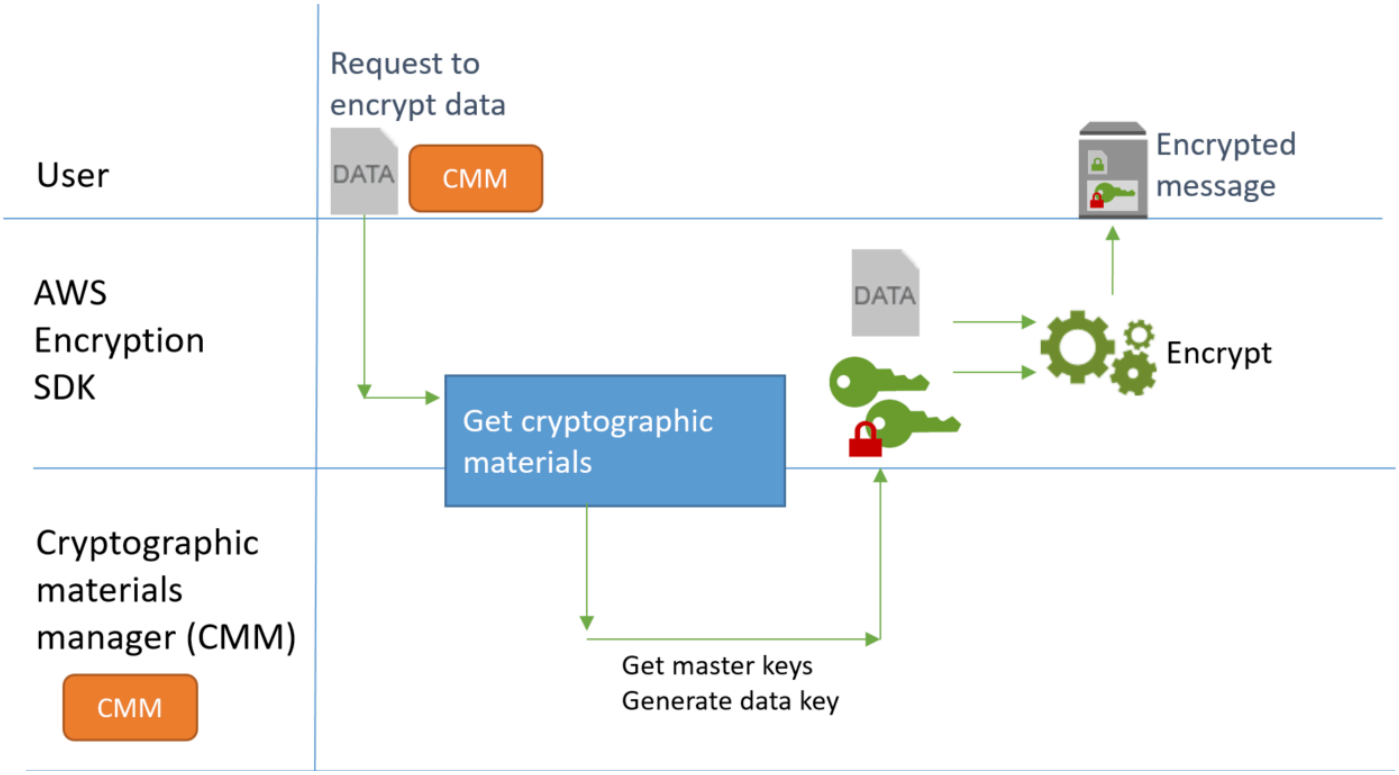
Cifrado de datos sin almacenamiento en caché

Para obtener materiales de cifrado sin almacenamiento en caché:

1. Una aplicación les pide que cifren AWS Encryption SDK los datos.

   La solicitud especifica un proveedor de claves maestras o un conjunto de claves. El AWS Encryption SDK crea un CMM predeterminado que interactúa con el proveedor de claves maestras o el conjunto de claves.
2. AWS Encryption SDK Solicita a la CMM materiales de cifrado (obtenga materiales criptográficos).

3. La CMM solicita materiales criptográficos a su conjunto de claves (C and JavaScript) o a su proveedor de claves maestras (Java y Python). Esto puede implicar una llamada a un servicio criptográfico, como (). AWS Key Management Service AWS KMS El CMM devuelve los materiales de cifrado al AWS Encryption SDK.
4. AWS Encryption SDK Utiliza la clave de datos de texto simple para cifrar los datos. Almacena los datos cifrados y las claves de datos cifrados en un mensaje cifrado, que devuelve al usuario.



Cifrado de datos con almacenamiento en caché

Para obtener materiales de cifrado con almacenamiento en caché de clave de datos:

1. Una aplicación les pide que cifren AWS Encryption SDK los datos.

   La solicitud especifica un administrador de materiales criptográficos de almacenamiento (CMM de almacenamiento en caché) que está asociado a un administrador de materiales criptográficos (CMM) subyacente. Cuando especifica un proveedor de claves maestras o un conjunto de claves, el AWS Encryption SDK le crea un CMM predeterminado.
2. El SDK solicita los materiales de cifrado al CMM de almacenamiento en caché especificado.
3. El CMM de almacenamiento en caché solicita materiales de cifrado de la caché.

a. Si la caché encuentra una coincidencia, actualiza los valores de antigüedad y uso de la entrada de caché coincidente y devuelve los materiales de cifrado almacenados en caché al CMM de almacenamiento en caché.

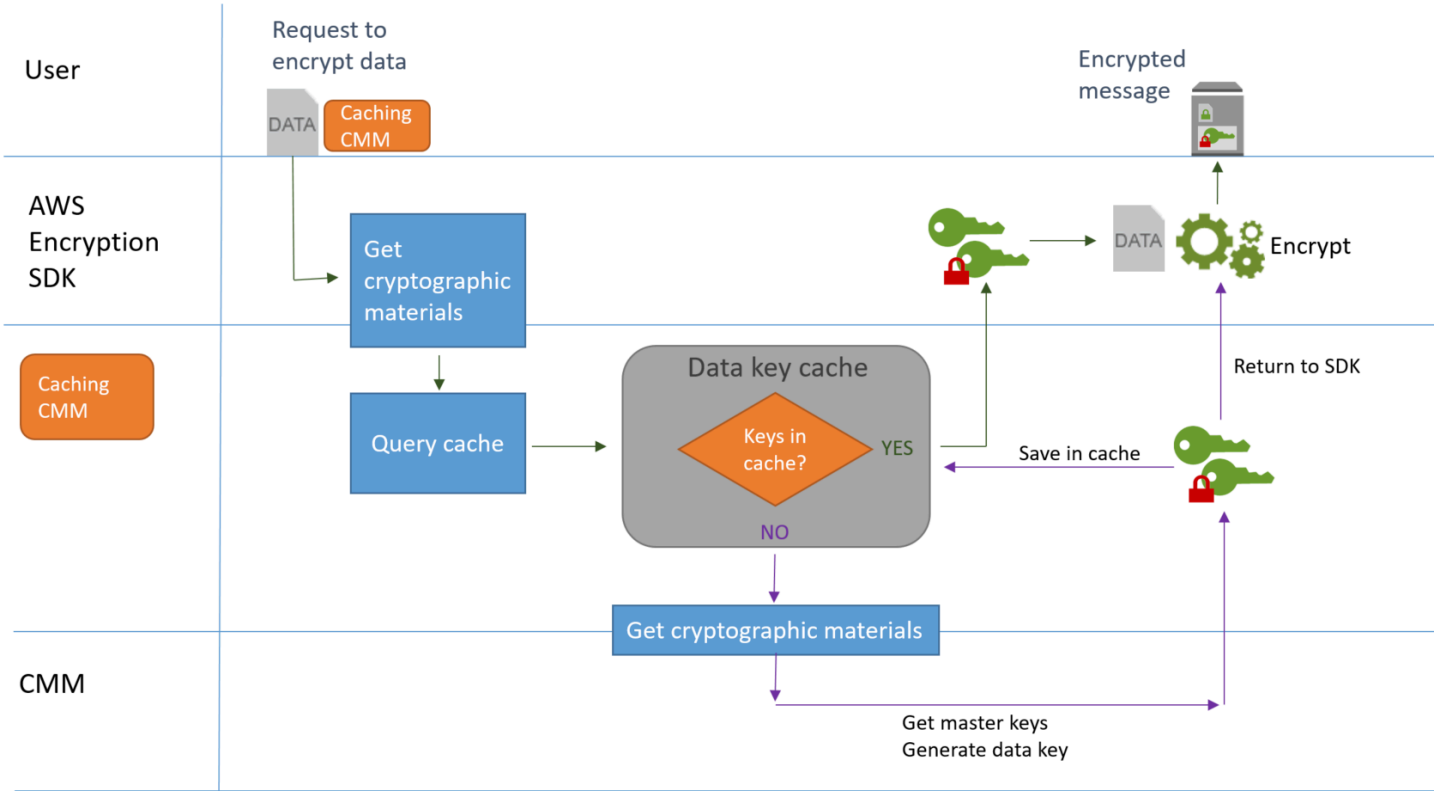
   Si la entrada de caché se ajusta a sus umbrales de seguridad, el CMM de almacenamiento en caché se la devuelve al SDK. De lo contrario, indica a la caché que expulse la entrada de la caché y continúa como si no hubiese encontrado ninguna coincidencia.

b. Si la caché no encuentra ninguna coincidencia válida, el CMM de almacenamiento en caché pide al CMM subyacente que genere una nueva clave de datos.

   La CMM subyacente obtiene los materiales criptográficos de su anillo de claves (C and JavaScript) o proveedor de claves maestras (Java y Python). Esto podría requerir una llamada a un servicio como AWS Key Management Service. El CMM subyacente devuelve las copias de texto no cifrado y cifrado de la clave de datos al CMM de almacenamiento en caché.

   El CMM de almacenamiento en caché guarda los nuevos materiales de cifrado en la caché.

4. El CMM de almacenamiento en caché devuelve los materiales de cifrado al AWS Encryption SDK.
5. AWS Encryption SDK Utiliza la clave de datos de texto plano para cifrar los datos. Almacena los datos cifrados y las claves de datos cifrados en un mensaje cifrado, que devuelve al usuario.



Creación de una caché de materiales criptográficos

AWS Encryption SDK Define los requisitos de una caché de materiales criptográficos utilizada en el almacenamiento en caché de claves de datos. También proporciona una caché local, que es una caché configurable, en memoria y usada menos recientemente (LRU). Para crear una instancia de la caché local, usa el `LocalCryptoMaterialsCache` constructor en Java y Python, la getLocalCryptographic MaterialsCache función en JavaScript o el `aws_cryptosdk_materials_cache_local_new` constructor en C.

La caché local incluye la lógica para la administración básica de la caché, lo que incluye la adición, expulsión y coincidencia de entradas almacenadas en la caché, así como para realizar el mantenimiento de la caché. No es necesario escribir ninguna lógica personalizada de administración de la caché. Puede utilizar la caché local tal cual, personalizarla o sustituirla por cualquier otra caché compatible.

Al crear una caché local, establece su capacidad, es decir, el número máximo de entradas que puede contener esa caché. Este ajuste ayuda a diseñar una caché eficiente con una reutilización limitada de las claves de datos.

Los SDK de cifrado de AWS para Java y los SDK de cifrado de AWS para Python también proporcionan una caché de materiales criptográficos nula (NullCryptoMaterialsCache). NullCryptoMaterialsCache Devuelve un error para todas `GET` las operaciones y no responde a `PUT` las operaciones. Puede usarlo para probar o deshabilitar temporalmente el almacenamiento NullCryptoMaterialsCache en caché en una aplicación que incluya el código de almacenamiento en caché.

En el AWS Encryption SDK, cada caché de materiales criptográficos está asociado a un administrador de materiales criptográficos de almacenamiento en caché (CMM). El CMM de almacenamiento en caché obtiene las claves de datos de la memoria caché, coloca las claves de datos en la caché y aplica los umbrales de seguridad establecidos. Cuando se crea un CMM de almacenamiento en caché, especifica la caché que este utiliza, así como el CMM subyacente o proveedor de claves maestras que genera las claves de datos almacenadas en esa caché.

Creación de un administrador de materiales criptográficos de almacenamiento en caché

Para habilitar el almacenamiento en caché de claves de datos, cree una caché y un administrador de materiales criptográficos de almacenamiento en caché (CMM de almacenamiento en caché). A continuación, en las solicitudes de cifrado o descifrado de datos, especifique un CMM de almacenamiento en caché, en lugar de un administrador de materiales criptográficos (CMM) estándar o un proveedor de claves maestras o conjunto de claves.

Hay dos tipos de. CMMs Ambos obtienen las claves de datos (y el material criptográfico relacionado) pero de maneras distintas, como se indica a continuación:

- Una CMM está asociada a un conjunto de claves (C o JavaScript) o a un proveedor de claves maestras (Java y Python). Cuando el SDK solicita al CMM materiales de cifrado o descifrado, el CMM obtiene los materiales de su conjunto de claves o proveedor de claves maestras. En Java y Python, el CMM utiliza estas claves maestras para generar, cifrar o descifrar las claves de datos. En C y JavaScript, el anillo de claves genera, cifra y devuelve el material criptográfico.
- Un CMM de almacenamiento en caché está asociado a una caché, como una caché local y un CMM subyacente. Cuando el SDK solicita materiales criptográficos al CMM, el CMM de almacenamiento en caché intenta obtenerlos en la caché. Si no encuentra ninguna coincidencia, el CMM de almacenamiento en caché pide los materiales al CMM subyacente. A continuación, almacena los nuevos materiales criptográficos antes de devolvérselos al intermediario.

El CMM de almacenamiento en caché también aplica los umbrales de seguridad que establece para cada entrada de la caché. Dado que el CMM de almacenamiento en caché establece y aplica los umbrales de seguridad, puede utilizar cualquier caché compatible, aunque no se haya diseñado para materiales confidenciales.

¿Qué es una entrada en la caché de claves de datos?

Con el almacenamiento en caché de claves de datos, estas y sus materiales criptográficos relacionados se almacenan en una caché. Cada entrada incluye los elementos que se indican a continuación. Esta información puede resultar útil al decidir si se va a utilizar la característica de almacenamiento en caché de claves de datos y al establecer los umbrales de seguridad en un administrador de materiales criptográficos de almacenamiento (CMM de almacenamiento en caché).

Entradas almacenadas en caché para solicitudes de cifrado

Las entradas que se agregan a una caché de claves de datos a consecuencia de una operación de cifrado incluyen los siguientes elementos:

- Clave de datos en texto no cifrado
- Claves de datos cifrados (una o más)
- Contexto de cifrado
- Clave de firma de mensaje (si procede)
- Conjunto de algoritmos
- Metadatos, incluidos los contadores de uso para aplicar los umbrales de seguridad

Entradas almacenadas en caché para solicitudes de descifrado

Las entradas que se agregan a una caché de claves de datos a consecuencia de una operación de descifrado incluyen los siguientes elementos:

- Clave de datos en texto no cifrado
- Clave de verificación de firma (si procede)
- Metadatos, incluidos los contadores de uso para aplicar los umbrales de seguridad

Contexto de cifrado: cómo seleccionar las entradas de la caché

Puede especificar un contexto de cifrado en cualquier solicitud de cifrado de datos. Sin embargo, el contexto de cifrado desempeña una función especial en el almacenamiento en caché de claves de datos. Permite crear subgrupos de claves de datos en la caché, aunque las claves de datos se originen a partir del mismo CMM de almacenamiento en caché.

Un contexto de cifrado es un conjunto de pares de clave-valor que contienen datos no secretos arbitrarios. Durante el cifrado, el contexto de cifrado se vincula criptográficamente a los datos cifrados, de tal forma que se requiere el mismo contexto de cifrado para descifrar los datos. En el AWS Encryption SDK, el contexto de cifrado se almacena en el mensaje cifrado con los datos cifrados y las claves de datos.

Cuando se utiliza una caché de claves de datos, también se puede utilizar el contexto de cifrado para seleccionar claves de datos concretas almacenadas en la caché para las operaciones de cifrado. El contexto de cifrado se guarda en la entrada de la caché con la clave de datos (forma parte del ID de la entrada de la caché). Las claves de datos almacenadas en caché se reutilizan únicamente si contextos de cifrado coinciden. Si desea reutilizar determinadas claves de datos para una solicitud de cifrado, especifique el mismo contexto de cifrado. Si desea evitar estas claves de datos, especifique otro contexto de cifrado.

El contexto de cifrado siempre es opcional, pero se recomienda. Si no especifica un contexto de cifrado en la solicitud, se incluirá un contexto de cifrado vacío en el identificador de la entrada de la caché y se hará coincidir con cada solicitud.

¿Usa mi aplicación claves de datos almacenadas en caché?

El almacenamiento en caché de claves de datos es una estrategia de optimización que es muy eficaz en determinadas aplicaciones y cargas de trabajo. Sin embargo, debido a que conlleva cierto riesgo, es importante que determine primero hasta qué punto es probable que sea eficaz para su situación y luego decida si los beneficios compensan los riesgos.

Dado que el almacenamiento en caché de claves de datos reutiliza claves de datos, el efecto más evidente es que se reduce el número de llamadas para generar nuevas claves de datos. Cuando se implementa el almacenamiento en caché de claves de datos, solo se AWS Encryption SDK llama a la AWS KMS `GenerateDataKey` operación para crear la clave de datos inicial y cuando no se utiliza la caché. Sin embargo, el almacenamiento en caché mejora el rendimiento perceptiblemente solo en el caso de aplicaciones que generen numerosas claves de datos con las mismas características, incluido el mismo contexto de cifrado y conjunto de algoritmos.

Para determinar si su implementación AWS Encryption SDK utiliza realmente claves de datos de la caché, pruebe las siguientes técnicas.

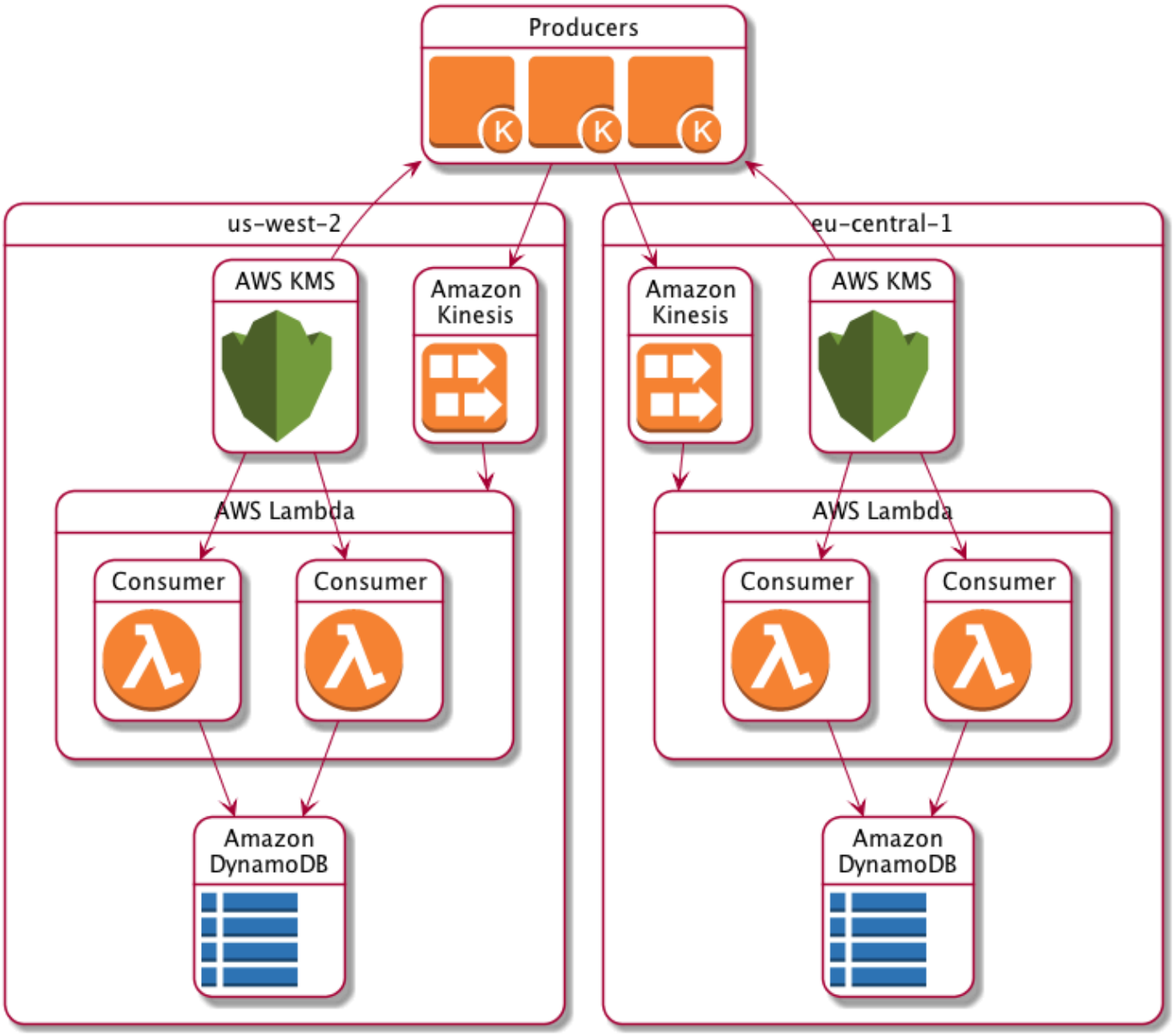
- En los registros de la infraestructura de clave maestra, consulte la frecuencia de las llamadas para crear nuevas claves de datos. Cuando existe un almacenamiento en caché de claves de datos efectivo, el número de llamadas para crear nuevas claves debe caer perceptiblemente. Por ejemplo, si utiliza un proveedor de claves AWS KMS maestras o un conjunto de claves, busque GenerateDataKeyllamadas en los CloudTrail registros.
- Compare los mensajes cifrados que el AWS Encryption SDK devuelve en respuesta a diferentes solicitudes de cifrado. Por ejemplo, si utiliza el SDK de cifrado de AWS para Java, compare el ParsedCiphertextobjeto de diferentes llamadas de cifrado. En el SDK de cifrado de AWS para JavaScript, compare el contenido de la `encryptedDataKeys` propiedad del MessageHeader. Cuando se reutilizan claves de datos, las claves de datos cifradas del mensaje cifrado son idénticas.

Ejemplo de almacenamiento en caché de claves de datos

En este ejemplo se utiliza el almacenamiento en caché de claves de datos con una caché local para acelerar una aplicación en la que los datos generados por varios dispositivos se cifren y almacenen en diferentes regiones.

En este caso, varios productores de datos generan datos, los cifran y escriben en una secuencia de Kinesis en cada región. Las funciones AWS Lambda (consumidores) descifran las transmisiones y escriben datos de texto no cifrado en una tabla de DynamoDB en la Región. Los productores y los consumidores de datos utilizan el AWS Encryption SDK y un AWS KMS proveedor de claves maestras. Para reducir las llamadas a KMS, cada productor y consumidor tiene su propio caché local.

Puede encontrar el código fuente de estos ejemplos en Java y Python. El ejemplo también incluye una CloudFormation plantilla que define los recursos de las muestras.



Resultados de la caché local

En la siguiente tabla se muestra que una caché local reduce las llamadas totales a KMS (por segundo y región) de este ejemplo al 1 % de su valor original.

Solicitudes de productores

	Solicitudes por segundo y cliente			Clientes por región	Promedio de solicitudes
	Generar clave de	Cifrado de clave de	Total (por región)		

	datos (us-west-2)	datos (eu-central-1)			por segundo y región
Sin caché	1	1	1	500	500
Caché local	1 rps/100 usos	1 rps/100 usos	1 rps/100 usos	500	5

Solicitudes de consumidores

	Solicitudes por segundo y cliente			Clientes por región	Promedio de solicitudes por segundo y región
	Descifrar clave de datos	Productores	Total		
Sin caché	1 rps por productor	500	500	2	1 000
Caché local	1 rps por productor / 100 usos	500	5	2	10

Ejemplo de almacenamiento en caché de claves de datos

Este ejemplo de código crea una implementación simple de almacenamiento en caché de claves de datos con una caché local en Java y Python. El código crea dos instancias de una caché local: una para los productores de datos que cifran datos y otra para los consumidores de datos (AWS Lambda funciones) que descifran los datos. Para obtener más información sobre la implementación del almacenamiento en caché de claves de datos en cada lenguaje, consulte la documentación de Javadoc y Python para AWS Encryption SDK.

El almacenamiento en caché de claves de datos está disponible para todos los lenguajes de programación compatibles. AWS Encryption SDK

Para ver ejemplos completos y comprobados del uso del almacenamiento en caché de claves de datos en el AWS Encryption SDK, consulte:

- C/C++: caching_cmm.cpp
- Java: SimpleDataKeyCachingExample .java
- JavaScript Navegador: caching_cmm.ts
- JavaScript Node.js: caching_cmm.ts
- Python: data_key_caching_basic.py

Productor

El productor obtiene un mapa, lo convierte a JSON, lo utiliza AWS Encryption SDK para cifrarlo y envía el registro de texto cifrado a una transmisión de Kinesis en cada uno de ellos. Región de AWS

El código define un administrador de materiales criptográficos de almacenamiento en caché (CMM de almacenamiento en caché) y lo asocia a una caché local y a un proveedor de claves maestras de AWS KMS subyacente. El CMM de almacenamiento en caché almacena en caché las claves de datos (y los materiales criptográficos relacionados) del proveedor de claves maestras. También interacciona con la caché en nombre del SDK y aplica los umbrales de seguridad establecidos.

Como la llamada al método de cifrado especifica un CMM de almacenamiento en caché, en lugar de un administrador de materiales criptográficos (CMM) o un proveedor de clave maestra normal, el cifrado utilizará el almacenamiento en caché de claves de datos.

Java

En el siguiente ejemplo, se utiliza la versión 2. x del SDK de cifrado de AWS para Java. Versión 3. x del SDK de cifrado de AWS para Java CMM de almacenamiento en caché de claves de datos está en desuso. Con la versión 3. x, también puede utilizar el llavero AWS KMS jerárquico, una solución alternativa de almacenamiento en caché de materiales criptográficos.

```
/*
 * 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.kmssdkv2.KmsMasterKey;
import com.amazonaws.encryptionsdk.kmssdkv2.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
            )
```

Consumidor

El consumidor de datos es una función de AWS Lambda que se activa mediante eventos de Kinesis. Descifra y deserializa cada registro y lo escribe en texto no cifrado en una tabla de Amazon DynamoDB de la misma región.

Al igual que el código de productor, el código de consumidor permite el almacenamiento en caché de claves de datos mediante el uso de un administrador de materiales criptográficos de almacenamiento en caché (CMM) en llamadas al método de descifrado.

El código Java crea un proveedor de claves maestras en modo estricto con una especificación. AWS KMS key El modo estricto no es obligatorio para descifrar, pero es una práctica recomendada. El código Python usa el modo de descubrimiento, que permite AWS Encryption SDK usar cualquier clave de empaquetado que cifre una clave de datos para descifrarla.

Java

El siguiente ejemplo usa la versión 2. x del SDK de cifrado de AWS para Java. Versión 3. x del SDK de cifrado de AWS para Java CMM de almacenamiento en caché de claves de datos está en desuso. Con la versión 3. x, también puede utilizar el llavero AWS KMS jerárquico, una solución alternativa de almacenamiento en caché de materiales criptográficos.

Este código crea un proveedor de claves maestras para descifrar en modo estricto. Solo AWS Encryption SDK puede usar AWS KMS keys lo que usted especifique para descifrar el mensaje.

```
/*
 * 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.kmssdkv2.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 UnsupportedEncodingException {
        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

Este código de Python se descifra con un proveedor de claves maestras en modo de detección. Permite AWS Encryption SDK utilizar cualquier clave de encapsulamiento que haya cifrado una clave de datos para descifrarla. La práctica recomendada es el modo estricto, en el que se especifican las claves de encapsulamiento que se pueden utilizar para el descifrado.

```
"""
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)
```

Ejemplo de almacenamiento en caché de claves de datos: plantilla CloudFormation

Esta CloudFormation plantilla configura todos los AWS recursos necesarios para reproducir el ejemplo de almacenamiento en caché de claves de datos.

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
```

Versiones del AWS Encryption SDK

Las implementaciones del AWS Encryption SDK lenguaje utilizan el control de versiones semántico para facilitar la identificación de la magnitud de los cambios en cada versión. Un cambio en el número de la versión principal, como 1.x.x a 2.x.x, indica un cambio importante que probablemente requiera cambios en el código y una implementación planificada. Es posible que los cambios importantes en una nueva versión no afecten a todos los casos de uso; consulta las notas de la versión para ver si te afectan a ti. Un cambio en una versión secundaria, como x.1.x a x.2.x, siempre es compatible con versiones anteriores, pero puede incluir elementos obsoletos.

Siempre que sea posible, usa la última versión del lenguaje AWS Encryption SDK de programación que hayas elegido. La política de mantenimiento y soporte de cada versión difiere entre las implementaciones del lenguaje de programación. Para obtener más información sobre las versiones compatibles en su lenguaje de programación preferido, consulte el `SUPPORT_POLICY.rst` archivo en su GitHubrepositorio.

Cuando las actualizaciones incluyen nuevas características que requieren una configuración especial para evitar errores de cifrado o descifrado, proporcionamos una versión intermedia e instrucciones detalladas para su uso. Por ejemplo, las versiones 1.7.x y 1.8.x están diseñadas para ser versiones de transición que le ayudan a actualizar desde versiones anteriores a la 1.7 x a las versiones 2.0.x y posteriores. Para obtener más información, consulte Migración de su AWS Encryption SDK.

> **Note**
>
> La x del número de versión representa cualquier parche de la versión principal y secundaria. Por ejemplo, la versión 1.7.x representa todas las versiones que comienzan por 1.7, incluidas la 1.7.1 y la 1.7.9.
>
> Las nuevas funciones de seguridad se publicaron originalmente en las versiones 1.7 de AWS Encryption CLI. x y 2.0. x. Sin embargo, AWS Encryption CLI versión 1.8. x reemplaza a la versión 1.7. x y CLI de AWS cifrado 2.1. x reemplaza a 2.0. x. Para obtener más información, consulte el aviso de seguridad correspondiente en el aws-encryption-sdk-clirepositorio de GitHub.

En las siguientes tablas se ofrece una visión general de las principales diferencias entre las versiones compatibles AWS Encryption SDK de cada lenguaje de programación.

C

Para obtener una descripción detallada de todos los cambios, consulta el ChangeLog.md en el repositorio de. aws-encryption-sdk-c GitHub

Versión principal	Detalles		Fase del ciclo de vida de la versión principal del SDK
1.x	1.0	Versión inicial.	End-of-Support fase
	1.7	Actualizaciones AWS Encryption SDK que ayudan a los usuarios de versiones anteriores a actualizar a las versiones 2.0. x y versiones posteriores. Para obtener más información, consulte la versión 1.7. x.	
2.x	2.0	Actualizaciones de AWS Encryption SDK. Para obtener más información, consulte la versión 2.0. x.	Disponibilidad general (GA)
	2.2	Mejoras en el proceso de descifrado de mensajes.	
	2.3	Añade compatibilidad con claves AWS KMS multirregionales.	

C#/.NET

Para obtener una descripción detallada de todos los cambios, consulta el ChangeLog.md en el repositorio de. `aws-encryption-sdk-net` GitHub

Versión principal	Detalles		Fase del ciclo de vida de la versión principal del SDK
3.x	3.1.0	Versión inicial.	Fin del soporte La versión 3.x AWS Encryption SDK para .NET ha llegado al final de su período de soporte; por favor, actualice a la versión 4.x.
4.x	4.0	Añade compatibilidad con el anillo de claves AWS KMS jerárquico, el contexto de cifrado requerido (CMM) y los anillos de claves RSA asimétricos. AWS KMS	Disponibilidad general (GA)

Interfaz de línea de comandos (CLI)

Para obtener una descripción detallada de todos los cambios, consulte Versiones de la CLI AWS de cifrado y el ChangeLog.rst en el repositorio de. aws-encryption-sdk-cli GitHub

Versión principal	Detalles		Fase del ciclo de vida de la versión principal del SDK

1.x	1.0	Versión inicial.	End-of-Support fase
	1.7	Actualizaciones AWS Encryption SDK que ayudan a los usuarios de versiones anteriores a actualizar a las versiones 2.0. x y versiones posteriores. Para obtener más información, consulte la versión 1.7. x.	
2.x	2.0	Actualizaciones de AWS Encryption SDK. Para obtener más información, consulte la versión 2.0. x.	End-of-Support fase
	2.1	Elimina el `--discovery` parámetro y lo sustituye por el `discovery` atributo del `--wrapping-keys` parámetro. La versión 2.1.0 de la CLI de AWS cifrado es equivalente a la versión 2.0 en otros lenguajes de programación.	
	2.2	Mejoras en el proceso de descifrado de mensajes.	

3.x	3.0	Añade compatibilidad con claves AWS KMS multirregionales.	End-of-Support fase
4.x	4.0	La CLI de AWS cifrado ya no es compatible con Python 2 o Python 3.4. A partir de la versión principal 4. x de la CLI de AWS cifrado, solo se admite Python 3.5 o posterior .	Disponibilidad general (GA)
	4.1	La CLI de AWS cifrado ya no es compatible con Python 3.5. A partir de la versión 4.1. x de la CLI de AWS cifrado, solo se admite Python 3.6 o posterior.	
	4.2	La CLI de AWS cifrado ya no es compatible con Python 3.6. A partir de la versión 4.2. x de la CLI de AWS cifrado, solo se admite Python 3.7 o posterior.	

Java

Para obtener una descripción detallada de todos los cambios, consulte ChangeLog.rst en el repositorio de. aws-encryption-sdk-java GitHub

Versión principal	Detalles		Fase del ciclo de vida de la versión principal del SDK
1.x	1.0	Versión inicial.	End-of-Support fase
	1.3	Añade compatibilidad con el administrador de materiales criptográficos y el almacenamiento en caché de claves de datos. Pasó a la generación IV determinista.	
	1.6.1	Los desaprueba `AwsCrypto.encryptString()` y los reemplaza por `AwsCrypto.decryptString()` y. `AwsCrypto.encryptData()` `AwsCrypto.decryptData()`	
	1.7	Actualizaciones de las AWS Encryption SDK que ayudan a los usuarios de versiones	

		anteriores a actualizar a las versiones 2.0. x y versiones posteriores. Para obtener más información, consulte la versión 1.7. x.	
2.x	2.0	Actualizaciones de AWS Encryption SDK. Para obtener más información, consulte la versión 2.0. x.	Disponibilidad general (GA) La versión 2.x SDK de cifrado de AWS para Java entrará en modo de mantenimiento en 2024.
	2.2	Mejoras en el proceso de descifrado de mensajes.	
	2.3	Añade compatibilidad con claves AWS KMS multirregionales.	
	2.4	Añade compatibilidad con. AWS SDK for Java 2.x	

3.x	3.0	Se integra SDK de cifrado de AWS para Java con la biblioteca de proveedores de materiales (MPL). Añade compatibilidad con llaveros RSA simétricos y asimétricos, anillos de AWS KMS claves AWS KMS ECDH, anillos AWS KMS jerárquicos, anillos de claves AES sin procesar, anillos de claves RSA sin procesar, anillos de claves ECDH sin procesar, anillos de claves múltiples y el contexto de cifrado requerido, CMM.	Disponibilidad general (GA)

Go

Para obtener una descripción detallada de todos los cambios, consulta el archivo Changelog.md en el directorio Go del repositorio correspondiente. aws-encryption-sdk GitHub

Versión principal	Detalles		Fase del ciclo de vida de la versión principal del SDK
0.1. x	0.1.0	Versión inicial.	Disponibilidad general (GA)

JavaScript

Para obtener una descripción detallada de todos los cambios, consulta el ChangeLog.md en el repositorio de. aws-encryption-sdk-javascript GitHub

Versión principal	Detalles		Fase del ciclo de vida de la versión principal del SDK
1.x	1.0	Versión inicial.	End-of-Support fase
	1.7	Actualizaciones AWS Encryption SDK que ayudan a los usuarios de versiones anteriores a actualizar a las versiones 2.0. x y versiones posteriores. Para obtener más información, consulte la versión 1.7. x.	
2.x	2.0	Actualizaciones de AWS Encryption SDK. Para obtener más información, consulte la versión 2.0. x.	End-of-Support fase
	2.2	Mejoras en el proceso de descifrado de mensajes.	
	2.3	Añade compatibilidad con claves AWS KMS multirregionales.	
3.x	3.0	Elimina la cobertura de CI para el nodo	Mantenimiento

		10. Actualiza las dependencias para que ya no sean compatibles con los nodos 8 y 10.	Support para la versión 3.x SDK de cifrado de AWS para JavaScript finalizará el 17 de enero de 2024.
4.x	4.0	Requiere la versión 3 de los s SDK de cifrado de AWS para JavaScript`kms-client` para usar el AWS KMS llavero.	Disponibilidad general (GA)

Python

Para obtener una descripción detallada de todos los cambios, consulta el ChangeLog.rst en el repositorio de. aws-encryption-sdk-python GitHub

Versión principal	Detalles		Fase del ciclo de vida de la versión principal del SDK
1.x	1.0	Versión inicial.	End-of-Support fase
	1.3	Añade compatibilidad con el administrador de materiales criptográficos y el almacenamiento en caché de claves de datos. Pasó a la generación IV determinista.	
	1.7	Actualizaciones AWS Encryption SDK que	

		ayudan a los usuarios de versiones anteriores a actualizar a las versiones 2.0. x y versiones posteriores. Para obtener más información, consulte la versión 1.7. x.	
2.x	2.0	Actualizaciones de AWS Encryption SDK. Para obtener más información, consulte la versión 2.0. x.	End-of-Support fase
	2.2	Mejoras en el proceso de descifrado de mensajes.	
	2.3	Añade compatibilidad con claves AWS KMS multirregionales.	
3.x	3.0	Ya SDK de cifrado de AWS para Python no es compatible con Python 2 o Python 3.4. A partir de la versión principal 3. x de SDK de cifrado de AWS para Python, solo se admite Python 3.5 o posterior.	Disponibilidad general (GA)

4.x	4.0	Se integra SDK de cifrado de AWS para Python con la biblioteca de proveedores de materiales (MPL).	Disponibilidad general (GA)

Rust

Para obtener una descripción detallada de todos los cambios, consulte el ChangeLog.md en el directorio Rust del repositorio en cuestión. aws-encryption-sdk GitHub

Versión principal	Detalles		Fase del ciclo de vida de la versión principal del SDK
1.x	1.0	Versión inicial.	Disponibilidad general (GA)

Detalles de la versión

La siguiente lista describe las principales diferencias entre las versiones compatibles de AWS Encryption SDK.

Temas

- Versiones anteriores a 1.7.x
- Versión 1.7.x
- Versión 2.0.x
- Versión 2.2.x
- Versión 2.3.x

Versiones anteriores a 1.7.x

> **Note**
>
> Todos los 1. x. Las versiones x de AWS Encryption SDK están en end-of-supportfase. Actualice a la última versión disponible AWS Encryption SDK para su lenguaje de programación tan pronto como sea posible. Para actualizar desde una AWS Encryption SDK versión anterior a la 1.7. x, primero debe actualizar a la 1.7. x. Para obtener más información, consulte Migración de su AWS Encryption SDK.

Versiones AWS Encryption SDK anteriores a la 1.7. Las x incluyen importantes funciones de seguridad, como el cifrado mediante el algoritmo estándar de cifrado avanzado en Galois/Counter modo (AES-GCM), una función de derivación de extract-and-expand claves basada en HMAC (HKDF), la firma y una clave de cifrado de 256 bits. Sin embargo, estas versiones no admiten las mejores prácticas que recomendamos, como el compromiso clave.

Versión 1.7.x

> **Note**
>
> Todos los 1. x. Las versiones x de AWS Encryption SDK están en end-of-supportfase.

Versión 1.7. x está diseñada para ayudar a los usuarios de versiones anteriores AWS Encryption SDK a actualizar a las versiones 2.0. x y versiones posteriores. Si es la primera vez que utiliza AWS Encryption SDK, puede omitir esta versión y empezar con la última versión disponible en su lenguaje de programación.

La versión 1.7.x es totalmente compatible con versiones anteriores; no introduce ningún cambio importante ni cambia el comportamiento del AWS Encryption SDK. También es compatible con versiones posteriores; le permite actualizar su código para que sea compatible con la versión 2.0.x. Incluye nuevas características, pero no las habilita por completo. Además, requiere valores de configuración que le impidan adoptar inmediatamente todas las características nuevas hasta que esté preparado.

La versión 1.7.x incluye los siguientes cambios:

AWS KMS actualizaciones del proveedor de claves maestras (obligatorias)

Versión 1.7. x introduce nuevos constructores SDK de cifrado de AWS para Python que crean explícitamente proveedores de claves AWS KMS maestras en modo estricto o de descubrimiento. SDK de cifrado de AWS para Java Esta versión añade cambios similares a la interfaz de AWS Encryption SDK línea de comandos (CLI). Para obtener más información, consulte Actualización de los proveedores de claves AWS KMS maestras.

- En el modo estricto, los proveedores de claves maestras de AWS KMS requieren una lista de claves de encapsulamiento y cifran y descifran únicamente con las claves de encapsulamiento que usted especifique. Es una práctica recomendada de AWS Encryption SDK que garantiza que utilice la clave de encapsulamiento deseada.
- En el modo de detección, los proveedores de claves maestras de AWS KMS no aceptan ninguna clave de encapsulamiento. No puede utilizarlas para cifrar. Cuando se descifra, puede utilizarse cualquier clave de encapsulamiento para descifrar una clave datos cifrada. Sin embargo, puede limitar las claves de encapsulamiento utilizadas para el descifrado a aquellas en particular Cuentas de AWS. Este filtro de detección es opcional, pero es una práctica recomendada.

Los constructores que crean versiones anteriores de los proveedores de claves AWS KMS maestras están obsoletos en la versión 1.7. x y se eliminaron en la versión 2.0. x. Estos constructores crean instancias de los proveedores de claves maestras que cifran con las claves de encapsulamiento que especifique. Sin embargo, descifran las claves de datos cifradas utilizando la clave de encapsulamiento que las cifró, sin tener en cuenta las claves de encapsulamiento especificadas. Los usuarios pueden descifrar involuntariamente los mensajes con claves de empaquetado que no tienen intención de utilizar, incluso AWS KMS keys en otras regiones Cuentas de AWS .

No hay cambios en los constructores de las claves maestras. AWS KMS Al cifrar y descifrar, las claves AWS KMS maestras utilizan únicamente las AWS KMS key que usted especifique.

AWS KMS actualizaciones del conjunto de claves (opcional)

Versión 1.7. x añade un nuevo filtro a SDK de cifrado de AWS para JavaScript las SDK de cifrado de AWS para C implementaciones que limita los llaveros de AWS KMS descubrimiento a determinados. Cuentas de AWS Este filtro de detección de cuentas nuevas es opcional, pero es una práctica recomendada. Para obtener más información, consulte Actualización de los AWS KMS llaveros.

No hay cambios en los fabricantes de los llaveros. AWS KMS AWS KMS Los llaveros estándar se comportan como los proveedores de claves maestras en modo estricto. AWS KMS Los llaveros de detección se crean de forma explícita en el modo de detección.

Pasar un identificador de clave a Decrypt AWS KMS

A partir de la versión 1.7. x, al descifrar las claves de datos cifradas, AWS Encryption SDK siempre especifica una AWS KMS key en sus llamadas a la operación de AWS KMS descifrado. AWS Encryption SDK obtiene el valor del identificador de clave AWS KMS key de los metadatos de cada clave de datos cifrados. Esta característica no requiere ningún cambio de código.

No AWS KMS key es necesario especificar el ID de clave para descifrar el texto cifrado con una clave KMS de cifrado simétrico, pero es una práctica recomendada.AWS KMS Al igual que si se especifican las claves de empaquetado en el proveedor de claves, esta práctica garantiza que AWS KMS solo se descifra con la clave de empaquetado que se va a utilizar.

Descifrar el texto cifrado con un compromiso clave

La versión 1.7.x puede descifrar el texto cifrado, cifrado con o sin compromiso clave. Sin embargo, no puede cifrar el texto cifrado con un compromiso clave. Esta propiedad le permite implementar por completo aplicaciones que pueden descifrar el texto cifrado con compromiso clave antes de que encuentren dicho texto cifrado. Como esta versión descifra los mensajes cifrados sin necesidad de asignar la clave, no es necesario volver a cifrar ningún texto cifrado.

Para implementar este comportamiento, versión 1.7. x incluye una nueva configuración de la política de compromiso que determina si AWS Encryption SDK pueden cifrar o descifrar con un compromiso de clave. En la versión 1.7.x, el único valor válido para la política de compromiso, `ForbidEncryptAllowDecrypt`, se utiliza en todas las operaciones de cifrado y descifrado. Este valor impide el cifrado del AWS Encryption SDK con cualquiera de los nuevos conjuntos de algoritmos que incluyen el compromiso clave. Permite descifrar el AWS Encryption SDK texto cifrado con y sin compromiso de clave.

Sin embargo, solo hay un valor de política de compromiso válido en la versión 1.7. x, necesitamos que puedas establecer este valor de forma explícita cuando utilices las nuevas APIs incorporadas en esta versión. Si se establece este valor de forma explícita, se evita que la política de compromiso cambie automáticamente a `require-encrypt-require-decrypt` cuando sube de categoría a la versión 2.1.x. En su lugar, puede migrar su política de compromisos por etapas.

Paquetes de algoritmos con un compromiso clave

La versión 1.7.x incluye dos nuevos conjuntos de algoritmos que respaldan el compromiso clave. Una incluye la firma; la otra no. Al igual que los conjuntos de algoritmos compatibles

anteriormente, estos dos nuevos conjuntos de algoritmos incluyen el cifrado con AES-GCM, una clave de cifrado de 256 bits y una función de derivación de claves (HKDF) basada en HMAC extract-and-expand.

Sin embargo, el conjunto de algoritmos predeterminado utilizado para el cifrado no cambia. Estos conjuntos de algoritmos se añaden a la versión 1.7.x para preparar la aplicación para utilizarlos en las versiones 2.0.x y posteriores.

Cambios en la implementación de CMM

La versión 1.7.x introduce cambios en la interfaz predeterminada del administrador de materiales criptográficos (CMM) para respaldar los compromisos de clave. Este cambio solo le afecta si ha escrito un CMM personalizada. Para obtener más información, consulte la documentación o el repositorio de la API de su lenguaje de programación. GitHub

Versión 2.0.x

Versión 2.0. x es compatible con las nuevas funciones de seguridad que se ofrecen en la AWS Encryption SDK, incluidas las claves de embalaje específicas y el compromiso clave. Para apoyar estas características, la versión 2.0.x incluye cambios importantes en todas las versiones anteriores del AWS Encryption SDK. Puede prepararse para estos cambios implementando la versión 1.7.x. La versión 2.0.x incluye todas las nuevas características introducidas en la versión 1.7.x con las siguientes adiciones y cambios.

> **Note**
>
> Versión 2. x. x de SDK de cifrado de AWS para Python SDK de cifrado de AWS para JavaScript, y la CLI de AWS cifrado están en end-of-supportfase.
> Para obtener información sobre el soporte y el mantenimiento de esta AWS Encryption SDK versión en su lenguaje de programación preferido, consulte el `SUPPORT_POLICY.rst` archivo en su GitHubrepositorio.

AWS KMS proveedores de claves maestras

Los constructores originales del proveedor de claves AWS KMS maestras que quedaron en desuso en la versión 1.7. x se eliminan en la versión 2.0. x. Debe construir de forma explícita proveedores de claves maestras de AWS KMS en modo estricto o modo de detección.

Descifre el texto cifrado con un compromiso clave

La versión 2.0.x puede cifrar y descifrar el texto cifrado con o sin compromiso clave. Su comportamiento viene determinado por la configuración de la política de compromiso. De forma predeterminada, siempre cifra con compromiso clave y solo descifra el texto cifrado con compromiso clave. A menos que cambie la política de compromiso, el AWS Encryption SDK no descifrará los textos cifrados con ninguna versión anterior de la AWS Encryption SDK, incluida la versión 1.7.x.

> **Important**
>
> De forma predeterminada, la versión 2.0.x no descifrará ningún texto cifrado que se haya cifrado sin el compromiso clave. Si la aplicación puede encontrar un texto cifrado, cifrado sin compromiso clave, defina el valor de la política de compromiso con `AllowDecrypt`.

En la versión 2.0.x, la configuración de la política de compromiso tiene tres valores válidos:

- `ForbidEncryptAllowDecrypt`: el AWS Encryption SDK no puede cifrar con un compromiso clave. Puede descifrar textos cifrados, cifrados con o sin compromiso clave.
- `RequireEncryptAllowDecrypt`: el AWS Encryption SDK se debe cifrar con un compromiso clave. Puede descifrar textos cifrados, cifrados con o sin compromiso clave.
- `RequireEncryptRequireDecrypt`(predeterminado): se AWS Encryption SDK debe cifrar con un compromiso de clave. Solo descifra los textos cifrados con un compromiso clave.

Si va a migrar de una versión anterior AWS Encryption SDK a la versión 2.0. x, defina la política de compromiso en un valor que garantice que puede descifrar todos los textos cifrados existentes que pueda encontrar la aplicación. Es probable que modifique esta configuración con el tiempo.

Versión 2.2.x

Añade compatibilidad con firmas digitales y limita las claves de datos cifrados.

> **Note**
>
> Versión 2. x. x de SDK de cifrado de AWS para Python SDK de cifrado de AWS para JavaScript, y la CLI de AWS cifrado están en end-of-supportfase.

Para obtener información sobre el soporte y el mantenimiento de esta AWS Encryption SDK versión en su lenguaje de programación preferido, consulte el `SUPPORT_POLICY.rst` archivo en su GitHubrepositorio.

Firmas digitales

Para mejorar el manejo de las firmas digitales al descifrar, AWS Encryption SDK incluye las siguientes funciones:

- Modo sin streaming: devuelve el texto no cifrado solo después de procesar todas las entradas, incluida la verificación de la firma digital, si existe alguna. Esta característica impide utilizar texto no cifrado antes de verificar la firma digital. Utilice esta característica siempre que descifre datos cifrados con firmas digitales (el conjunto de algoritmos predeterminado). Por ejemplo, dado que la CLI de AWS cifrado siempre procesa los datos en modo streaming, utilice el `--buffer` parámetro al descifrar el texto cifrado con firmas digitales.
- Modo de descifrado solo sin firmar: esta característica solo descifra el texto cifrado sin firmar. Si el descifrado encuentra una firma digital en el texto cifrado, se produce un error en la operación. Utilice esta característica para evitar procesar involuntariamente el texto no cifrado de los mensajes firmados antes de verificar la firma.

Limitar las claves de datos cifrados

Puede limitar el número de claves de datos cifradas en un mensaje cifrado. Esta característica puede ayudarle a detectar un proveedor de claves maestras o un conjunto de claves mal configurados al cifrar, o a identificar un texto cifrado malicioso al descifrar.

Debe limitar las claves de datos cifradas al descifrar los mensajes de una fuente que no sea de confianza. Evita las llamadas innecesarias, costosas y potencialmente exhaustivas a su infraestructura clave.

Versión 2.3.x

Añade compatibilidad con claves multirregionales AWS KMS . Para obtener más información, consulte Uso de varias regiones AWS KMS keys.

> **Note**
>
> La CLI de AWS cifrado admite claves multirregionales a partir de la versión 3.0. x.

Versión 2. x. x de SDK de cifrado de AWS para Python SDK de cifrado de AWS para JavaScript, y la CLI de AWS cifrado están en end-of-supportfase.
Para obtener información sobre el soporte y el mantenimiento de esta AWS Encryption SDK versión en su lenguaje de programación preferido, consulte el `SUPPORT_POLICY.rst` archivo en su GitHubrepositorio.

Migración de su AWS Encryption SDK

AWS Encryption SDK Admite múltiples implementaciones de lenguajes de programación interoperables, cada una de las cuales está desarrollada en un repositorio de código abierto en. GitHub Como práctica recomendada, le recomendamos que utilice la última versión de AWS Encryption SDK para cada lenguaje.

Puede actualizar de forma segura desde la versión 2.0. x o posterior AWS Encryption SDK a la última versión. Sin embargo, la 2.0. La versión x AWS Encryption SDK presenta nuevas e importantes funciones de seguridad, algunas de las cuales son cambios importantes. Para actualizar desde versiones anteriores a la 1.7.x a las versiones 2.0.x y versiones posteriores, primero debe actualizar a la última versión 1.x. Los temas de esta sección están diseñados para ayudarle a comprender los cambios, seleccionar la versión correcta para su aplicación y migrar de forma segura y correcta a las versiones más recientes de la AWS Encryption SDK.

Para obtener información sobre las versiones más importantes de AWS Encryption SDK, consulteVersiones del AWS Encryption SDK.

Important

No actualice directamente desde una versión anterior a la 1.7.x a la versión 2.0.x o posterior sin actualizar primero a la última versión 1.versión x. Si actualiza directamente a la versión 2.0. x o posterior y habilite todas las funciones nuevas de forma inmediata, AWS Encryption SDK no podrá descifrar el texto cifrado en versiones anteriores de. AWS Encryption SDK

Note

La versión más antigua de AWS Encryption SDK para .NET es la 3.0. x. Todas las AWS Encryption SDK versiones de.NET son compatibles con las mejores prácticas de seguridad introducidas en la versión 2.0. x de AWS Encryption SDK. Puede actualizar de forma segura a la última versión sin cambios de código o datos.
AWS CLI de cifrado: Al leer esta guía de migración, utilice la 1.7. x instrucciones de migración para AWS Encryption CLI 1.8. x y utilice la 2.0. x instrucciones de migración para AWS Encryption CLI 2.1. x. Para obtener más información, consulte Versiones de la CLI AWS de cifrado.

Las nuevas funciones de seguridad se publicaron originalmente en las versiones 1.7 de AWS Encryption CLI. x y 2.0. x. Sin embargo, AWS Encryption CLI versión 1.8. x reemplaza a la versión 1.7. x y CLI de AWS cifrado 2.1. x reemplaza a 2.0. x. Para obtener más información, consulte el aviso de seguridad correspondiente en el aws-encryption-sdk-clirepositorio de GitHub.

Nuevos usuarios

Si es la primera vez que lo AWS Encryption SDK utiliza, instale la última versión de AWS Encryption SDK para su lenguaje de programación. Los valores predeterminados habilitan todas las funciones de seguridad del AWS Encryption SDK, incluido el cifrado con firma, la derivación de claves y el compromiso de claves. del AWS Encryption SDK

Usuarios actuales

Se recomienda que lleva a cabo la actualización de la versión actual a la versión más reciente disponible lo antes posible. Todos los 1. Las versiones x de AWS Encryption SDK están en end-of-support fase, al igual que las versiones posteriores en algunos lenguajes de programación. Para obtener más información sobre el estado de soporte y mantenimiento de su lenguaje AWS Encryption SDK de programación, consulte Compatibilidad y mantenimiento.

AWS Encryption SDK versiones 2.0. x y versiones posteriores ofrecen nuevas funciones de seguridad para ayudar a proteger sus datos. Sin embargo, AWS Encryption SDK la versión 2.0. x incluye cambios importantes que no son compatibles con versiones anteriores. Para garantizar una transición segura, comience por migrar de su versión actual a la última versión 1.x en su lenguaje de programación. Cuando su última versión 1.x está completamente implementada y funciona correctamente, puede migrar de forma segura a las versiones 2.0.x y versiones posteriores. Este proceso de dos pasos es fundamental, especialmente para las aplicaciones distribuidas.

Para obtener más información sobre las características de AWS Encryption SDK seguridad que sustentan estos cambios, consulte el artículo Mejora del cifrado del lado del cliente: compromiso explícito KeyIds y clave en el AWS blog sobre seguridad.

¿Buscas ayuda para usar el? SDK de cifrado de AWS para Java AWS SDK for Java 2.x Consulte Requisitos previos.

Temas

- ¿Cómo migrar e implementar el AWS Encryption SDK
- Actualización de los proveedores de claves AWS KMS maestras
- Actualización de los AWS KMS llaveros
- Establecer su política de compromiso
- Solución de problemas de migración a las versiones más recientes

¿Cómo migrar e implementar el AWS Encryption SDK

Al migrar desde una AWS Encryption SDK versión anterior a la 1.7. x a la versión 2.0. x o posterior, debe realizar la transición al cifrado de forma segura con un compromiso de clave. De lo contrario, la aplicación encontrará textos cifrados que no podrá descifrar. Si utiliza proveedores de claves AWS KMS maestras, debe actualizar a nuevos constructores que creen proveedores de claves maestras en modo estricto o modo de descubrimiento.

> **Note**
>
> Este tema está diseñado para los usuarios que migran de versiones anteriores del AWS Encryption SDK a la versión 2.0.x o posterior. Si es la primera vez que utiliza el AWS Encryption SDK, puede empezar a utilizar inmediatamente la última versión disponible con la configuración predeterminada.

Para evitar una situación crítica en la que no pueda descifrar el texto cifrado que necesita leer, le recomendamos que migre e implemente en varias etapas distintas. Compruebe que cada etapa esté completa y completamente implementada antes de comenzar la siguiente. Esto es particularmente importante en el caso de las aplicaciones distribuidas con varios hosts.

Etapa 1: actualice su aplicación a la última versión 1.x

Actualice a la versión más reciente 1.x para su lenguaje de programación. Pruébelo detenidamente, implemente los cambios y confirme que la actualización se ha propagado a todos los hosts de destino antes de comenzar la fase 2.

> **Important**
>
> Verifique que su última versión 1.x es 1.7.x o una versión posterior de AWS Encryption SDK.

La última 1. Las versiones x AWS Encryption SDK son compatibles con versiones anteriores AWS Encryption SDK y posteriores con las versiones 2.0. x y versiones posteriores. Incluyen las características nuevas que están presentes en la versión 2.0.x, pero incluyen valores predeterminados seguros diseñados para esta migración. Permiten actualizar sus proveedores de claves AWS KMS maestras, si es necesario, e implementarlos completamente con conjuntos de algoritmos que pueden descifrar el texto cifrado con un compromiso clave.

- Sustituya los elementos obsoletos, incluidos los constructores de proveedores de claves maestras heredados de AWS KMS . En Python, asegúrese de activar las advertencias de obsolescencia. Elementos de código que están en desuso en la última versión 1.x se han eliminado de las versiones 2.0.x y versiones posteriores.
- Establezca explícitamente su política de compromiso en `ForbidEncryptAllowDecrypt`. Aunque este es el único valor válido en el último número 1. x versiones, esta configuración es necesaria cuando se utiliza la versión APIs introducida en esta versión. Evita que su aplicación rechace el texto cifrado sin compromiso clave al migrar a la versión 2.0.x y versiones posteriores. Para obtener más información, consulte the section called “Establecer su política de compromiso”.
- Si utiliza proveedores de claves AWS KMS maestras, debe actualizar los proveedores de claves maestras antiguos a proveedores de claves maestras que admitan el modo estricto y el modo de descubrimiento. Esta actualización es necesaria para la SDK de cifrado de AWS para Java SDK de cifrado de AWS para Python, y la CLI de AWS cifrado. Si utiliza proveedores de claves maestras en modo de detección, le recomendamos que implemente el filtro de detección que limite las claves de encapsulamiento utilizadas a esas claves en particular Cuentas de AWS. Esta actualización es opcional, pero es una práctica recomendada. Para obtener más información, consulte Actualización de los proveedores de claves AWS KMS maestras.
- Si utiliza conjuntos de claves de detección de AWS KMS, le recomendamos que incluya un filtro de detección que limite las claves de encapsulamiento utilizadas en el descifrado a aquellas en particular Cuentas de AWS. Esta actualización es opcional, pero es una práctica recomendada. Para obtener más información, consulte Actualización de los AWS KMS llaveros.

Etapa 2: actualice la aplicación a la versión más reciente

Después de implementar la última versión 1.x correctamente en todos los hosts, puede actualizarla a la versión 2.0.x y versiones posteriores. Versión 2.0. x incluye cambios importantes en todas las versiones anteriores de AWS Encryption SDK. Sin embargo, si realiza los cambios de código recomendados en la fase 1, puede evitar errores al migrar a la última versión.

Antes de actualizar a la última versión, compruebe que su política de compromiso esté establecida de forma coherente en `ForbidEncryptAllowDecrypt`. A continuación, dependiendo de su configuración de datos, puede migrar a su propio ritmo a `RequireEncryptAllowDecrypt` y luego a la configuración predeterminada, `RequireEncryptRequireDecrypt`. Recomendamos una serie de pasos de transición como el siguiente patrón.

1. Comience con su política de compromiso establecida en `ForbidEncryptAllowDecrypt`. El AWS Encryption SDK puede descifrar los mensajes con un compromiso clave, pero aún no lo hace con un compromiso clave.
2. Cuando esté listo, actualice su política de compromisos a `RequireEncryptAllowDecrypt`. AWS Encryption SDK Comienza a cifrar sus datos con un compromiso clave. Puede descifrar texto cifrado con y sin compromiso clave.

   Antes de actualizar su política de compromiso a`RequireEncryptAllowDecrypt`, compruebe que su última versión 1.x está implementada en todos los hosts, incluidos los servidores de cualquier aplicación que descifre el texto cifrado que genere. Versiones AWS Encryption SDK anteriores a la 1.7. x no puede descifrar los mensajes cifrados con un compromiso de clave.

   También es un buen momento para añadir métricas a su aplicación para medir si sigue procesando texto cifrado sin compromiso clave. Esto lo ayudará a determinar cuándo es seguro actualizar la configuración de su política de compromiso a `RequireEncryptRequireDecrypt`. En el caso de algunas aplicaciones, como las que cifran los mensajes de una cola de Amazon SQS, esto puede implicar esperar el tiempo suficiente para volver a cifrar o eliminar todo el texto cifrado en versiones anteriores. En el caso de otras aplicaciones, como los objetos S3 cifrados, es posible que tenga que descargar, volver a cifrar y volver a cargar todos los objetos.
3. Cuando esté seguro de que no tiene ningún mensaje cifrado sin un compromiso clave, puede actualizar su política de compromiso a `RequireEncryptRequireDecrypt`. Este valor garantiza que sus datos estén siempre cifrados y descifrados con el compromiso clave. Esta configuración es la predeterminada, por lo que no es necesario que la establezca de forma explícita, pero le recomendamos que la configure. Una configuración explícita ayudará a la depuración y a cualquier posible reversión que pueda ser necesaria si su aplicación encuentra texto cifrado sin compromiso clave.

Actualización de los proveedores de claves AWS KMS maestras

Para migrar a la última versión 1. a la versión x y AWS Encryption SDK, a continuación, a la versión 2.0. x o posterior, debe sustituir los proveedores de claves AWS KMS maestras antiguos por

proveedores de claves maestras creados explícitamente en modo estricto o modo de descubrimiento. Los proveedores de claves maestras antiguos están en desuso en la versión 1.7.x y se eliminaron en la versión 2.0.x. Este cambio es necesario para las aplicaciones y los scripts que utilizan el SDK de cifrado de AWS para Java, el SDK de cifrado de AWS para Python, y la CLI de cifrado de AWS. Los ejemplos de esta sección le mostrarán cómo actualizar el código.

> **Note**
>
> En Python, active las advertencias de obsolescencia. Esto le ayudará a identificar las partes del código que necesita actualizar.

Si utiliza una clave AWS KMS maestra (no un proveedor de claves maestras), puede omitir este paso. AWS KMS las claves maestras no están en desuso ni se han eliminado. Solo cifran y descifran con las claves de encapsulamiento que usted especifique.

Los ejemplos de esta sección se centran en los elementos del código que debe cambiar. Para ver un ejemplo completo del código actualizado, consulta la sección de ejemplos del GitHub repositorio correspondiente a tu lenguaje de programación. Además, estos ejemplos suelen utilizar la clave ARNs para representar AWS KMS keys. Al crear un proveedor de claves maestras para el cifrado, puede utilizar cualquier identificador de AWS KMS clave válido para representar un AWS KMS key . Al crear un proveedor de claves maestras para el descifrado, debe utilizar un ARN de clave.

Más información sobre migración

Para todos los AWS Encryption SDK usuarios, obtén información sobre cómo establecer tu política de compromiso enthe section called “Establecer su política de compromiso”.

Para SDK de cifrado de AWS para C los SDK de cifrado de AWS para JavaScript usuarios finales, infórmese sobre una actualización opcional de los llaveros. Actualización de los AWS KMS llaveros

Temas

- Migración al modo estricto
- Migración al modo de detección

Migración al modo estricto

Después de actualizar a la última versión 1. versión x de AWS Encryption SDK, sustituya sus proveedores de claves maestras antiguos por proveedores de claves maestras en modo estricto.

En el modo estricto, debe especificar las claves de encapsulamiento que se utilizarán al cifrar y descifrar. AWS Encryption SDK Utiliza solo las claves de empaquetado que especifique. Los proveedores de claves maestras en desuso pueden descifrar los datos utilizando cualquier AWS KMS key sistema que cifre una clave de datos, incluso AWS KMS keys en diferentes regiones Cuentas de AWS .

Los proveedores de claves maestras en modo estricto se introducen en la AWS Encryption SDK versión 1.7. x. Sustituyen a los proveedores de claves maestras antiguos, que están en desuso en la versión 1.7.x y se eliminaron en la versión 2.0.x. Utilizar proveedores de claves maestras en modo estricto es una AWS Encryption SDK buena práctica.

El código siguiente crea un proveedor de claves maestras en modo estricto que puede utilizar para cifrar y descifrar.

Java

Este ejemplo representa el código de una aplicación que utiliza la versión 1.6.2 o anterior de SDK de cifrado de AWS para Java.

Este código usa el `KmsMasterKeyProvider.builder()` método para crear una instancia de un proveedor de claves AWS KMS maestras que usa una AWS KMS key como clave de empaquetado.

```
// 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();
```

Este ejemplo representa el código de una aplicación que usa la versión 1.7.x o una versión posterior de SDK de cifrado de AWS para Java . Para ver un ejemplo completo, consulta BasicEncryptionExample .java.

Los métodos `Builder.build()` y `Builder.withKeysForEncryption()` utilizados en el ejemplo anterior están obsoletos en la versión 1.7.x y se han eliminado de la versión 2.0.x.

Para actualizar a un proveedor de claves maestras en modo estricto, este código reemplaza las llamadas a métodos obsoletos por una llamada al método `Builder.buildStrict()`

nuevo. En este ejemplo se especifica una AWS KMS key como clave de ajuste, pero el `Builder.buildStrict()` método puede tomar una lista de múltiples AWS KMS keys.

```
// Create a master key provider in strict mode
// Replace the example key ARN with a valid one from your Cuenta de AWS.
String awsKmsKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

KmsMasterKeyProvider masterKeyProvider = KmsMasterKeyProvider.builder()
    .buildStrict(awsKmsKey);
```

Python

Este ejemplo representa el código de una aplicación que utiliza la versión 1.4.1 de SDK de cifrado de AWS para Python. Este código usa `KMSMasterKeyProvider`, que está obsoleto en la versión 1.7.x y eliminado de la versión 2.0.x. Al descifrar, utiliza cualquier clave AWS KMS key que haya cifrado una clave de datos sin tener en cuenta la AWS KMS keys que especifique.

Tenga en cuenta que `KMSMasterKey` no está en desuso ni se ha eliminado. Al cifrar y descifrar, utiliza únicamente la que usted especifique. 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]
)
```

Este ejemplo representa el código de una aplicación que usa la versión 1.7.x de SDK de cifrado de AWS para Python. Para ver un ejemplo completo, consulte basic_encryption.py.

Para actualizar a un proveedor de claves maestras en modo estricto, este código reemplaza la llamada a `KMSMasterKeyProvider()` por una llamada a `StrictAwsKmsMasterKeyProvider()`.

```
# Create a master key provider in strict mode
# Replace the example key ARNs with valid values from your Cuenta de AWS
```

```
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

En este ejemplo se muestra cómo cifrar y descifrar mediante la AWS CLI de cifrado, versión 1.1.7 o anterior.

En la versión 1.1.7 y anteriores, al cifrar, se especifican una o más claves maestras (o claves de encapsulamiento), como una AWS KMS key. Al descifrar, no puede especificar ninguna clave de encapsulamiento a menos que utilice un proveedor de claves maestras personalizado. La CLI de AWS cifrado puede usar cualquier clave de empaquetado que cifre una clave de datos.

```
\\ 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 .
```

En este ejemplo se muestra cómo cifrar y descifrar mediante la versión 1.7 de la AWS CLI de cifrado. x o posterior. Para ver ejemplos completos, consulte Ejemplos de la CLI AWS de cifrado.

El parámetro `--master-keys` está obsoleto en la versión 1.7.x y se eliminó en la versión 2.0.x. Se ha sustituido por el parámetro `--wrapping-keys`, que es obligatorio en los comandos de

cifrado y descifrado. Este parámetro admite el modo estricto y el modo de detección. El modo estricto es una práctica AWS Encryption SDK recomendada que garantiza que utilice la clave de embalaje deseada.

Para actualizar al modo estricto, utilice el atributo key del `--wrapping-keys` parámetro para especificar una clave de encapsulamiento al cifrar y descifrar.

```
\\ 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 .
```

Migración al modo de detección

A partir de la versión 1.7. x, se recomienda utilizar AWS Encryption SDK el modo estricto para los proveedores de claves AWS KMS maestras, es decir, especificar las claves de empaquetado al cifrar y descifrar. Siempre debe especificar las claves de encapsulamiento al cifrar. Sin embargo, hay situaciones en las que no es práctico especificar ARNs la clave AWS KMS keys para el descifrado. Por ejemplo, si utiliza alias para identificar AWS KMS keys al cifrar, pierde la ventaja de los alias si tiene que incluir la clave al descifrar. ARNs Además, dado que los proveedores de claves maestras en el modo de detección se comportan igual que los proveedores de claves maestras originales, puede utilizarlos temporalmente como parte de su estrategia de migración y, posteriormente, pasarse a proveedores de claves maestras en modo estricto.

En casos como este, puede utilizar los proveedores de claves maestras en el modo de detección. Estos proveedores de claves maestras no permiten especificar claves de encapsulamiento, por lo

que no puede utilizarlas para cifrar. Al descifrar, pueden usar cualquier clave de encapsulamiento que cifre una clave de datos. Sin embargo, a diferencia de los proveedores de claves maestras tradicionales, que se comportan de la misma manera, se crean en modo de detección de forma explícita. Al utilizar proveedores de claves maestras en modo de detección, puede limitar las claves de ajuste que se pueden usar para aquellas en particular Cuentas de AWS. Este filtro de detección es opcional, pero es una práctica recomendada. Para obtener información sobre particiones y cuentas de AWS , consulte Nombres de recursos de Amazon en Referencia general de AWS.

En los ejemplos siguientes se crea un proveedor de clave AWS KMS maestra en modo estricto para el cifrado y un proveedor de clave AWS KMS maestra en modo de detección para descifrar. El proveedor de claves maestras en el modo de detección utiliza un filtro de detección para limitar las claves de encapsulamiento utilizadas para descifrar a la `aws` partición y a un ejemplo concreto Cuentas de AWS. Si bien el filtro de cuentas no es necesario en este ejemplo tan sencillo, es una práctica recomendada que resulta muy beneficiosa cuando una aplicación cifra los datos y otra los descifra.

Java

Este ejemplo representa el código de una aplicación que usa la versión 1.7.x o una versión posterior de SDK de cifrado de AWS para Java. Para ver un ejemplo completo, consulte DiscoveryDecryptionExample .java.

Para crear una instancia de un proveedor de claves maestras en modo estricto para el cifrado, en este ejemplo se utiliza el método `Builder.buildStrict()`. Para crear una instancia de un proveedor de claves maestras en modo de detección para el descifrado, se utiliza el método `Builder.buildDiscovery()`. El `Builder.buildDiscovery()` método utiliza una `DiscoveryFilter` que limita el AWS Encryption SDK to AWS KMS keys en la AWS partición y las cuentas especificadas.

```
// Create a master key provider in strict mode for encrypting
// Replace the example alias ARN with a valid one from your Cuenta de AWS.
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

Este ejemplo representa el código de una aplicación que usa la versión 1.7.x o una versión posterior de SDK de cifrado de AWS para Python . Para ver un ejemplo completo, consulte: discovery_kms_provider.py.

Para crear un proveedor de claves maestras en modo estricto para el cifrado, en este ejemplo se utiliza el `StrictAwsKmsMasterKeyProvider`. Para crear un proveedor de claves maestras en modo de descubrimiento para el descifrado, utiliza `DiscoveryAwsKmsMasterKeyProvider` un `DiscoveryFilter` que limita el AWS Encryption SDK to AWS KMS keys en la AWS partición y las cuentas especificadas.

```
# Create a master key provider in strict mode
# Replace the example key ARN and alias ARNs with valid values from your Cuenta de
 AWS.
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

En este ejemplo se muestra cómo cifrar y descifrar mediante la versión 1.7 de la AWS CLI de cifrado. x o posterior. A partir de la versión 1.7.x, el parámetro `--wrapping-keys` es obligatorio

para cifrar y descifrar. El parámetro `--wrapping-keys` admite el modo estricto y el modo de detección. Para ver ejemplos completos, consulte the section called “Ejemplos”.

Al cifrar, en este ejemplo se especifica una clave de encapsulamiento, que es obligatoria. Al descifrar, elige explícitamente el modo de detección mediante el atributo `discovery` del parámetro `--wrapping-keys` con un valor de `true`.

Para limitar las claves de empaquetado que AWS Encryption SDK pueden usar en el modo de detección a aquellas en particular Cuentas de AWS, en este ejemplo se utilizan los `discovery-account` atributos `discovery-partition` y del `--wrapping-keys` parámetro. Estos atributos opcionales solo son válidos cuando el atributo `discovery` está establecido en `true`. Debe utilizar los atributos `discovery-partition` y `discovery-account` juntos; ninguno de los dos es válido por sí solo.

```
\\ 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 .
```

Actualización de los AWS KMS llaveros

AWS KMS Los anillos de claves de SDK de cifrado de AWS para C, los de .NET y el AWS Encryption SDK son SDK de cifrado de AWS para JavaScriptcompatibles con las mejores prácticas, ya que permiten especificar claves de empaquetado al cifrar y descifrar. Si crea un conjunto de claves de detección de AWS KMS, lo hace de forma explícita.

> **Note**
>
> La versión más antigua de AWS Encryption SDK para .NET es la 3.0. x. Todas las AWS Encryption SDK versiones de.NET son compatibles con las mejores prácticas de seguridad introducidas en la versión 2.0. x de AWS Encryption SDK. Puede actualizar de forma segura a la última versión sin cambios de código o datos.

Al actualizar a la última versión 1. versión x de AWS Encryption SDK, puede utilizar un filtro de descubrimiento para limitar las claves de empaquetado que un conjunto de claves de AWS KMS descubrimiento o un conjunto de claves de descubrimiento AWS KMS regional utiliza al descifrar a esas en particular. Cuentas de AWSSe recomienda filtrar un conjunto de claves de detección. AWS Encryption SDK

Los ejemplos de esta sección le mostrarán cómo añadir el filtro de detección a un conjunto de claves de detección regional AWS KMS .

Obtenga más información sobre la migración

Para todos los AWS Encryption SDK usuarios, infórmese sobre cómo establecer su política de compromiso. the section called “Establecer su política de compromiso”

Para SDK de cifrado de AWS para Java SDK de cifrado de AWS para Python, y los usuarios de CLI de AWS cifrado, infórmese sobre la actualización necesaria para los proveedores de claves maestras enthe section called “Actualización de los proveedores de claves AWS KMS maestras”.

Es posible que tenga un código como el siguiente en su aplicación. En este ejemplo, se crea un conjunto de claves de detección regional AWS KMS que solo puede usar claves de encapsulamiento en la región del oeste de EE. UU. (Oregón) (us-west-2). Este ejemplo representa el código de las

AWS Encryption SDK versiones anteriores a la 1.7. x. Sin embargo, sigue siendo válido en las versiones 1.7.x y versiones posteriores.

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 })
```

A partir de la versión 1.7. x, puede añadir un filtro de detección a cualquier conjunto de claves de AWS KMS detección. Este filtro de detección limita las AWS KMS keys que se AWS Encryption SDK pueden utilizar para el descifrado a las de la partición y las cuentas especificadas. Antes de usar este código, cambie la partición, si es necesario, y sustituya la cuenta IDs de ejemplo por una válida.

C

Para ver un ejemplo completo, consulte: 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

Para ver un ejemplo completo, consulte: 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' }
})
```

Establecer su política de compromiso

El compromiso clave garantiza que sus datos cifrados siempre se descifren en el mismo texto no cifrado. Para proporcionar esta propiedad de seguridad, a partir de la versión 1.7. x, AWS Encryption SDK utiliza nuevos conjuntos de algoritmos con un compromiso clave. Para determinar si sus datos están cifrados y descifrados con un compromiso clave, utilice la opción de configuración de la política de compromiso. Cifrar y descifrar los datos con un compromiso clave es una AWS Encryption SDK práctica recomendada.

Establecer una política de compromiso es una parte importante del segundo paso del proceso de migración: migrar desde la última versión. x versiones de las dos AWS Encryption SDK versiones 2.0. x y versiones posteriores. Tras configurar y cambiar la política de compromiso, asegúrese de

probar minuciosamente la aplicación antes de implementarla en producción. Para obtener orientación sobre la migración, consulte¿Cómo migrar e implementar el AWS Encryption SDK.

La configuración de la política de compromiso tiene tres valores válidos en las versiones 2.0.x y versiones posteriores. En las últimas versiones 1.x (empezando por la versión 1.7.x), solo `ForbidEncryptAllowDecrypt` es válida.

- `ForbidEncryptAllowDecrypt`— AWS Encryption SDK No se puede cifrar con un compromiso de clave. Puede descifrar textos cifrados, cifrados con o sin compromiso clave.

  En las últimas versiones 1.x, este es el único valor válido. Garantiza que no se cifre con compromiso clave hasta que esté totalmente preparado para descifrar con compromiso clave. Si se establece este valor de forma explícita, se evita que la política de compromiso se modifique automáticamente a `require-encrypt-require-decrypt` cuando actualice su versión a la versión 2.0.x o posterior. En su lugar, puede migrar su política de compromisos por etapas.
- `RequireEncryptAllowDecrypt`— AWS Encryption SDK Siempre cifra con un compromiso de clave. Puede descifrar textos cifrados, cifrados con o sin compromiso clave. Este valor se añade en la versión 2.0.x.
- `RequireEncryptRequireDecrypt`— AWS Encryption SDK Siempre cifra y descifra con compromiso de clave. Este valor se añade en la versión 2.0.x. Es el valor predeterminado en las versiones 2.0.x y posteriores.

En las últimas versiones 1.x, el único valor válido de la política de compromiso es `ForbidEncryptAllowDecrypt`. Después de migrar hasta la versión 2.0.x o una versión posterior, puede cambiar su política de compromisos por etapas a medida que esté preparado. No actualice su política de compromisos a `RequireEncryptRequireDecrypt` hasta que esté seguro de que no tiene ningún mensaje cifrado sin un compromiso clave.

Estos ejemplos le muestran cómo establecer su política de compromiso en la última versión 1.x y en las versiones 2.0.x y versiones posteriores. La técnica depende del lenguaje de programación.

Más información sobre la migración

Para SDK de cifrado de AWS para Java SDK de cifrado de AWS para Python, y la CLI de AWS cifrado, obtenga información sobre los cambios necesarios en los proveedores de claves maestras enthe section called “Actualización de los proveedores de claves AWS KMS maestras”.

Para SDK de cifrado de AWS para C y SDK de cifrado de AWS para JavaScript, infórmese sobre una actualización opcional de los llaveros. Actualización de los AWS KMS llaveros

¿Cómo establecer su política de compromiso?

La técnica que se utiliza para establecer la política de compromiso varía ligeramente en función de la implementación en cada lenguaje. En estos ejemplos de .NET puede ver cómo: Antes de cambiar su política de compromiso, revise el enfoque de varias etapas que se incluye en Cómo migrar e implementar.

C

A partir de la versión 1.7. x SDK de cifrado de AWS para C, utiliza la `aws_cryptosdk_session_set_commitment_policy` función para establecer la política de compromiso en sus sesiones de cifrado y descifrado. La política de compromiso que establezca se aplica a todas las operaciones de cifrado y descifrado realizadas en esa sesión.

Las funciones `aws_cryptosdk_session_new_from_keyring` y `aws_cryptosdk_session_new_from_cmm` están en desuso en la versión 1.7.x y se eliminaron en la versión 2.0.x. Estas funciones se sustituyen por funciones `aws_cryptosdk_session_new_from_keyring_2` y `aws_cryptosdk_session_new_from_cmm_2` que devuelven una sesión.

Cuando usa `aws_cryptosdk_session_new_from_keyring_2` y `aws_cryptosdk_session_new_from_cmm_2` en las últimas versiones 1.x, debe llamar a la función `aws_cryptosdk_session_set_commitment_policy` con el valor de la política de compromiso `COMMITMENT_POLICY_FORBID_ENCRYPT_ALLOW_DECRYPT`. En versiones 2.0.x y versiones posteriores, llamar a esta función es opcional y toma todos los valores válidos. La política de compromiso predeterminada para las versiones 2.0.x y versiones posteriores es `COMMITMENT_POLICY_REQUIRE_ENCRYPT_REQUIRE_DECRYPT`.

Para obtener un ejemplo completo, consulte 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

El `require-encrypt-require-decrypt` valor es la política de compromiso predeterminada en todas las versiones de AWS Encryption SDK para .NET. Puede configurarlo de forma explícita como práctica recomendada, pero no es necesario establecerlo. Sin embargo, si utiliza para.NET AWS Encryption SDK para descifrar el texto cifrado en otro idioma de la implementación AWS Encryption SDK sin compromiso clave, debe cambiar el valor de la política de compromiso a o. `REQUIRE_ENCRYPT_ALLOW_DECRYPT` `FORBID_ENCRYPT_ALLOW_DECRYPT` De lo contrario, el intento de descifrar el texto cifrado devolverá un error.

En el AWS Encryption SDK caso de.NET, se establece la política de compromiso en una instancia de. AWS Encryption SDK Cree una instancia de un `AwsEncryptionSdkConfig` objeto con un `CommitmentPolicy` parámetro y utilice el objeto de configuración para crear la AWS Encryption SDK instancia. A continuación, llama a los `Decrypt()` métodos `Encrypt()` y de la instancia configurada AWS Encryption SDK .

En este ejemplo, se establece la política de compromiso en `require-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

Para establecer una política de compromiso en la CLI de AWS cifrado, utilice el `--commitment-policy` parámetro. Este parámetro se introduce en la versión 1.8.x.

En la última versión 1.x, cuando se utiliza el parámetro `--wrapping-keys` en un comando `--encrypt` o `--decrypt`, se requiere un parámetro `--commitment-policy` con el valor `forbid-encrypt-allow-decrypt`. De lo contrario, el parámetro `--commitment-policy` no es válido.

En las versiones 2.1.x y versiones posteriores, el parámetro `--commitment-policy` es opcional y tiene el valor predeterminado `require-encrypt-require-decrypt`, que no cifrará ni descifrará ningún texto cifrado sin el compromiso clave. Sin embargo, recomendamos establecer la política de compromiso de forma explícita en todas las llamadas de cifrado y descifrado para facilitar el mantenimiento y la solución de problemas.

En este ejemplo se establece la política de compromiso. También usa el parámetro `--wrapping-keys` que reemplaza al parámetro `--master-keys` que comienza en la versión 1.8.x. Para obtener más información, consulte the section called “Actualización de los proveedores de claves AWS KMS maestras”. Para ver ejemplos completos, consulte Ejemplos de la CLI AWS de cifrado.

```
\\ 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

A partir de la versión 1.7. x SDK de cifrado de AWS para Java, estableces la política de compromiso en tu instancia del `AwsCrypto` objeto que representa al AWS Encryption SDK cliente. Esta configuración de política de compromiso se aplica a todas las operaciones de cifrado y descifrado realizadas en ese cliente.

El `AwsCrypto()` constructor está en desuso en la última versión. x versiones de SDK de cifrado de AWS para Java y se elimina en la versión 2.0. x. Se sustituye por una nueva clase `Builder`, un método `Builder.withCommitmentPolicy()` y el tipo `CommitmentPolicy` enumerado.

En las últimas versiones 1.x, la clase `Builder` requiere el método `Builder.withCommitmentPolicy()` y el argumento `CommitmentPolicy.ForbidEncryptAllowDecrypt`. A partir de la versión 2.0.x, el método `Builder.withCommitmentPolicy()` es opcional; el valor predeterminado es `CommitmentPolicy.RequireEncryptRequireDecrypt`.

Para ver un ejemplo completo, consulte 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

A partir de la versión 1.7. x del SDK de cifrado de AWS para JavaScript, puede establecer la política de compromiso al llamar a la nueva `buildClient` función que crea una instancia de un AWS Encryption SDK cliente. La función `buildClient` toma un valor enumerado que representa su política de compromiso. Devuelve funciones actualizadas `encrypt` y `decrypt` que hacen cumplir su política de compromiso al cifrar y descifrar.

En las últimas versiones 1.x, la función `buildClient` requiere el argumento `CommitmentPolicy.FORBID_ENCRYPT_ALLOW_DECRYPT`. A partir de la versión 2.0.x, el argumento de la política de compromiso es opcional y el valor predeterminado es `CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT`.

El código de Node.js y el navegador son idénticos para este propósito, excepto que el navegador necesita una declaración para establecer las credenciales.

En el siguiente ejemplo, se cifran los datos con un anillo de claves. AWS KMS La nueva función `buildClient` establece la política de compromiso en `FORBID_ENCRYPT_ALLOW_DECRYPT`, el valor predeterminado en las últimas versiones 1.x. Las funciones actualizadas `encrypt`

y `decrypt` que `buildClient` devuelve hacen cumplir la política de compromiso que haya establecido.

```
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

A partir de la versión 1.7. x SDK de cifrado de AWS para Python, estableces la política de compromiso en tu instancia de`EncryptionSDKClient`, un nuevo objeto que representa al AWS Encryption SDK cliente. La política de compromiso que establezca se aplica a todas las llamadas a `encrypt` y `decrypt` que usen esa instancia del cliente.

En las últimas versiones 1x, el constructor `EncryptionSDKClient` requiere el valor `CommitmentPolicy.FORBID_ENCRYPT_ALLOW_DECRYPT` enumerado. A partir de la versión 2.0.x, el argumento de la política de compromiso es opcional y el valor predeterminado es `CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT`.

En este ejemplo, se utiliza el nuevo constructor `EncryptionSDKClient` y se establece la política de compromiso en 1.7.x valor predeterminado. El constructor crea una instancia de un cliente que representa el AWS Encryption SDK. Cuando llama a los métodos `encrypt`, `decrypt` o `stream` de este cliente, estos aplican la política de compromiso que haya establecido. En este ejemplo también se utiliza el nuevo constructor de la `StrictAwsKmsMasterKeyProvider` clase, que especifica AWS KMS keys cuándo se cifra y se descifra.

Para ver un ejemplo completo, consulte set_commitment.py.

```
# Instantiate the client
client =
 aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.FORBID_ENCRYPT_AL

// 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
)
```

Rust

El `require-encrypt-require-decrypt` valor es la política de compromiso predeterminada en todas las versiones de Rust AWS Encryption SDK . Puede configurarlo de forma explícita como práctica recomendada, pero no es necesario establecerlo. Sin embargo, si utiliza la implementación AWS Encryption SDK de Rust AWS Encryption SDK sin compromiso de clave cifrada cifrada en otro idioma, tendrá que cambiar el valor de la política de compromiso a o. `REQUIRE_ENCRYPT_ALLOW_DECRYPT` `FORBID_ENCRYPT_ALLOW_DECRYPT` De lo contrario, el intento de descifrar el texto cifrado devolverá un error.

En el caso AWS Encryption SDK de Rust, se establece la política de compromiso en una instancia de. AWS Encryption SDK Cree una instancia de un `AwsEncryptionSdkConfig` objeto con un `comitment_policy` parámetro y utilice el objeto de configuración para crear la AWS Encryption SDK instancia. A continuación, llama a los `Decrypt()` métodos `Encrypt()` y de la instancia configurada AWS Encryption SDK .

En este ejemplo, se establece la política de compromiso en `forbid-encrypt-allow-decrypt`.

```
// Configure the commitment policy on the AWS Encryption SDK instance
let esdk_config = AwsEncryptionSdkConfig::builder()
                    .commitment_policy(ForbidEncryptAllowDecrypt)
                    .build()?;
let esdk_client = esdk_client::Client::from_conf(esdk_config)?;

// Create an AWS KMS client
let sdk_config =
 aws_config::load_defaults(aws_config::BehaviorVersion::latest()).await;
let kms_client = aws_sdk_kms::Client::new(&sdk_config);

// Create your encryption context
let encryption_context = HashMap::from([
    ("encryption".to_string(), "context".to_string()),
    ("is not".to_string(), "secret".to_string()),
    ("but adds".to_string(), "useful metadata".to_string()),
    ("that can help you".to_string(), "be confident that".to_string()),
    ("the data you are handling".to_string(), "is what you think it
 is".to_string()),
]);

// Instantiate the material providers library
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;

// Create an AWS KMS keyring
let kms_keyring = mpl
    .create_aws_kms_keyring()
    .kms_key_id(kms_key_id)
    .kms_client(kms_client)
    .send()
    .await?;

// Encrypt your plaintext data
let plaintext = example_data.as_bytes();

let encryption_response = esdk_client.encrypt()
    .plaintext(plaintext)
    .keyring(kms_keyring.clone())
    .encryption_context(encryption_context.clone())
```

```
    .send()
    .await?;

// Decrypt your ciphertext
let decryption_response = esdk_client.decrypt()
    .ciphertext(ciphertext)
    .keyring(kms_keyring)
    // Provide the encryption context that was supplied to the encrypt method
    .encryption_context(encryption_context)
    .send()
    .await?;
```

Go

```
import (
    "context"

 mpl "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygenerated"
 mpltypes "aws/aws-cryptographic-material-providers-library/releases/go/mpl/
awscryptographymaterialproviderssmithygeneratedtypes"
 client "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygenerated"
 esdktypes "github.com/aws/aws-encryption-sdk/
awscryptographyencryptionsdksmithygeneratedtypes"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/kms"
)

// Instantiate the AWS Encryption SDK client
commitPolicyForbidEncryptAllowDecrypt :=
 mpltypes.ESDKCommitmentPolicyForbidEncryptAllowDecrypt
encryptionClient, err :=
 client.NewClient(esdktypes.AwsEncryptionSdkConfig{CommitmentPolicy:
 &commitPolicyForbidEncryptAllowDecrypt})
if err != nil {
    panic(err)
}

// Create an AWS KMS client
cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    panic(err)
```

```
}
kmsClient := kms.NewFromConfig(cfg, func(o *kms.Options) {
    o.Region = KmsKeyRegion
})

// Optional: Create an encryption context
encryptionContext := map[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 material providers library
matProv, err := mpl.NewClient(mpltypes.MaterialProvidersConfig{})
if err != nil {
    panic(err)
}

// Create an AWS KMS keyring
awsKmsKeyringInput := mpltypes.CreateAwsKmsKeyringInput{
    KmsClient: kmsClient,
    KmsKeyId:  kmsKeyId,
}
awsKmsKeyring, err := matProv.CreateAwsKmsKeyring(context.Background(),
 awsKmsKeyringInput)
if err != nil {
    panic(err)
}

// Encrypt your plaintext data
res, err := forbidEncryptClient.Encrypt(context.Background(),
 esdktypes.EncryptInput{
    Plaintext:         []byte(exampleText),
    EncryptionContext: encryptionContext,
    Keyring:           awsKmsKeyring,
})
if err != nil {
    panic(err)
}

// Decrypt your ciphertext
```

```
decryptOutput, err := forbidEncryptClient.Decrypt(context.Background(),
 esdktypes.DecryptInput{
    Ciphertext:        res.Ciphertext,
    EncryptionContext: encryptionContext,
    Keyring:           awsKmsKeyring,
})
if err != nil {
    panic(err)
}
```

Solución de problemas de migración a las versiones más recientes

Antes de actualizar la aplicación a la versión 2.0. x o posterior del AWS Encryption SDK, actualice a la última versión 1. x versión de AWS Encryption SDK e impleméntela por completo. Esto le ayudará a evitar la mayoría de los errores que puedan producirse al actualizar a las versiones 2.0.x y versiones posteriores. Para obtener una guía detallada, incluidos ejemplos, consulte Migrando su AWS Encryption SDK.

Important

Verifique que su última versión 1.x es 1.7.x o una versión posterior de AWS Encryption SDK.

Note

AWS CLI de cifrado: referencias en esta guía a la versión 1.7. x de la AWS Encryption SDK se aplica a la versión 1.8. x de la CLI AWS de cifrado. Referencias en esta guía a la versión 2.0. x del AWS Encryption SDK se aplica a la 2.1. x de la CLI AWS de cifrado.
Las nuevas funciones de seguridad se publicaron originalmente en las versiones 1.7 de AWS Encryption CLI. x y 2.0. x. Sin embargo, AWS Encryption CLI versión 1.8. x reemplaza a la versión 1.7. x y CLI de AWS cifrado 2.1. x reemplaza a 2.0. x. Para obtener más información, consulte el aviso de seguridad correspondiente en el aws-encryption-sdk-clirepositorio de GitHub.

Este tema está diseñado para ayudarlo a reconocer y resolver los errores más comunes que pueda encontrar.

Temas

- Objetos obsoletos o eliminados
- Conflicto de configuración: conjunto de políticas y algoritmos de compromiso
- Conflicto de configuración: política de compromiso y texto cifrado
- Falla en la validación del compromiso clave
- Otras fallas de cifrado
- Otras fallas de descifrado
- Consideraciones de restauración

Objetos obsoletos o eliminados

Versión 2.0.x incluye varios cambios importantes, como la eliminación de constructores, métodos, funciones y clases antiguos que estaban en desuso en la versión 1.7.x. Para evitar errores del compilador, errores de importación, errores de sintaxis y errores de símbolos no encontrados (según el lenguaje de programación), actualice primero a la última versión. x versión de la AWS Encryption SDK para su lenguaje de programación. (La versión debe ser 1.7.x o posterior). Mientras usa la última versión 1.x, puede empezar a utilizar los elementos de reemplazo antes de eliminar los símbolos originales.

Si necesita actualizar a la versión 2.0.x o posterior inmediatamente, consulte el registro de cambios de su lenguaje de programación y sustituya los símbolos antiguos por los símbolos recomendados en el registro de cambios.

Conflicto de configuración: conjunto de políticas y algoritmos de compromiso

Si especifica un conjunto de algoritmos que entra en conflicto con su política de compromiso, la llamada al cifrado fallará y aparecerá un error de conflicto de configuración.

Para evitar este tipo de error, no especifique un conjunto de algoritmos. De forma predeterminada, AWS Encryption SDK elige el algoritmo más seguro que sea compatible con su política de compromiso. Sin embargo, si debe especificar un conjunto de algoritmos, por ejemplo, uno sin firmar, asegúrese de elegir un conjunto de algoritmos que sea compatible con su política de compromiso.

Política de compromiso	Compatibilidad con los conjuntos de algoritmos
ForbidEncryptAllowDecrypt	Cualquier conjunto de algoritmos sin compromiso clave, como: `AES_256_GCM_IV12_TAG16_HKDF_SHA384_ECDSA_P384` (03 78) (con firma) `AES_256_GCM_IV12_TAG16_HKDF_SHA256` (01 78) (sin firma)
RequireEncryptAllowDecrypt RequireEncryptRequireDecrypt	Cualquier conjunto de algoritmos con un compromiso clave, como: `AES_256_GCM_HKDF_SHA512_COMMIT_KEY_ECDSA_P384` (05 78) (con firma) `AES_256_GCM_HKDF_SHA512_COMMIT_KEY` (04 78) (sin firma)

Si encuentra este error cuando no ha especificado un conjunto de algoritmos, es posible que su administrador de materiales criptográficos (CMM) haya elegido el conjunto de algoritmos conflictivo. El CMM predeterminado no seleccionará un conjunto de algoritmos conflictivo, pero sí un CMM personalizado. Para obtener ayuda, consulte la documentación de su CMM personalizado.

Conflicto de configuración: política de compromiso y texto cifrado

La política de compromiso de `RequireEncryptRequireDecrypt` no permite a AWS Encryption SDK descifrar un mensaje que se haya cifrado sin el compromiso clave. Si le pide que descifre un mensaje sin confirmar la clave, devolverá un error de conflicto de configuración. AWS Encryption SDK

Para evitar este error, antes de establecer la política de compromiso de `RequireEncryptRequireDecrypt`, asegúrese de que todos los textos cifrados sin compromiso clave se descifren y se vuelvan a cifrar con compromiso clave, o de que sean gestionados por otra aplicación. Si se produce este error, puede arrojar un error para el texto cifrado conflictivo o cambiar temporalmente la política de compromiso a `RequireEncryptAllowDecrypt`.

Si se produce este error porque actualizó a la versión 2.0.x o posterior desde una versión anterior a la 1.7.x sin actualizar primero a la última versión 1.x (versión 1.7.x o posterior), considere la

posibilidad de volver a la última versión 1.x e implementar esa versión en todos los hosts antes de actualizar a la versión 2.0.x o posterior. Para obtener ayuda, consulte ¿Cómo migrar e implementar el AWS Encryption SDK.

Falla en la validación del compromiso clave

Al descifrar los mensajes cifrados con un compromiso clave, es posible que aparezca un mensaje de error en la validación del compromiso clave. Esto indica que la llamada de descifrado ha fallado porque la clave de datos de un mensaje cifrado no es idéntica a la clave de datos única del mensaje. Al validar la clave de datos durante el descifrado, el compromiso clave evita que se descodifique un mensaje que pueda tener como resultado más de un texto no cifrado.

Este error indica que el mensaje cifrado que estaba intentando descifrar no fue devuelto por el AWS Encryption SDK. Puede ser un mensaje creado manualmente o el resultado de una corrupción de datos. Si se produce este error, la aplicación puede rechazar el mensaje y continuar o dejar de procesar los mensajes nuevos.

Otras fallas de cifrado

El cifrado puede fallar por varios motivos. No puede utilizar un conjunto de claves de detección de AWS KMS ni un proveedor de claves maestras en el modo de detección para cifrar un mensaje.

Asegúrese de especificar un conjunto de claves o un proveedor de claves maestras con claves de encapsulamiento que tenga permiso para usar para el cifrado. Para obtener ayuda sobre los permisos AWS KMS keys, consulte Visualización de una política clave y Determinación del acceso a una AWS KMS key en la Guía para AWS Key Management Service desarrolladores.

Otras fallas de descifrado

Si se produce un error al intentar descifrar un mensaje cifrado, significa que AWS Encryption SDK no ha podido (o no quiere) descifrar ninguna de las claves de datos cifrados del mensaje.

Si ha utilizado un conjunto de claves o un proveedor de claves maestras que especifique las claves de empaquetado, AWS Encryption SDK utilizará únicamente las claves de empaquetado que especifique. Compruebe que está utilizando las claves de encapsulamiento que desea utilizar y que tiene `kms:Decrypt` permiso para utilizar al menos una de las claves de encapsulamiento. Si lo utiliza AWS KMS keys, como alternativa, puede intentar descifrar el mensaje con un anillo de claves de AWS KMS detección o un proveedor de claves maestras en modo de detección. Si la operación

se realiza correctamente, antes de devolver el texto no cifrado, compruebe que la clave utilizada para descifrar el mensaje es de confianza.

Consideraciones de restauración

Si su aplicación no logra cifrar o descifrar los datos, normalmente puede resolver el problema actualizando los símbolos de código, los conjuntos de claves, los proveedores de claves maestras o la política de compromiso. Sin embargo, en algunos casos, puede que decidas que es mejor revertir la aplicación a una versión anterior de AWS Encryption SDK.

Si tiene que revertirla, hágalo con precaución. Versiones anteriores a la 1.7 AWS Encryption SDK . x no puede descifrar el texto cifrado con un compromiso de clave.

- Retroceder desde la última versión 1.x a una versión anterior de la AWS Encryption SDK es generalmente seguro. Puede que tenga que deshacer los cambios realizados en el código para utilizar símbolos y objetos que no son compatibles con las versiones anteriores.
- Una vez que haya empezado a cifrar con un compromiso clave (estableciendo su política de compromiso en `RequireEncryptAllowDecrypt`) en la versión 2.0.x o posterior, puede volver a la versión 1.7.x, pero no a una versión anterior. Versiones anteriores a la 1.7 AWS Encryption SDK . x no puede descifrar el texto cifrado con un compromiso de clave.

Si habilita accidentalmente el cifrado con compromiso clave antes de que todos los hosts puedan descifrar con compromiso clave, sería mejor continuar con la implementación en lugar de revertirla. Si los mensajes son transitorios o se pueden eliminar de forma segura, podría considerar la posibilidad de revertirlos con la pérdida de mensajes. Si es necesaria una reversión, podría considerar la posibilidad de crear una herramienta que descifre y vuelva a cifrar todos los mensajes.

Preguntas frecuentes

Preguntas frecuentes

- ¿En qué se AWS Encryption SDK diferencia del AWS SDKs?
- ¿En qué se AWS Encryption SDK diferencia del cliente de cifrado Amazon S3?
- ¿Qué algoritmos criptográficos admite y cuál es el AWS Encryption SDK predeterminado?
- ¿Cómo es el vector de inicialización (IV) generado y dónde se almacena?
- ¿Cómo se genera, cifra y descifra cada clave de datos?
- ¿Cómo se realizar el seguimiento de las claves de datos que se han utilizado para cifrar los datos?
- ¿Cómo se AWS Encryption SDK almacenan las claves de datos cifrados con sus datos cifrados?
- ¿Cuánta sobrecarga añade el formato del AWS Encryption SDK mensaje a mis datos cifrados?
- ¿Puedo utilizar mi propio proveedor de claves maestras?
- ¿Puedo cifrar datos con más de una clave de encapsulamiento?
- ¿Qué tipos de datos puedo cifrar con el? AWS Encryption SDK
- ¿Cómo se AWS Encryption SDK cifran y descifran las transmisiones input/output (E/S)?

¿En qué se AWS Encryption SDK diferencia del AWS SDKs?

AWS SDKsProporcionan bibliotecas para interactuar con Amazon Web Services (AWS), incluidas AWS Key Management Service (AWS KMS). Algunas de las implementaciones lingüísticas del AWS Encryption SDK, como la AWS Encryption SDK de .NET, siempre requieren que el AWS SDK esté en el mismo lenguaje de programación. Las implementaciones de otros lenguajes requieren el AWS SDK correspondiente solo cuando utilizas AWS KMS claves en tus llaveros o proveedores de claves maestras. Para obtener más información, consulte el tema sobre su lenguaje de programación en AWS Encryption SDK lenguajes de programación.

Puede utilizarlos AWS SDKs para interactuar con ellos AWS KMS, por ejemplo, para cifrar y descifrar pequeñas cantidades de datos (hasta 4 096 bytes con una clave de cifrado simétrica) y para generar claves de datos para el cifrado del lado del cliente. Sin embargo, al generar una clave de datos, debe gestionar todo el proceso de cifrado y descifrado, incluido el cifrado de los datos con la clave de datos externa, el descarte de forma segura la clave de datos sin formato AWS KMS, el almacenamiento de la clave de datos cifrada y, a continuación, el descifrado de la clave de datos y el descifrado de los datos. El AWS Encryption SDK se encarga de este proceso por usted.

AWS Encryption SDK Proporciona una biblioteca que cifra y descifra los datos según los estándares y las mejores prácticas del sector. Genera la clave de datos, la cifra con las claves de encapsulamiento que especifique y devuelve un mensaje cifrado, un objeto de datos portátil que incluye los datos cifrados y las claves de datos cifrados que necesita para descifrarlos. Cuando llega el momento de descifrar, se pasa el mensaje cifrado y al menos una de las claves de empaquetado (opcional), y se AWS Encryption SDK devuelven los datos en texto plano.

Se puede utilizar AWS KMS keys como envoltorio de claves AWS Encryption SDK, pero no es obligatorio. Puede usar las claves de cifrado que genere y las del administrador de claves o del módulo de seguridad de hardware en las instalaciones. Puedes usarlo AWS Encryption SDK incluso si no tienes una AWS cuenta.

¿En qué se AWS Encryption SDK diferencia del cliente de cifrado Amazon S3?

El cliente de cifrado Amazon S3 del AWS SDKs proporciona cifrado y descifrado para los datos que almacene en Amazon Simple Storage Service (Amazon S3). Estos clientes están estrechamente acoplados a Amazon S3 y se han diseñado para utilizarlos exclusivamente con los datos almacenados en este servicio.

AWS Encryption SDK Proporciona cifrado y descifrado para los datos que puede almacenar en cualquier lugar. El cliente de cifrado Amazon S3 AWS Encryption SDK y el cliente de cifrado no son compatibles porque producen textos cifrados con diferentes formatos de datos.

¿Qué algoritmos criptográficos admite y cuál es el AWS Encryption SDK predeterminado?

AWS Encryption SDK Utiliza el algoritmo simétrico en Galois/Counter modo (GCM) del Estándar de Cifrado Avanzado (AES), conocido como AES-GCM, para cifrar los datos. Le permite elegir entre varios algoritmos simétricos y asimétricos para cifrar las claves de datos que cifran sus datos.

En el caso del AES-GCM, el conjunto de algoritmos predeterminado es el AES-GCM, con una clave de 256 bits, una derivación de claves (HKDF), firmas digitales y un compromiso de claves. AWS Encryption SDK también admite claves de cifrado y algoritmos de cifrado de 192 y 128 bits sin firmas digitales ni compromiso de claves.

En todos los casos, la longitud del vector de inicialización (IV) siempre es de 12 bytes; la longitud de la etiqueta de autenticación siempre es de 16 bytes. De forma predeterminada, el SDK utiliza la clave

de datos como entrada en la función de derivación de claves basada en HMAC (HKDF) para obtener la extract-and-expand clave de cifrado AES-GCM, y también añade una firma con un algoritmo de firma digital de curva elíptica (ECDSA).

Para obtener información sobre qué algoritmo utilizar, consulte Conjuntos de algoritmos admitidos.

Para obtener más información sobre la implementación de los algoritmos compatibles, consulte Referencia de algoritmos.

¿Cómo es el vector de inicialización (IV) generado y dónde se almacena?

AWS Encryption SDK Utiliza un método determinista para construir un valor IV diferente para cada fotograma. Este procedimiento garantiza que nunca IVs se repitan en un mensaje. (Antes de la versión 1.3.0 de SDK de cifrado de AWS para Java y el SDK de cifrado de AWS para Python, generaban AWS Encryption SDK aleatoriamente un valor IV único para cada fotograma).

El IV se almacena en el mensaje cifrado que AWS Encryption SDK devuelven. Para obtener más información, consulte la AWS Encryption SDK referencia de formato de mensaje.

¿Cómo se genera, cifra y descifra cada clave de datos?

El método depende del conjunto de claves o proveedor de claves maestras que utilice.

Los conjuntos de AWS KMS claves y los proveedores de claves maestras de la AWS Encryption SDK red utilizan la operación AWS KMS GenerateDataKeyAPI para generar cada clave de datos y cifrarla con su clave de empaquetado. Para cifrar copias de la clave de datos en claves KMS adicionales, utilizan la AWS KMS operación de cifrado. Para descifrar las claves de datos, utilizan la operación de descifrado. AWS KMS Para obtener más información, consulte el conjunto de AWS KMS claves en la especificación de AWS Encryption SDK . GitHub

Otros conjuntos de claves generan la clave de datos, la cifran y la descifran utilizando los métodos de las prácticas recomendadas para cada lenguaje de programación. Para obtener más información, consulte la especificación del conjunto de claves o del proveedor de claves maestras en la sección Marco de la AWS Encryption SDK especificación en. GitHub

¿Cómo se realizar el seguimiento de las claves de datos que se han utilizado para cifrar los datos?

Lo AWS Encryption SDK hace por ti. Cuando se cifran datos, el SDK cifra la clave de datos y almacena la clave cifrada junto con los datos cifrados en el mensaje cifrado que devuelve. Al descifrar los datos, el AWS Encryption SDK extrae la clave de datos cifrada del mensaje cifrado, la descifra y luego la usa para descifrar los datos.

¿Cómo se AWS Encryption SDK almacenan las claves de datos cifrados con sus datos cifrados?

Las operaciones de cifrado AWS Encryption SDK devuelven un mensaje cifrado, una estructura de datos única que contiene los datos cifrados y sus claves de datos cifrados. El formato del mensaje consta de al menos dos partes: un encabezado y un cuerpo. El encabezado del mensaje contiene las claves de datos cifradas e información sobre la composición del cuerpo del mensaje. El cuerpo del mensaje contiene los datos cifrados. Si el conjunto de algoritmos incluye una firma digital, el formato del mensaje incluye un pie de página que contiene la firma. Para obtener más información, consulte AWS Encryption SDK referencia de formato de mensaje.

¿Cuánta sobrecarga añade el formato del AWS Encryption SDK mensaje a mis datos cifrados?

La cantidad de gastos adicionales que añada AWS Encryption SDK depende de varios factores, entre los que se incluyen los siguientes:

- El tamaño de los datos en texto no cifrado.
- Cuál de los algoritmos admitidos se utiliza.
- Si se proporciona información autenticada adicional (AAD) y cuál es la longitud de esa AAD.
- El número y el tipo de claves de encapsulamiento o claves maestras
- El tamaño de la trama (cuando se utilizan datos con tramas).

Si se utiliza AWS Encryption SDK con su configuración predeterminada (una AWS KMS key como clave de empaquetado (o clave maestra), sin AAD, datos no estructurados y un algoritmo de cifrado con firma), la sobrecarga es de aproximadamente 600 bytes. En general, es razonable suponer que

el AWS Encryption SDK agrega una sobrecarga de 1 KB o menos, sin incluir la AAD proporcionada. Para obtener más información, consulte AWS Encryption SDK referencia de formato de mensaje.

¿Puedo utilizar mi propio proveedor de claves maestras?

Sí. Los detalles de implementación varían en función del lenguaje de programación compatible que utilice. Sin embargo, todos los lenguajes compatibles permiten definir gestores de materiales criptográficos (MCMMs) personalizados, proveedores de claves maestras, llaveros, claves maestras y claves de empaquetado.

¿Puedo cifrar datos con más de una clave de encapsulamiento?

Sí. Es posible cifrar la clave de datos con claves de encapsulamiento (o claves maestras) adicionales con el fin de aportar redundancia en caso de que una clave se encuentre en una región diferente o no esté disponible para el descifrado.

Para cifrar datos con varias claves de encapsulamiento, cree un conjunto de claves o un proveedor de claves maestras con varias claves de encapsulamiento. Cuando trabaje con conjuntos de claves, puede crear un solo conjunto de claves con varias claves de encapsulamiento o un conjunto de claves múltiple.

Cuando se cifran datos con varias claves de empaquetado, AWS Encryption SDK utiliza una clave de empaquetado para generar una clave de datos de texto simple. La clave de datos es única y no está relacionada matemáticamente con la clave de encapsulamiento. Esta operación devuelve una copia de texto no cifrado de la clave de datos y una copia de la clave de datos que está cifrada por la clave de encapsulamiento. Luego, el método de cifrado cifra la clave de datos con las demás claves de encapsulamiento. El mensaje cifrado obtenido incluye los datos cifrados y una clave de datos cifrada para cada clave de encapsulamiento.

El mensaje cifrado se puede descifrar mediante cualquiera de las claves de empaquetado utilizadas en la operación de cifrado. AWS Encryption SDK Utiliza una clave de empaquetado para descifrar una clave de datos cifrados. Luego, usa la clave de datos de texto no cifrado para descifrar los datos.

¿Qué tipos de datos puedo cifrar con el? AWS Encryption SDK

La mayoría de las implementaciones de lenguajes de programación AWS Encryption SDK pueden cifrar bytes sin procesar (matrices de bytes), I/O flujos (flujos de bytes) y cadenas. El AWS

Encryption SDK for.NET no admite transmisiones. I/O Proporcionamos ejemplos de código para cada uno de los lenguajes de programación admitidos.

¿Cómo se AWS Encryption SDK cifran y descifran las transmisiones input/output (E/S)?

AWS Encryption SDK Crea un flujo de cifrado o descifrado que envuelve un flujo subyacente. I/O La secuencia de cifrado o descifrado realiza una operación criptográfica en una llamada de lectura o escritura. Por ejemplo, puede leer datos en texto no cifrado en la secuencia subyacente y cifrarlos antes de devolver el resultado. O puede leer texto cifrado en una secuencia subyacente y descifrarlo antes de devolver el resultado. Proporcionamos ejemplos de código para cifrar y descifrar secuencias en cada uno de los lenguajes de programación compatibles con las secuencias.

El AWS Encryption SDK para .NET no admite transmisiones. I/O

AWS Encryption SDK referencia

La información de esta página le servirá de referencia para crear su propia biblioteca de cifrado compatible con el AWS Encryption SDK. Si no está creando su propia biblioteca de cifrado compatible, es probable que no necesite esta información.

Para utilizarla AWS Encryption SDK en uno de los lenguajes de programación compatibles, consulteLenguajes de programación.

Para ver la especificación que define los elementos de una AWS Encryption SDK implementación adecuada, consulte la AWS Encryption SDK especificación en GitHub.

AWS Encryption SDK Utiliza los algoritmos compatibles para devolver una única estructura de datos o mensaje que contiene datos cifrados y las claves de datos cifrados correspondientes. En los siguientes temas se explican los algoritmos y la estructura de datos. Utilice esta información para crear bibliotecas que puedan leer y escribir textos cifrados que sean compatibles con este SDK.

Temas

- AWS Encryption SDK referencia de formato de mensaje
- AWS Encryption SDK ejemplos de formatos de mensajes
- Referencia de datos autenticados adicionales (AAD) del cuerpo para el AWS Encryption SDK
- AWS Encryption SDK referencia de algoritmos
- AWS Encryption SDK referencia vectorial de inicialización
- AWS KMS Detalles técnicos del llavero jerárquico

AWS Encryption SDK referencia de formato de mensaje

La información de esta página le servirá de referencia para crear su propia biblioteca de cifrado compatible con el AWS Encryption SDK. Si no está creando su propia biblioteca de cifrado compatible, es probable que no necesite esta información.

Para utilizarla AWS Encryption SDK en uno de los lenguajes de programación compatibles, consulteLenguajes de programación.

Para ver la especificación que define los elementos de una AWS Encryption SDK implementación adecuada, consulte la AWS Encryption SDK especificación en GitHub.

Las operaciones de cifrado AWS Encryption SDK devuelven una estructura de datos única o un mensaje cifrado que contiene los datos cifrados (texto cifrado) y todas las claves de datos cifrados. Para comprender esta estructura de datos o crear bibliotecas que la lean y escriban, es preciso comprender el formato del mensaje.

El formato del mensaje consta de al menos dos partes: un encabezado y un cuerpo. En algunos casos, el formato del mensaje incluye una tercera parte, un pie de página. El formato del mensaje define una secuencia ordenada de bytes en orden de bytes de red, también denominado formato big-endian. El formato del mensaje comienza con el encabezado, que va seguido del cuerpo y, a continuación, del pie de página (cuando lo hay).

Los conjuntos de algoritmos que admite el AWS Encryption SDK utilizan una de las dos versiones de formato de mensaje. Los conjuntos de algoritmos sin compromiso clave utilizan el formato de mensaje de la versión 1. Los conjuntos de algoritmos sin compromiso clave utilizan el formato de mensaje de la versión 2.

Temas

- Estructura del encabezado
- Estructura del cuerpo
- Estructura del pie de página

Estructura del encabezado

El encabezado del mensaje contiene la clave de datos cifrada e información sobre la composición del cuerpo del mensaje. En la siguiente tabla se describen los campos que forman el encabezado en las versiones 1 y 2 del formato de mensaje. Los bytes se anexan en el orden mostrado

El valor No presente indica que el campo no existe en esa versión del formato del mensaje. El texto en negrita indica valores que son diferentes en cada versión.

> **Note**
>
> Es posible que tenga que desplazarse horizontal o verticalmente para ver todos los datos de esta tabla.

Estructura del encabezado

Campo	Formato de mensaje versión 1 Longitud (bytes)	Formato de mensaje versión 2 Longitud (bytes)
Version	1	1
Type	1	No presente
Algorithm ID	2	2.
Message ID	16	32
AAD Length	2 Cuando el contexto de cifrado está vacío, el valor del campo Longitud AAD de 2 bytes es 0.	2 Cuando el contexto de cifrado está vacío, el valor del campo Longitud AAD de 2 bytes es 0.
AAD	Variable. La longitud de este campo aparece en los 2 bytes anteriores (campo de Longitud AAD). Cuando el contexto de cifrado está vacío, no hay ningún campo AAD en el encabezad o.	Variable. La longitud de este campo aparece en los 2 bytes anteriores (campo de Longitud AAD). Cuando el contexto de cifrado está vacío, no hay ningún campo AAD en el encabezad o.
Encrypted Data Key Count	2	2.
Encrypted Data Key(s)	Variable. Se determina según el número de claves de datos cifradas y la longitud de cada una de ellas.	Variable. Se determina según el número de claves de datos cifradas y la longitud de cada una de ellas.
Content Type	1	1
Reserved	4	No presente

Campo	Formato de mensaje versión 1 Longitud (bytes)	Formato de mensaje versión 2 Longitud (bytes)
IV Length	1	No presente
Frame Length	4	4
Algorithm Suite Data	No presente	Variable. Se determina mediante el algoritmo que generó el mensaje.
Header Authentication	Variable. Se determina mediante el algoritmo que generó el mensaje.	Variable. Se determina mediante el algoritmo que generó el mensaje.

Versión

La versión de este formato de mensaje. La versión es de 1 o 2, que se codifica como el byte `01` o `02` en notación hexadecimal.

Tipo

El tipo de este formato de mensaje. El tipo indica el tipo de estructura. El único tipo admitido es el de datos cifrados autenticados por el cliente. El valor del tipo es 128, codificado como el byte `80` en notación hexadecimal.

Este campo no está presente en la versión 2 del formato de mensaje.

ID de algoritmo

Un identificador del algoritmo utilizado. Se trata de un valor de 2 bytes interpretado como un entero sin signo de 16 bits. Para obtener más información sobre los algoritmos, consulte AWS Encryption SDK referencia de algoritmos.

ID del mensaje

Un valor generado de manera aleatoria que identifica el mensaje. El campo Message ID:

- identifica de forma exclusiva el mensaje cifrado;
- establece un vínculo débil entre el encabezado del mensaje y el cuerpo del mensaje;

- proporciona un mecanismo para reutilizar una clave de datos de forma segura con varios mensajes cifrados;
- protege contra la reutilización accidental de una clave de datos o el desgaste de las claves en el AWS Encryption SDK.

Este valor es de 128 bits en la versión 1 del formato de mensaje y de 256 bits en la versión 2.

Longitud del AAD

La longitud de la información autenticada adicional (AAD). Se trata de un valor de 2 bytes interpretado como un entero sin signo de 16 bits que especifica el número de bytes que contiene la AAD.

Cuando el contexto de cifrado está vacío, el valor del campo Longitud AAD es 0.

AAD

La información autenticada adicional. AAD es una codificación del contexto de cifrado, una matriz de pares de clave-valor donde cada clave y valor es una cadena de caracteres con codificación UTF-8. El contexto de cifrado se convierte en una secuencia de bytes y se utiliza para el valor de AAD. Cuando el contexto de cifrado está vacío, no hay ningún campo AAD en el encabezado.

Cuando se utilizan algoritmos con firma, el contexto de cifrado debe contener el par de clave-valor `{'aws-crypto-public-key', Qtxt}`. Qtxt representa el punto Q de la curva elíptica comprimido según SEC 1 versión 2.0 y, a continuación, codificado en base64. El contexto de cifrado puede contener valores adicionales, pero la longitud máxima de la AAD construida es de 2 ^ 16 - 1 bytes.

En la siguiente tabla se describen los campos que componen la AAD. Los pares de clave-valor están ordenados, por clave, en orden ascendente de acuerdo con el código de caracteres UTF-8. Los bytes se anexan en el orden mostrado

Estructura de AAD

Campo	Longitud (bytes)
Key-Value Pair Count	2
Key Length	2.
Key	Variable. Equivalente al valor especificado en los últimos 2 bytes (Key Length).

Campo	Longitud (bytes)
Value Length	2
Value	Variable. Equivalente al valor especificado en los últimos 2 bytes (Value Length).

Recuento de pares clave-valor

El número de pares de clave-valor de la AAD. Se trata de un valor de 2 bytes interpretado como un entero sin signo de 16 bits que especifica el número de pares de clave-valor de la AAD. El número máximo de pares de clave-valor de la AAD es 2^16 - 1.

Cuando no hay ningún contexto de cifrado o está vacío, este campo no está presente en la estructura de AAD.

Longitud de clave

La longitud de la clave para el par de clave-valor. Se trata de un valor de 2 bytes interpretado como un entero sin signo de 16 bits que especifica el número de bytes que contienen la clave.

Clave

La clave en el par de clave-valor. Se trata de una secuencia de bytes codificados con UTF-8.

Longitud del valor

La longitud del valor en el par de clave-valor. Se trata de un valor de 2 bytes interpretado como un entero sin signo de 16 bits que especifica el número de bytes que contienen el valor.

Valor

El valor en el par de clave-valor. Se trata de una secuencia de bytes codificados con UTF-8.

Recuento de claves de datos cifrados

El número de claves de datos cifradas. Se trata de un valor de 2 bytes interpretado como un entero sin signo de 16 bits que especifica el número de claves de datos cifradas. El número máximo de claves de datos cifrados en cada mensaje es 65 535 (2^16 - 1).

Clave (s) de datos cifrados

Una secuencia de claves de datos cifradas. La longitud de la secuencia se determina según el número de claves de datos cifradas y la longitud de cada una de ellas. La secuencia contiene al menos una clave de datos cifrada.

En la siguiente tabla se describen los campos que componen cada clave de datos cifrada. Los bytes se anexan en el orden mostrado

Encrypted Data Key Structure

Campo	Longitud (bytes)
Key Provider ID Length	2
Key Provider ID	Variable. Equivalente al valor especificado en los últimos 2 bytes (Key Provider ID Length).
Key Provider Information Length	2
Key Provider Information	Variable. Equivalente al valor especificado en los últimos 2 bytes (Key Provider Information Length).
Encrypted Data Key Length	2
Encrypted Data Key	Variable. Equivalente al valor especificado en los últimos 2 bytes (Encrypted Data Key Length).

Longitud del identificador del proveedor de claves

La longitud del identificador del proveedor de claves. Se trata de un valor de 2 bytes interpretado como un entero sin signo de 16 bits que especifica el número de bytes que contienen el ID del proveedor de claves.

ID de proveedor clave

El identificador del proveedor de claves. Se utiliza para indicar el proveedor de la clave de datos cifrada y está previsto que sea extensible.

Longitud de la información clave del proveedor

La longitud de la información del proveedor de claves. Se trata de un valor de 2 bytes interpretado como un entero sin signo de 16 bits que especifica el número de bytes que contienen la información del proveedor de claves.

Información clave del proveedor

La información del proveedor de claves. Viene determinada por el proveedor de claves.

Si AWS KMS es el proveedor de la clave maestra o utiliza un AWS KMS anillo de claves, este valor contiene el nombre de recurso de Amazon (ARN) del. AWS KMS key

Longitud de la clave de datos cifrados

La longitud de la clave de datos cifrada. Se trata de un valor de 2 bytes interpretado como un entero sin signo de 16 bits que especifica el número de bytes que contienen la clave de datos cifrada.

Clave de datos cifrados

La clave de datos cifrada. Se trata de la clave de cifrado de datos cifrada por el proveedor de claves.

Tipo de contenido

El tipo de contenido cifrado, que puede ser con o sin trama.

> **Note**
>
> Siempre que sea posible, utilice datos con trama. AWS Encryption SDK Admite datos no enmarcados solo para uso heredado. Algunas implementaciones lingüísticas del aún AWS Encryption SDK pueden generar texto cifrado sin marcos. Todas las implementaciones de idiomas compatibles pueden descifrar texto cifrado con trama y sin trama.

Los datos con trama están divididos en partes con la misma longitud; cada una de ellas está cifrada por separado. El contenido con trama es de tipo 2, que se codifica como el byte `02` en notación hexadecimal.

Los datos sin trama no se dividen; se trata de un único bloque cifrado. El contenido sin trama es de tipo 1, que se codifica como el byte `01` en notación hexadecimal.

Reservado

Una secuencia reservada de 4 bytes. Este valor debe ser 0. Se codifica como los bytes `00 00 00 00` en notación hexadecimal (es decir, una secuencia de 4 bytes de un valor entero de 32 bits igual a 0).

Este campo no está presente en la versión 2 del formato de mensaje.

Longitud IV

La longitud del vector de inicialización (IV). Se trata de un valor de 1 byte interpretado como un entero sin signo de 8 bits que especifica el número de bytes que contienen el IV. Este valor se determina en función del valor en bytes del IV del algoritmo que generó el mensaje.

Este campo no está presente en la versión 2 del formato de mensaje, que solo admite conjuntos de algoritmos que utilizan valores IV deterministas en el encabezado del mensaje.

Longitud del marco

La longitud de cada trama de datos con trama. Se trata de un valor de 4 byte interpretado como un entero sin signo de 32 bits que especifica el número de bytes en cada trama. Cuando el contenido es sin trama, es decir, cuando el valor del campo de tipo de `Content Type` es 1 este valor debe ser 0.

> **Note**
>
> Siempre que sea posible, utilice datos con trama. AWS Encryption SDK Admite datos no enmarcados solo para uso heredado. Algunas implementaciones lingüísticas del aún AWS Encryption SDK pueden generar texto cifrado sin marcos. Todas las implementaciones de idiomas compatibles pueden descifrar texto cifrado con trama y sin trama.

Conjunto de datos de algoritmos

Datos complementarios que necesita el algoritmo que generó el mensaje. El algoritmo determina la longitud y el contenido. Su longitud puede ser 0.

Este campo no está presente en la versión 1 del formato de mensaje.

Autenticación de

La autenticación del encabezado viene determinada por el algoritmo que generó el mensaje. La autenticación del encabezado se calcula para todo el encabezado. Se compone de un IV y una etiqueta de autenticación. Los bytes se anexan en el orden mostrado

Estructura de Header Authentication

Campo	Longitud en la versión 1.0 (bytes)	Longitud en la versión 2.0 (bytes)
IV	Variable. Se determina en función del valor en bytes del IV del algoritmo que generó el mensaje.	N/A
Authentication Tag	Variable. Se determina en función del valor en bytes de la etiqueta de autenticación del algoritmo que generó el mensaje.	Variable. Se determina en función del valor en bytes de la etiqueta de autenticación del algoritmo que generó el mensaje.

IV

El vector de inicialización (IV) utilizado para calcular la etiqueta de autenticación del encabezado.

Este campo no está presente en la versión 2 del formato de encabezado del mensaje. La versión 2 del formato de mensaje solo admite conjuntos de algoritmos que utilizan valores IV deterministas en el encabezado del mensaje.

Etiqueta de autenticación

El valor de autenticación del encabezado. Se utiliza para autenticar todo el contenido del encabezado.

Estructura del cuerpo

El cuerpo del mensaje contiene los datos cifrados, denominados texto cifrado. La estructura del cuerpo depende del tipo de contenido (con o sin trama). En las secciones siguientes se describe el formato del cuerpo del mensaje para cada tipo de contenido. La estructura del cuerpo del mensaje es la misma en las versiones 1 y 2 del formato de mensaje.

Temas

- Datos sin trama

- Datos con trama

Datos sin trama

Los datos sin trama se cifran en un solo blob con un IV y un AAD de cuerpo únicos.

> **Note**
>
> Siempre que sea posible, utilice datos con trama. AWS Encryption SDK Admite datos no enmarcados solo para uso heredado. Algunas implementaciones lingüísticas del aún AWS Encryption SDK pueden generar texto cifrado sin marcos. Todas las implementaciones de idiomas compatibles pueden descifrar texto cifrado con trama y sin trama.

En la siguiente tabla se describen los campos que componen los datos sin trama. Los bytes se anexan en el orden mostrado

Estructura del cuerpo sin trama

Campo	Longitud, en bytes
IV	Variable. Equivalente al valor especificado en el byte de IV Length del encabezado.
Encrypted Content Length	8
Encrypted Content	Variable. Equivalente al valor especificado en los últimos 8 bytes (Encrypted Content Length).
Authentication Tag	Variable. Su valor lo determina la implementación del algoritmo utilizada.

IV

El vector de inicialización (IV) que se va a utilizar con el algoritmo de cifrado.

Longitud del contenido cifrado

La longitud del contenido cifrado (o texto cifrado). Se trata de un valor de 8 bytes interpretado como un entero sin signo de 64 bits que especifica el número de bytes que incluyen el contenido cifrado.

Técnicamente, el valor máximo permitido es 2^63 - 1, u 8 exbibytes (8 EiB). Sin embargo, en la práctica, el valor máximo es 2^36 - 32, o 64 gibibytes (64 GiB), debido a las restricciones impuestas por los algoritmos implementados.

> **Note**
>
> La implementación para Java de este SDK restringe aún más este valor a 2^31 - 1, o 2 gibibytes (2 GiB), debido a las restricciones del lenguaje.

Contenido cifrado

El contenido cifrado (texto cifrado) devuelto por el algoritmo de cifrado.

Etiqueta de autenticación

El valor de autenticación del cuerpo. Se utiliza para autenticar el cuerpo del mensaje.

Datos con trama

En los datos con trama, los datos en texto no cifrado se dividen en partes de igual longitud denominadas tramas. AWS Encryption SDK Cifra cada fotograma por separado con un AAD único en forma intravenosa y corporal.

> **Note**
>
> Siempre que sea posible, utilice datos con trama. AWS Encryption SDK Admite datos no enmarcados solo para uso tradicional. Algunas implementaciones lingüísticas del aún AWS Encryption SDK pueden generar texto cifrado sin marcos. Todas las implementaciones de idiomas compatibles pueden descifrar texto cifrado con trama y sin trama.

La longitud de la trama, que es la longitud del contenido cifrado en la trama, puede ser diferente para cada mensaje. El número máximo de bytes en una trama es 2^32 - 1. El número máximo de tramas en un mensaje es 2^32 - 1.

Existen dos tipos de tramas: normal y final. Cada mensaje debe consistir o incluir una trama final.

Todas las tramas normales de un mensaje tienen la misma longitud de trama. La trama final puede tener una longitud de trama diferente.

La composición de las tramas en los datos con trama varía según la longitud del contenido cifrado.

- Igual a la longitud de la trama: cuando la longitud del contenido cifrado es igual a la longitud de la trama de las tramas normales, el mensaje puede consistir en una trama normal que contenga los datos, seguido de una trama final de longitud cero (0). O bien, el mensaje puede consistir únicamente en una trama final que contenga los datos. En este caso, la trama final tiene la misma longitud que las tramas normales.
- Múltiplo de la longitud de la trama: cuando la longitud del contenido cifrado es un múltiplo exacto de la longitud de la trama de las tramas normales, el mensaje puede consistir en una trama normal que contenga los datos, seguido de una trama final de longitud cero (0). O bien, el mensaje puede terminar en una trama final que contenga los datos. En este caso, la trama final tiene la misma longitud que las tramas normales.
- No es un múltiplo de la longitud de la trama: cuando la longitud del contenido cifrado no es un múltiplo exacto de la longitud de la trama de las tramas normales, la trama final contiene los datos restantes. La longitud de la trama final es menor que la longitud de las tramas normales.
- Menor que la longitud de la trama: cuando la longitud del contenido cifrado es menor que la longitud de la trama de las tramas normales, el mensaje puede consistir en una trama final que contenga todos los datos. La longitud de la trama final es menor que la longitud de las tramas normales.

En la siguiente tabla se describen los campos que componen las tramas. Los bytes se anexan en el orden mostrado

Estructura de cuerpo con trama, trama normal

Campo	Longitud, en bytes
Sequence Number	4

Campo	Longitud, en bytes
IV	Variable. Equivalente al valor especificado en el byte de IV Length del encabezado.
Encrypted Content	Variable. Equivalente al valor especificado en el Frame Length del encabezado.
Authentication Tag	Variable. Se determina mediante el algoritmo utilizado, según lo especificado en el Algorithm ID del encabezado.

Número de secuencia

El número secuencial de la trama. Es un número de contador incremental para la trama. Se trata de un valor de 4 bytes interpretado como un entero sin signo de 32 bits.

Los datos con trama deben comenzar en el número de secuencia 1. Las tramas posteriores deben estar en orden y contener un incremento de 1 respecto a la trama anterior. De lo contrario, el proceso de descifrado se detiene y registra un error.

IV

El vector de inicialización (IV) de la trama. El SDK utiliza un método determinista para construir un IV diferente para cada trama del mensaje. Su longitud se especifica mediante el conjunto de algoritmos utilizado.

Contenido cifrado

El contenido cifrado (texto cifrado) de la trama, devuelto por el algoritmo de cifrado.

Etiqueta de autenticación

El valor de autenticación de la trama. Se utiliza para autenticar la trama completa.

Estructura de cuerpo con trama, trama final

Campo	Longitud, en bytes
Sequence Number End	4

Campo	Longitud, en bytes
Sequence Number	4
IV	Variable. Equivalente al valor especificado en el byte de IV Length del encabezado.
Encrypted Content Length	4
Encrypted Content	Variable. Equivalente al valor especificado en los últimos 4 bytes (Encrypted Content Length).
Authentication Tag	Variable. Se determina mediante el algoritmo utilizado, según lo especificado en el Algorithm ID del encabezado.

Final del número de secuencia

Un indicador de la trama final. El valor se codifica como los 4 bytes `FF FF FF FF` en notación hexadecimal.

Número de secuencia

El número secuencial de la trama. Es un número de contador incremental para la trama. Se trata de un valor de 4 bytes interpretado como un entero sin signo de 32 bits.

Los datos con trama deben comenzar en el número de secuencia 1. Las tramas posteriores deben estar en orden y contener un incremento de 1 respecto a la trama anterior. De lo contrario, el proceso de descifrado se detiene y registra un error.

IV

El vector de inicialización (IV) de la trama. El SDK utiliza un método determinista para construir un IV diferente para cada trama del mensaje. La longitud del IV se especifica mediante el conjunto de algoritmos.

Longitud del contenido cifrado

La longitud del contenido cifrado. Se trata de un valor de 4 bytes interpretado como un entero sin signo de 32 bits que especifica el número de bytes que incluyen el contenido cifrado para la trama.

Contenido cifrado

El contenido cifrado (texto cifrado) de la trama, devuelto por el algoritmo de cifrado.

Etiqueta de autenticación

El valor de autenticación de la trama. Se utiliza para autenticar la trama completa.

Estructura del pie de página

Cuando se utilizan algoritmos con firma, el formato de mensaje contiene un pie de página. El pie de página del mensaje contiene una firma digital calculada con el encabezado y el cuerpo del mensaje. En la siguiente tabla se describen los campos que componen el pie de página. Los bytes se anexan en el orden mostrado La estructura del pie de página del mensaje es la misma en las versiones 1 y 2 del formato de mensaje.

Estructura del pie de página

Campo	Longitud, en bytes
Signature Length	2
Signature	Variable. Equivalente al valor especificado en los últimos 2 bytes (Signature Length).

Longitud de la firma

La longitud de la firma. Se trata de un valor de 2 bytes interpretado como un entero sin signo de 16 bits que especifica el número de bytes que contienen la firma.

Firma

La firma.

AWS Encryption SDK ejemplos de formatos de mensajes

La información de esta página le servirá de referencia para crear su propia biblioteca de cifrado compatible con el AWS Encryption SDK. Si no está creando su propia biblioteca de cifrado compatible, es probable que no necesite esta información.

Para usarlo AWS Encryption SDK en uno de los lenguajes de programación compatibles, consulteLenguajes de programación.

Para ver la especificación que define los elementos de una AWS Encryption SDK implementación adecuada, consulte la AWS Encryption SDK especificación en GitHub.

En los temas siguientes se muestran ejemplos del formato de los AWS Encryption SDK mensajes. Cada ejemplo muestra los bytes sin procesar, en notación hexadecimal, seguidos de una descripción de lo que representan esos bytes.

Temas

- Datos con trama (formato de mensaje versión 1)
- Datos con trama (formato de mensaje versión 2)
- Datos sin trama (formato de mensaje versión 1)

Datos con trama (formato de mensaje versión 1)

En el siguiente ejemplo se muestra el formato del mensaje de los datos con trama en el formato de mensaje versión 1.

```
+--------+
| Header |
+--------+
01                                      Version (1.0)
80                                      Type (128, customer authenticated encrypted
 data)
0378                                    Algorithm ID (see Referencia de algoritmos)
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+0OkpOZ1cc8Tg2qd7rs5aLTg7lvfUEW/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                                       Content Type (2, framed data)
00000000                                 Reserved
0C                                       IV Length (12)
00000100                                 Frame Length (256)
4ECBD5C0 9899CA65 923D2347               IV
0B896144 0CA27950 CA571201 4DA58029       Authentication Tag
+------+
| Body |
+------+
00000001                                 Frame 1, Sequence Number (1)
6BD3FE9C ADBCB213 5B89E8F1               Frame 1, IV
1F6471E0 A51AF310 10FA9EF6 F0C76EDF       Frame 1, Encrypted Content
F5AFA33C 7D2E8C6C 9C5D5175 A212AF8E
FBD9A0C3 C6E3FB59 C125DBF2 89AC7939
BDEE43A8 0F00F49E ACBBD8B2 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        Frame 1, Authentication Tag
00000002                                   Frame 2, Sequence Number (2)
F1140984 FF25F943 959BE514                 Frame 2, IV
216C7C6A 2234F395 F0D2D9B9 304670BF        Frame 2, Encrypted Content
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
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        Frame 2, Authentication Tag
FFFFFFFF                                   Final Frame, Sequence Number End
00000003                                   Final Frame, Sequence Number (3)
35F74F11 25410F01 DD9E04BF                 Final Frame, IV
0000008E                                   Final Frame, Encrypted Content Length (142)
F7A53D37 2F467237 6FBD0B57 D1DFE830        Final Frame, Encrypted Content
B965AD1F A910AA5F 5EFFFFF4 BC7D431C
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        Final Frame, Authentication Tag
+--------+
| Footer |
+--------+
```

```
0066                                    Signature Length (102)
30640230 085C1D3C 63424E15 B2244448     Signature
639AED00 F7624854 F8CF2203 D7198A28
758B309F 5EFD9D5D 2E07AD0B 467B8317
5208B133 02301DF7 2DFC877A 66838028
3C6A7D5E 4F8B894E 83D98E7C E350F424
7E06808D 0FE79002 E24422B9 98A0D130
A13762FF 844D
```

Datos con trama (formato de mensaje versión 2)

En el siguiente ejemplo se muestra el formato del mensaje de los datos con trama en el formato de mensaje versión 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
 ("QXRnM3JwOEVBNFFhNnBmaTk3MUlTNTk3NHpOMnlZWE5vSmtwRHFPc0dIYkVaVDRqME5OMlFkRStmbTFVY01WdThnPT0=
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
```

Datos sin trama (formato de mensaje versión 1)

En el siguiente ejemplo se muestra el formato de mensaje de los datos sin trama.

> **Note**
>
> Siempre que sea posible, utilice datos con trama. Solo AWS Encryption SDK admite datos no enmarcados para uso tradicional. Algunas implementaciones lingüísticas del aún AWS Encryption SDK pueden generar texto cifrado sin marcos. Todas las implementaciones de idiomas compatibles pueden descifrar texto cifrado con trama y sin trama.

```
+--------+
| Header |
+--------+
01                                      Version (1.0)
80                                      Type (128, customer authenticated encrypted
 data)
0378                                    Algorithm ID (see Referencia de algoritmos)
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
 ("AsG8gG9InLPu16YKlqXTOD+nykG8YqHAhqecj8aXfD2e5B4gtVE73dZkyClA+rAMOQ==")
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                                      Content Type (1, nonframed data)
00000000                                Reserved
0C                                      IV Length (12)
00000000                                Frame Length (0, nonframed data)
734C1BBE 032F7025 84CDA9D0              IV
2C82BB23 4CBF4AAB 8F5C6002 622E886C      Authentication Tag
+------+
| Body |
+------+
D39DD3E5 915E0201 77A4AB11              IV
00000000 0000028E                       Encrypted Content Length (654)
E8B6F955 B5F22FE4 FD890224 4E1D5155      Encrypted Content
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 DAEBA4AF 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
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
```

Referencia de datos autenticados adicionales (AAD) del cuerpo para el AWS Encryption SDK

La información de esta página le servirá de referencia para crear su propia biblioteca de cifrado compatible con el AWS Encryption SDK. Si no está creando su propia biblioteca de cifrado compatible, es probable que no necesite esta información.

Para usarlo AWS Encryption SDK en uno de los lenguajes de programación compatibles, consulte. Lenguajes de programación

Para ver la especificación que define los elementos de una AWS Encryption SDK implementación adecuada, consulte la AWS Encryption SDK especificación en GitHub.

Debe proporcionar información autenticada adicional (AAD) al algoritmo AES-GCM para cada operación criptográfica. Esto es válido para datos del cuerpo con o sin trama. Para obtener más información sobre el AAD y su uso en Galois/Counter modo (GCM), consulte Recomendaciones para los modos de operación con cifrado por bloques: Galois/Counter modo (GCM) y GMAC.

En la siguiente tabla se describen los campos que componen la AAD del cuerpo. Los bytes se anexan en el orden mostrado

Estructura de la AAD del cuerpo

Campo	Longitud, en bytes
Message ID	16
Body AAD Content	Variable. Consulte Body AAD Content en la siguiente lista.
Sequence Number	4
Content Length	8

ID del mensaje

El mismo valor de Message ID establecido en el encabezado del mensaje.

Cuerpo: contenido AAD

Un valor con codificación UTF-8 determinado por el tipo de datos del cuerpo utilizados.

Para datos sin trama, utilice el valor `AWSKMSEncryptionClient Single Block`.

Para las tramas normales en los datos con trama, utilice el valor `AWSKMSEncryptionClient Frame`.

Para la trama final en los datos con trama, utilice el valor `AWSKMSEncryptionClient Final Frame`.

Número de secuencia

Un valor de 4 bytes interpretado como un entero sin signo de 32 bits.

Para los datos con trama, este es el número de secuencia de la trama.

Para datos sin trama, utilice el valor 1, codificado como `00 00 00 01` de 4 bytes en notación hexadecimal.

Longitud del contenido

La longitud, en bytes, de los datos en texto no cifrado proporcionados al algoritmo para el cifrado. Se trata de un valor de 8 bytes interpretado como un entero sin signo de 64 bits.

AWS Encryption SDK referencia de algoritmos

La información de esta página le servirá de referencia para crear su propia biblioteca de cifrado compatible con el AWS Encryption SDK. Si no está creando su propia biblioteca de cifrado compatible, es probable que no necesite esta información.

Para utilizarla AWS Encryption SDK en uno de los lenguajes de programación compatibles, consulteLenguajes de programación.

Para ver la especificación que define los elementos de una AWS Encryption SDK implementación adecuada, consulte la AWS Encryption SDK especificación en GitHub.

Si va a crear su propia biblioteca que pueda leer y escribir textos cifrados que sean compatibles con ella AWS Encryption SDK, tendrá que entender cómo AWS Encryption SDK implementa los conjuntos de algoritmos compatibles para cifrar los datos sin procesar.

AWS Encryption SDK Es compatible con los siguientes conjuntos de algoritmos. Todos los conjuntos de algoritmos AES-GCM tienen un vector de inicialización de 12 bytes y una etiqueta de autenticación AES-GCM de 16 bytes. El conjunto de algoritmos predeterminado varía según la AWS Encryption SDK versión y la política de compromiso clave seleccionada. Para obtener más información, consulte Política de compromiso y conjunto de algoritmos.

AWS Encryption SDK Conjuntos de algoritmos

Algorithm ID	Versión de formato de mensaje	Algoritmo de cifrado	Longitud de la clave de datos (bits)	Algoritmo de derivación de clave	Algoritmo de firma	Algoritmo de compromiso clave	Longitud de los datos del conjunto de algoritmos (bytes)
05 78	0x02	AES-GCM	256	HKDF con SHA-512	ECDSA con P-384 y SHA-384	HKDF con SHA-512	32 (compromiso clave)
04 78	0x02	AES-GCM	256	HKDF con SHA-512	Ninguno	HKDF con SHA-512	32 (compromiso clave)
03 78	0x01	AES-GCM	256	HKDF con SHA-384	ECDSA con P-384 y SHA-384	Ninguno	N/A
03 46	0x01	AES-GCM	192	HKDF con SHA-384	ECDSA con P-384 y SHA-384	Ninguno	N/A

Algorithm ID	Versión de formato de mensaje	Algoritmo de cifrado	Longitud de la clave de datos (bits)	Algoritmo de derivación de clave	Algoritmo de firma	Algoritmo de compromiso clave	Longitud de los datos del conjunto de algoritmos (bytes)
`02 14`	0x01	AES-GCM	128	HKDF con SHA-256	ECDSA con P-256 y SHA-256	Ninguno	N/A
`01 78`	0x01	AES-GCM	256	HKDF con SHA-256	Ninguno	Ninguno	N/A
`01 46`	0x01	AES-GCM	192	HKDF con SHA-256	Ninguno	Ninguno	N/A
`01 14`	0x01	AES-GCM	128	HKDF con SHA-256	Ninguno	Ninguno	N/A
`00 78`	0x01	AES-GCM	256	Ninguno	Ninguna	Ninguno	N/A
`00 46`	0x01	AES-GCM	192	Ninguno	Ninguna	Ninguno	N/A
`00 14`	0x01	AES-GCM	128	Ninguno	Ninguna	Ninguno	N/A

Algorithm ID

Un valor hexadecimal de 2 bytes que identifica de forma exclusiva una implementación del algoritmo. Este valor se almacena en el encabezado del mensaje del texto cifrado.

Versión de formato de mensaje

La versión del formato de mensaje. Los conjuntos de algoritmos con compromiso clave utilizan el formato de mensaje de la versión 2 (0x02). Los conjuntos de algoritmos sin compromiso clave utilizan el formato de mensaje de la versión 1 (0x01).

Longitud de los datos del conjunto de algoritmos

La longitud en bytes de los datos específicos del conjunto de algoritmos. Este campo solo se admite en la versión 2 del formato de mensaje (0x02). En la versión 2 del formato de mensaje (0x02), estos datos aparecen en el campo `Algorithm suite data` del encabezado del mensaje. Los conjuntos de algoritmos que admiten el compromiso clave utilizan 32 bytes para la cadena de compromiso clave. Para obtener más información, consulte Algoritmo de compromiso clave en esta lista.

Longitud de la clave de datos

La longitud de la clave de datos en bits. El AWS Encryption SDK admite claves de 256, 192 y 128 bits. La clave de datos se genera mediante un conjunto de claves o una clave maestra.

En algunas implementaciones, esta clave de datos se utiliza como entrada para una función de derivación de extract-and-expand claves (HKDF) basada en HMAC. El resultado de la HKDF se utiliza como clave de cifrado de datos en el algoritmo de cifrado. Para obtener más información, consulte Algoritmo de derivación de clave en esta lista.

Algoritmo de cifrado

El nombre y el modo del algoritmo de cifrado que se utilizó. Los conjuntos de algoritmos AWS Encryption SDK utilizan el algoritmo de cifrado avanzado (AES) con Galois/Counter modo (GCM).

Algoritmo de compromiso clave

El algoritmo utilizado para calcular la cadena de compromiso clave. El resultado se almacena en el campo `Algorithm suite data` del encabezado del mensaje y se utiliza para validar la clave de datos del compromiso clave.

Para obtener una explicación técnica sobre cómo añadir un compromiso clave a un conjunto de algoritmos, consulte Key Committing AEADs in Cryptology ePrint Archive.

Algoritmo de derivación de clave

La función de derivación de extract-and-expand claves basada en HMAC (HKDF) utilizada para obtener la clave de cifrado de datos. AWS Encryption SDK Utiliza la HKDF definida en el RFC 5869.

Conjuntos de algoritmos sin compromiso clave (ID de algoritmo `01xx` – `03xx`)

- La función hash que se utiliza es SHA-384 o SHA-256, según el conjunto de algoritmos.
- Para el paso de extracción:
  - No se utiliza sal. Según el RFC, la sal se establece en una cadena de ceros. La longitud de la cadena es igual a la longitud de la salida de la función hash, es decir, de 48 bytes para SHA-384 y de 32 bytes para SHA-256.
  - El material de entrada para las claves es la clave de datos del conjunto de claves o proveedor de claves maestras.
- Para el paso de expansión:
  - La clave pseudoaleatoria de entrada es el resultado del paso de extracción.
  - La información de entrada es una concatenación del ID de algoritmo seguido del ID de mensaje (en ese orden).
  - La longitud del material de salida para las claves es la Longitud de la clave de datos. Este resultado se utiliza como clave de cifrado de datos en el algoritmo de cifrado.

Conjuntos de algoritmos con compromiso clave (ID de algoritmo `04xx` – `05xx`)

- La función hash que se utiliza es SHA-512.
- Para el paso de extracción:
  - La sal es un valor aleatorio criptográfico de 256 bits. En la versión 2 del formato de mensaje (0x02), este valor se almacena en el campo `MessageID`.
  - El material inicial para las claves es la clave de datos del conjunto de claves o proveedor de claves maestras.
- Para el paso de expansión:
  - La clave pseudoaleatoria de entrada es el resultado del paso de extracción.
  - La etiqueta de clave son los bytes codificados en UTF-8 de la cadena `DERIVEKEY` en el orden de bytes big endian.
  - La información de entrada es una concatenación del ID de algoritmo seguido de la etiqueta de clave (en ese orden).
  - La longitud del material de salida para las claves es la Longitud de la clave de datos. Este resultado se utiliza como clave de cifrado de datos en el algoritmo de cifrado.

Versión de formato de mensaje

La versión del formato de mensaje utilizada con el conjunto de algoritmos. Para obtener más información, consulte Referencia de formato de mensajes.

Algoritmo de firma

El algoritmo de firma que se utiliza para generar una firma digital sobre el encabezado y el cuerpo del texto cifrado. AWS Encryption SDK Utiliza el algoritmo de firma digital de curva elíptica (ECDSA) con las siguientes especificaciones:

- La curva elíptica utilizada es la curva P-384 o P-256, según lo especificado en el ID de algoritmo. Estas curvas se definen en Digital Signature Standard (DSS) (FIPS PUB 186-4).
- La función hash que se utiliza es SHA-384 (con la curva P-384) o SHA-256 (con la curva P-256).

AWS Encryption SDK referencia vectorial de inicialización

La información de esta página le servirá de referencia para crear su propia biblioteca de cifrado compatible con el AWS Encryption SDK. Si no está creando su propia biblioteca de cifrado compatible, es probable que no necesite esta información.

Para utilizarla AWS Encryption SDK en uno de los lenguajes de programación compatibles, consulteLenguajes de programación.

Para ver la especificación que define los elementos de una AWS Encryption SDK implementación adecuada, consulte la AWS Encryption SDK especificación en GitHub.

AWS Encryption SDK Proporciona los vectores de inicialización (IVs) que requieren todos los conjuntos de algoritmos compatibles. El SDK utiliza números secuenciales con trama para construir un IV de tal forma que no puede haber dos tramas en el mismo mensaje que tengan el mismo IV.

Cada IV de 96 bits (12 bytes) se crea a partir de dos matrices de bytes big-endian concatenadas en el orden que se indica a continuación:

- 64 bits: 0 (reservado para uso futuro)
- 32 bits: número secuencial de la trama. Para la etiqueta de autenticación del encabezado, este valor es todo ceros.

Antes de introducir el almacenamiento en caché de claves de datos, AWS Encryption SDK siempre utilizaban una nueva clave de datos para cifrar cada mensaje y se generaba todo de forma aleatoria. IVs Las claves de datos generadas aleatoriamente IVs eran criptográficamente seguras porque las

claves de datos nunca se reutilizaban. Cuando el SDK introdujo el almacenamiento en caché de claves de datos, que reutiliza intencionadamente las claves de datos, cambiamos la forma en que el SDK genera. IVs

El uso del método determinista, IVs que no se puede repetir en un mensaje, aumenta considerablemente el número de invocaciones que se pueden ejecutar de forma segura con una única clave de datos. Además, las claves de datos que se almacenan en caché siempre utilizan un conjunto de algoritmos con una función de derivación de clave. El uso de una IV determinista con una función de derivación de claves pseudoaleatorias para obtener claves de cifrado a partir de una clave de datos permite cifrar 2^32 mensajes sin AWS Encryption SDK sobrepasar los límites criptográficos.

AWS KMS Detalles técnicos del llavero jerárquico

El conjunto de claves AWS KMS jerárquico utiliza una clave de datos única para cifrar cada mensaje y cifra cada clave de datos con una clave de empaquetado única derivada de una clave de rama activa. Utiliza una derivación de claves en modo contador con una función pseudoaleatoria con el HMAC SHA-256 para obtener la clave de encapsulamiento de 32 bytes con las siguientes entradas.

- Una sal de asignación al azar de 16 bytes
- La clave de rama activa
- El valor codificado en UTF-8 para el identificador del proveedor de claves "» aws-kms-hierarchy

El conjunto de claves jerárquico utiliza la clave de encapsulamiento derivada para cifrar una copia de la clave de datos de texto no cifrado mediante el AES-GCM-256 con una etiqueta de autenticación de 16 bytes y las siguientes entradas.

- La clave de encapsulamiento derivada se utiliza como clave de cifrado AES-GCM
- La clave de datos se utiliza como mensaje AES-GCM
- Se utiliza un vector de inicialización aleatoria (IV) de 12 bytes como AES-GCM IV
- Datos autenticados adicionales (AAD) que contienen los siguientes valores serializados.

Valor	Longitud en bytes	Interpretado como
"aws-kms-hierarchy"	17	Codificado con UTF-8

Valor	Longitud en bytes	Interpretado como
El identificador de la clave de la rama	Variable	Codificado con UTF-8
La versión de clave de la rama	16	Codificado con UTF-8
Contexto de cifrado	Variable	Pares de valores de clave con codificación UTF-8

Historial de documentos de la Guía para AWS Encryption SDK desarrolladores

En este tema se describen actualizaciones importantes en la Guía para desarrolladores de AWS Encryption SDK .

Temas

- Actualizaciones recientes
- Actualizaciones anteriores

Actualizaciones recientes

En la tabla siguiente, se describen los cambios importantes introducidos en esta documentación desde noviembre de 2017. Además de los cambios importantes que se indican a continuación, también actualizamos la documentación con frecuencia para mejorar las descripciones y los ejemplos y para dar cuenta de los comentarios que nos envía. Si desea recibir notificaciones sobre cambios importantes, suscríbase a la fuente RSS.

Cambio	Descripción	Fecha
Disponibilidad general	Se agregó documentación para el conjunto de claves AWS KMS ECDH y el anillo de claves ECDH sin procesar.	17 de junio de 2024
SDK de cifrado de AWS para Java versión 3.x	Se integra SDK de cifrado de AWS para Java con la biblioteca de proveedores de materiales. Añade soporte para los anillos de claves y el contexto de cifrado requerido (CMM).	6 de diciembre de 2023
AWS Encryption SDK para .NET versión 4.x	Añade compatibilidad con el anillo de claves AWS KMS	12 de octubre de 2023

	jerárquico, el contexto de cifrado requerido (CMM) y los anillos de claves RSA asimétricos. AWS KMS	
Disponibilidad general	Presentamos la compatibilidad con .NET. AWS Encryption SDK	17 de mayo de 2022
Cambio de documentación	Sustituya el AWS Key Management Service término clave maestra del cliente (CMK) por clave KMS. AWS KMS key	30 de agosto de 2021
Disponibilidad general	Se agregó soporte para. AWS Key Management Service(AWS KMS) Claves multirregionales. Las claves multirregionales son AWS KMS claves diferentes Regiones de AWS que se pueden usar indistintamente porque tienen el mismo identificador de clave y material clave.	8 de junio de 2021
Disponibilidad general	Se agregó y actualizó documentación sobre el proceso mejorado de descifrado de mensajes.	11 de mayo de 2021

Disponibilidad general	Se agregó y actualizó la documentación para la versión 1.8 de AWS Encryption CLI para disponibilidad general. x para reemplazar la versión 1.7 de la CLI de AWS cifrado. x y CLI AWS de cifrado 2.1. x para sustituir a la CLI de AWS cifrado 2.0. x.	27 de octubre de 2020
Disponibilidad general	Se agregó y actualizó la documentación para la versión de disponibilidad general de las versiones 1.7.x y 2.0.x de AWS Encryption SDK , que incluye una guía de prácticas recomendadas, una guía de migración, conceptos actualiza dos, temas de lenguajes de programación actualizados, una referencia actualizada sobre conjuntos de algoritmo s, una referencia actualiza da sobre el formato de los mensajes y un nuevo ejemplo de formato de mensaje.	24 de septiembre de 2020
Disponibilidad general	Documentación agregada y actualizada para la versión del SDK de cifrado de AWS para JavaScript disponible en general.	1 de octubre de 2019

Versión de prueba	Se ha añadido y actualizado la documentación de la versión beta pública del SDK de cifrado de AWS para JavaScript.	21 de junio de 2019
Disponibilidad general	Documentación agregada y actualizada para la versión del SDK de cifrado de AWS para C disponible en general.	16 de mayo de 2019
Versión de prueba	Se ha añadido documentación de la versión preliminar del SDK de cifrado de AWS para C para C.	5 de febrero de 2019
Nueva versión	Se ha agregado documentación sobre la interfaz de línea de comandos para el AWS Encryption SDK.	20 de noviembre de 2017

Actualizaciones anteriores

En la siguiente tabla, se describen los cambios significativos introducidos en la Guía para desarrolladores de AWS Encryption SDK antes de noviembre de 2017.

Cambio	Descripción	Fecha
Nueva versión	Se ha agregado el capítulo Almacenamiento en caché de claves de datos de la nueva característica. Se agregó el the section called “Referencia de vectores de inicialización” tema que explica que el SDK pasó	31 de julio de 2017

Cambio	Descripción	Fecha
	de generar aleatorios IVs a construir deterministas IVs. Se agregó el tema the section called “Conceptos” para explicar conceptos, incluido el nuevo administrador de materiales criptográficos.	
Actualización	Se ha ampliado la documentación de Referencia de formato de mensajes en una nueva sección AWS Encryption SDK referencia. Se agregó una sección sobre. AWS Encryption SDK Conjuntos de algoritmos admitidos	21 de marzo de 2017
Nueva versión	AWS Encryption SDK Ahora es compatible con el lenguaje de Python programación, además deJava.	21 de marzo de 2017
Versión inicial	Versión inicial de AWS Encryption SDK y de esta documentación.	22 de marzo de 2016

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.