aws

Guía para desarrolladores

AWS SDK de cifrado de bases de datos

AWS SDK de cifrado de bases de datos: Guía para desarrolladores

Table of Contents

¿Qué es el SDK de cifrado de bases de datos deAWS?

Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

El SDK de cifrado de bases de datos de AWS es un conjunto de bibliotecas de software que le permiten incluir el cifrado del cliente en el diseño de la base de datos. El SDK de cifrado de bases de datos de AWS proporciona soluciones de cifrado a nivel de registro. Usted especifica qué campos se cifran y qué campos se incluyen en las firmas que garantizan la autenticidad de sus datos. El cifrado de sus datos en tránsito y en reposo confidenciales ayuda a garantizar que los datos de texto no cifrado no estén disponibles para ningún tercero, incluido AWS. El SDK de cifrado de bases de datos de AWS se suministra gratuitamente con la licencia Apache 2.0.

Esta guía para desarrolladores ofrece información general conceptual del SDK de cifrado de bases de datos de AWS, que incluye una introducción a su arquitectura, detalles acerca de cómo protege los datos, cómo difiere del cifrado del servidor y asesoramiento sobre la selección de componentes críticos para su aplicación para ayudarle a comenzar.

El SDK de cifrado de bases de datos de AWS es compatible con Amazon DynamoDB con el cifrado a nivel de atributos. Versión 3.x de la biblioteca de cifrado del cliente de Java para DynamoDB es una reescritura importante del cliente de cifrado de DynamoDB para Java. Incluye numerosas actualizaciones, como un nuevo formato de datos estructurados, una compatibilidad mejorada con multitenencia, un cifrado para búsquedas y una compatibilidad para cambios de esquema perfectos.

El SDK de cifrado de bases de datos de AWS incluye los siguientes beneficios:

Diseñado especialmente para aplicaciones de bases de datos

No es necesario ser un experto en criptografía para utilizar el SDK de cifrado de bases de datos de AWS. Las implementaciones incluyen métodos de ayudante que se diseñaron para funcionar con sus aplicaciones existentes.

Después de crear y configurar los componentes requeridos, el cliente de cifrado descifra y firma los registros de forma transparente cuando los agrega a una base de datos y los verifica y los descifra cuando los recupera.

Incluye cifrado y firma seguros

El SDK de cifrado de bases de datos de AWS incluye implementaciones seguras que cifran los valores de campo en cada registro utilizando una clave de cifrado de datos única y, a continuación, firman el elemento para registro de cambios no autorizados como, por ejemplo, agregar o eliminar atributos o cambiar valores cifrados.

Utiliza materiales criptográficos desde cualquier origen

El SDK de cifrado de bases de datos de AWS utiliza anillos de claves para generar, cifrar y descifrar la clave de cifrado de datos única que protege su registro. Los llaveros determinan las claves de empaquetado que cifran esa clave de datos.

Puede utilizar claves de empaquetado de cualquier fuente, incluidos los servicios de criptografía, como AWS Key Management Service (AWS KMS) o AWS CloudHSM. El SDK de cifrado de bases de datos de AWS no requiere una Cuenta de AWS ni ningún otro servicio de AWS.

Compatibilidad para el almacenamiento en caché de materiales criptográficos

El llavero AWS KMS 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. Le permite proteger sus materiales criptográficos con una clave KMS de cifrado simétrico sin tener que llamar AWS KMS cada vez que cifra o descifra un registro. El AWS KMS llavero jerárquico es una buena opción para las aplicaciones que necesitan minimizar las llamadas a AWS KMS.

Cifrado para búsquedas

Puede diseñar bases de datos que puedan buscar registros cifrados sin necesidad de descifrar toda la base de datos. Según el modelo de amenazas y los requisitos de consulta, puede utilizar el cifrado con capacidad de búsqueda para realizar búsquedas de coincidencias exactas o consultas complejas más personalizadas en la base de datos cifrada.

Compatibilidad para esquemas de bases de datos de multitenencia

El SDK de cifrado de bases de datos de AWS le permite proteger los datos almacenados en bases de datos con un esquema compartido al aislar cada inquilino con materiales de cifrado distintos. Si tiene varios usuarios que realizan operaciones de cifrado en su base de datos, utilice uno de los AWS KMS llaveros para proporcionar a cada usuario una clave distinta para utilizarla

en sus operaciones criptográficas. Para obtener más información, consulte Trabajar con bases de datos de varios inquilinos.

Compatibilidad para actualizaciones de esquemas fluidas

Al configurar el SDK de cifrado de bases de datos de AWS, se proporcionan acciones criptográficas que indican al cliente qué campos debe cifrar y firmar, qué campos debe firmar (pero no cifrar) y cuáles debe ignorar. Una vez que haya utilizado el SDK de cifrado de bases de datos de AWS para proteger sus registros, podrá seguir realizando cambios en el modelo de datos. Puede actualizar sus acciones criptográficas, como agregar o eliminar campos cifrados, en una sola implementación.

Desarrollado en repositorios de código abierto

El SDK de cifrado de bases de datos de AWS se desarrolla en repositorios de código abierto en GitHub. Puede usar estos repositorios para ver el código, leer y enviar los problemas y encontrar información específica de la implementación.

El SDK de cifrado de bases de datos de AWS para DynamoDB

- Biblioteca de cifrado del cliente de Java para DynamoDB: aws-database-encryption-sdk-dynamodb-java

  Versión 3.x de la biblioteca de cifrado del cliente de Java para DynamoDB es un producto del SDK de cifrado de bases de datos de AWS en Dafny, un lenguaje compatible con la verificación en el que se escriben las especificaciones, el código para implementarlas y las pruebas para comprobarlas. El resultado es una biblioteca que implementa las características del SDK de cifrado de bases de datos de AWS para DynamoDB en un marco que garantiza la corrección funcional.

Compatibilidad y mantenimiento

El SDK de cifrado de bases de datos de AWS utiliza la misma política de mantenimiento que utilizan el SDK y las herramientas de AWS, incluidas las fases de control de versiones y de ciclo de vida. Como práctica recomendada, le recomendamos que utilice la última versión disponible del SDK de cifrado de bases de datos de AWS para la implementación de su base de datos y que la actualice a medida que se publiquen nuevas versiones.

Para obtener más información, consulte Política de mantenimiento de SDK y herramientas de AWS en la Guía de referencia de SDK y herramientas de AWS.

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 SDK de cifrado de bases de datos de AWS, informe a seguridad de AWS. No cree un problema público en GitHub.

Para enviar comentarios sobre esta documentación, utilice el enlace de comentarios de cualquier página.

AWS Conceptos del SDK de encriptación de bases

> Se cambió el nombre de nuestra biblioteca de cifrado del lado del cliente por el de SDK de cifrado de AWS bases de datos. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

En este tema se explican los conceptos y la terminología que se utilizan en el SDK de cifrado de AWS bases de datos.

Para obtener información sobre cómo interactúan los componentes del SDK de cifrado de AWS bases de datos, consulteCómo funciona el SDK de cifrado de bases de datos de AWS.

Para obtener más información sobre el SDK AWS de cifrado de bases de datos, consulte los siguientes temas.

- Descubra cómo el SDK AWS de cifrado de bases de datos 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 registros y las claves de encapsulación que protegen sus claves de datos.
- Obtenga información sobre los conjuntos de claves que determinan qué claves de encapsulación debe utilizar.
- Obtenga información sobre el contexto de cifrado que agrega integridad a su proceso de cifrado.
- Obtenga información sobre la descripción del material que los métodos de cifrado agregan a su registro.

- Obtenga información sobre las acciones criptográficas que indican al SDK de cifrado de bases de datos de AWS qué campos debe cifrar y firmar.

Temas

- Cifrado de sobre
- Clave de datos
- Clave de encapsulación
- Conjuntos de claves
- Funciones criptográficas
- Descripción de material
- Contexto de cifrado
- Administrador de materiales criptográficos
- Cifrado simétrico y asimétrico
- Compromiso clave
- Firmas digitales

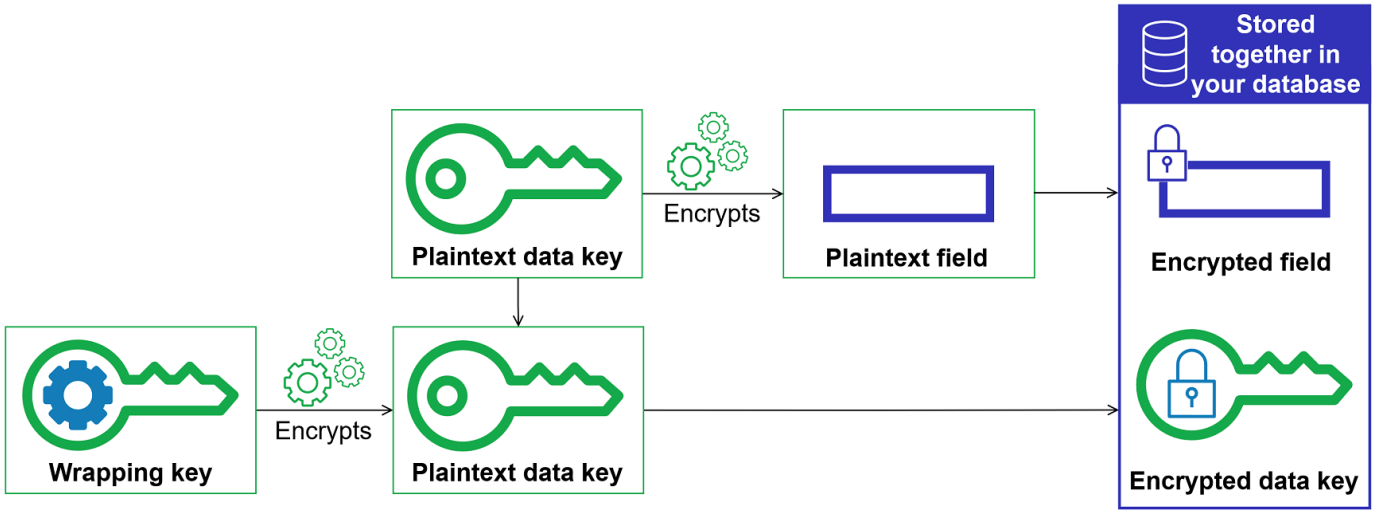
Cifrado de sobre

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

El SDK AWS de cifrado de bases de datos cifra cada campo con una clave de datos única. A continuación, cifra cada clave de datos con la clave de encapsulación que especifique. Almacena las claves de datos cifradas en la descripción del material.

Para especificar la clave de encapsulación, utilice un conjunto de claves.



Cifrado de los mismos datos con varias claves múltiples

Puede cifrar la clave de datos con varias claves de encapsulación. Es posible que desee proporcionar diferentes claves de encapsulación para distintos usuarios, o claves de encapsulación de diferentes tipos o en diferentes ubicaciones. Cada una de las claves de encapsulamiento cifra la misma clave de datos. El SDK AWS de cifrado de bases de datos almacena todas las claves de datos cifrados junto con los campos cifrados de la descripción del material.

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

Combinación de los puntos fuertes de varios algoritmos

Para cifrar los datos, de forma predeterminada, el SDK de cifrado de AWS bases de datos utiliza un conjunto de algoritmos con cifrado simétrico AES-GCM, una función de derivación de claves basada en HMAC (HKDF) y firma ECDSA. 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 de las funciones. Para combinar las fortalezas de cada uno, puede cifrar la clave de datos con el cifrado de clave pública.

Recomendamos utilizar AWS KMS uno de los anillos de claves siempre que sea posible. Al usar el AWS KMS llavero, puede optar por combinar los puntos fuertes de varios algoritmos especificando un RSA asimétrico AWS KMS key como clave de empaquetado. También puede utilizar una clave de KMS de cifrado simétrico.

Clave de datos

Una clave de datos es una clave de cifrado que el SDK de cifrado de AWS bases de datos utiliza para cifrar los campos de un registro que están marcados `ENCRYPT_AND_SIGN` en las acciones criptográficas. Cada clave de datos es una matriz de bytes que es conforme a los requisitos para claves criptográficas. El SDK AWS de cifrado de bases de datos utiliza una clave de datos única para cifrar cada atributo.

No es necesario especificar, generar, implementar, extender, proteger ni usar claves de datos. El SDK de cifrado de bases de datos de AWS hace ese trabajo por usted cuando llama a las operaciones de cifrado y descifrado.

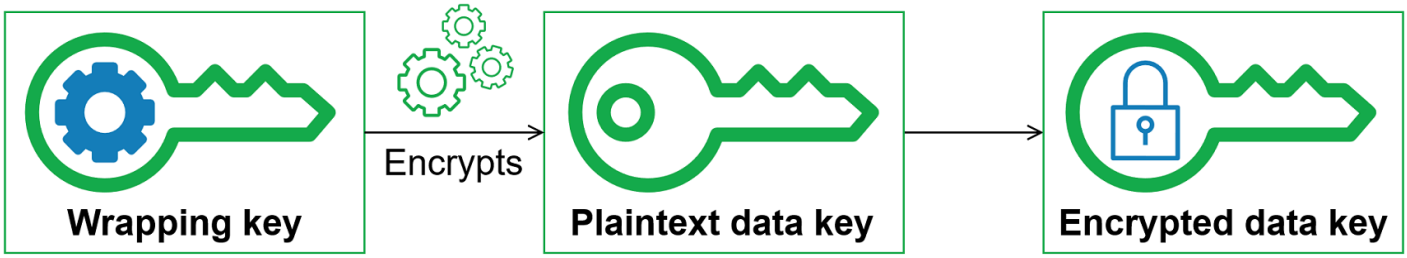
Para proteger sus claves de datos, el SDK de cifrado AWS de bases de datos las cifra en una o más claves de cifrado clave conocidas como claves de empaquetado. Una vez que el SDK de cifrado de AWS bases de datos utiliza las claves de datos de texto sin formato para cifrar los datos, las elimina de la memoria lo antes posible. Almacena las claves de datos cifradas en la descripción del material. Para obtener más detalles, consulte Cómo funciona el SDK de cifrado de bases de datos de AWS.

> **Tip**
>
> En el SDK de cifrado AWS de bases de datos, distinguimos las claves de datos de las claves de cifrado de datos. Como práctica recomendada, todos los conjuntos de algoritmos compatibles utilizan una función de derivación de clave. La función de derivación de clave toma una clave de datos como entrada y devuelve las claves de cifrado de datos que son las que se utilizan realmente para cifrar los registros. 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 permiten que el SDK de cifrado de AWS bases de datos identifique las claves de empaquetado válidas al descifrar.

Clave de encapsulación

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



El SDK de cifrado de AWS bases de datos admite varias claves de empaquetado que se utilizan habitualmente, como AWS Key Management Service(AWS KMS) las claves KMS de cifrado simétrico (incluidas las claves multirregionales) y las AWS KMS claves KMS RSA asimétricas, las claves AES-GCM (Advanced Encryption Standard/Galois Counter Mode) sin procesar y las claves RSA sin procesar. Recomendamos utilizar claves KMS siempre que sea posible. Para decidir qué clave de encapsulación debe usar, consulte Seleccionar claves de encapsulación.

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

- Utilice un servicio 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 KMSEl SDK AWS de cifrado de bases de datos se integra AWS KMS para ayudarle a proteger y utilizar sus claves de empaquetado.

Conjuntos de claves

Para especificar las claves de encapsulación que utiliza para el cifrado y el descifrado, utilice un conjunto de claves Puede usar los conjuntos de claves que proporciona el SDK AWS de cifrado de bases de datos o diseñar sus propias implementaciones.

Un conjunto de claves genera, cifra y descifra claves de datos. También genera las claves MAC que se utilizan para calcular los códigos de autenticación de mensajes basados en hash (HMAC) en la firma. Al definir un conjunto de claves, puede especificar las claves de encapsulación que cifran las 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. Al cifrar, el SDK de cifrado AWS de bases de datos utiliza todas las claves de empaquetado especificadas en

el anillo de claves para cifrar la clave de datos. Si necesita ayuda para elegir y usar los conjuntos de claves que define el SDK de cifrado de AWS bases de datos, consulte Uso de conjuntos de claves.

Funciones criptográficas

Las acciones criptográficas indican al encriptador qué acciones debe realizar en cada campo de un registro.

Los valores de las acciones criptográficas pueden ser uno de los siguientes:

- Encrypt and sign: cifre el campo. Incluya el campo cifrado en la firma.
- Sign only: incluya el campo en la firma.
- Do nothing: no cifre ni incluya el campo en la firma.

Para cualquier campo que pueda almacenar datos confidenciales, utilice Encrypt and sign. Para los valores de la clave principal (por ejemplo, una clave de partición y una clave de clasificación en una tabla de DynamoDB), utilice Sign only. No es necesario especificar acciones criptográficas para la descripción del material. El SDK AWS de cifrado de bases de datos firma automáticamente el campo en el que está almacenada la descripción del material.

Elija sus acciones criptográficas con cuidado. En caso de duda, use Encrypt and sign. Una vez que haya utilizado el SDK de cifrado de AWS bases de datos para proteger sus registros, no podrá cambiar un `SIGN_ONLY` campo `ENCRYPT_AND_SIGN` o un campo existente ni cambiar la acción criptográfica asignada a un `DO_NOTHING` campo existente. `DO_NOTHING` Sin embargo, aún puede realizar otros cambios en su modelo de datos. Por ejemplo, puede agregar o eliminar campos cifrados en una sola implementación.

Descripción de material

La descripción del material sirve como encabezado de un registro cifrado. Al cifrar y firmar campos con el SDK de cifrado de AWS bases de datos, el cifrador registra la descripción del material a medida que reúne los materiales criptográficos y almacena la descripción del material en un campo nuevo (`aws_dbe_head`) que el cifrador añade al registro.

La descripción del material es una estructura de datos con formato portátil que contiene copias cifradas de las claves de datos y otra información, como los algoritmos de cifrado, el contexto de cifrado y las instrucciones de cifrado y firma. El encriptador registra la descripción del material mientras reúne los materiales criptográficos para el cifrado y la firma. Posteriormente, cuando

necesita reunir los materiales criptográficos para verificar y descifrar un campo, utiliza la descripción del material como guía.

Almacenar las claves de datos cifrados y los campos cifrados 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 la descripción del material, consulteFormato de descripción del material.

Contexto de cifrado

Para mejorar la seguridad de sus operaciones criptográficas, el SDK de cifrado de AWS bases de datos incluye un contexto de cifrado en todas las solicitudes de cifrado y firma de un registro.

Un contexto de cifrado es un conjunto de pares de nombre-valor que contienen datos autenticados adicionales no secretos y arbitrarios. El SDK AWS de cifrado de bases de datos incluye el nombre lógico de la base de datos y los valores de la clave principal (por ejemplo, una clave de partición y una clave de clasificación en una tabla de DynamoDB) en el contexto de cifrado. Cuando se cifra y firma un campo, 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.

Si utiliza un conjunto de AWS KMS claves, el SDK de cifrado de AWS bases de datos también utiliza el contexto de cifrado para proporcionar datos autenticados (AAD) adicionales en las llamadas que realiza el conjunto de claves. AWS KMS

Cuando se utiliza el conjunto de algoritmo predeterminado, el administrador de materiales criptográficos (CMM) agrega 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. La clave de verificación pública se guarda en la descripción del material.

Administrador de materiales criptográficos

El administrador de materiales criptográficos (CMM) reúne los materiales criptográficos que se utilizan para cifrar, descifrar y firmar los datos. Siempre que utilice el conjunto de algoritmos predeterminado, los materiales criptográficos incluyen claves de datos cifrados y de texto no cifrado, claves de firma simétricas y una clave de firma asimétrica. Nunca se interactúa directamente con el CMM. Los métodos de cifrado y descifrado se encargan de ello.

Como el CMM actúa como enlace entre el SDK de cifrado de AWS bases de datos y un conjunto de claves, es un punto ideal para la personalización y la ampliación, por ejemplo, como apoyo a la aplicación de políticas. Puede especificar un CMM de forma explícita, pero no es obligatorio. Al especificar un conjunto de claves, el SDK de cifrado de bases de datos de AWS crea automáticamente un CMM predeterminado para usted. El CMM predeterminado obtiene los materiales de cifrado o descifrado del conjunto de claves que especifique. Esto podría requerir una llamada a un servicio criptográfico como AWS Key Management Service (AWS KMS).

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. Para obtener más información, consulte Algoritmos criptográficos en la AWS Guía de herramientas y servicios criptográficos.

El SDK de cifrado AWS de bases de datos utiliza el cifrado por sobres. Cifra los datos con una clave de datos simétrica. Cifra la clave de datos simétrica con una o más claves de encapsulación simétricas o asimétricas. Agrega una descripción del material al registro que incluye al menos una copia cifrada de la clave de datos.

Cifrar los datos (cifrado simétrico)

> Para cifrar los datos, el SDK de cifrado de AWS bases de datos utiliza una clave de datos simétrica y un conjunto de algoritmos que incluye un algoritmo de cifrado simétrico. Para descifrar los datos, el SDK de cifrado de AWS bases de datos 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 que se 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 que utilice cifrado simétrico, como un anillo de AWS KMS claves con una clave KMS de cifrado simétrico, o uno que utilice cifrado asimétrico, como un AWS KMS anillo de claves con una clave KMS RSA asimétrica.

Compromiso clave

El SDK AWS de cifrado de bases de datos admite el compromiso de claves (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 el registro para descifrarlo. El SDK de cifrado de bases de datos de AWS incluye un compromiso clave para todas las operaciones de cifrado y descifrado.

La mayoría de los sistemas de cifrado simétricos modernos (incluido el AES) cifran el texto sin formato con una única clave secreta, como la clave de datos única que el SDK de cifrado de AWS bases de datos utiliza para cifrar cada campo de texto sin formato marcado en un registro. `ENCRYPT_AND_SIGN` Al descifrar este registro con la misma clave de datos, se obtiene un texto no cifrado idéntico al original. El descifrado con una clave diferente suele fallar. Aunque es difícil, técnicamente es posible descifrar un texto cifrado con dos claves diferentes. En raras ocasiones, es posible encontrar una clave que pueda descifrar parcialmente el texto cifrado y convertirlo en un texto no cifrado diferente, pero aún inteligible.

El SDK de cifrado AWS de bases de datos siempre cifra cada atributo con una clave de datos única. Puede cifrar esa clave de datos con varias claves de encapsulación, pero las claves de encapsulación siempre cifran la misma clave de datos. Sin embargo, un registro cifrado sofisticado y creado manualmente puede contener diferentes claves de datos, cada una cifrada con una clave de encapsulación diferente. Por ejemplo, si un usuario descifra el registro cifrado, devuelve 0x0 (falso), mientras que otro usuario que descifra el mismo registro cifrado obtiene 0x1 (verdadero).

Para evitar esta situación, el SDK de cifrado AWS de bases de datos incluye la asignación de claves al cifrar y descifrar. El método de cifrado vincula criptográficamente la clave de datos única que produjo el texto cifrado con el compromiso clave, un código de autenticación de mensajes basado en hash (HMAC) que se calcula sobre la descripción del material mediante una derivación de la clave de datos. A continuación, almacena el compromiso de clave en la descripción del material. Cuando descifra un registro con un compromiso de clave, el SDK de cifrado de AWS bases de datos comprueba que la clave de datos es la única clave de ese registro cifrado. Si se produce un error en la verificación de la clave de datos, se produce un error en la operación de descifrado.

Firmas digitales

Para garantizar la autenticidad de los datos al pasar de un sistema a otro, puede aplicar una firma digital al registro. Las firmas digitales son siempre asimétricas. Utiliza su clave privada para crear la firma y anexarla al registro original. El destinatario usa una clave pública para comprobar que el registro no se ha modificado desde que lo firmó. Debe utilizar firmas digitales si los usuarios que cifran los datos y los que los descifran no tienen el mismo nivel de confianza.

El SDK de cifrado de AWS bases de datos cifra los datos mediante un algoritmo de cifrado autenticado, el AES-GCM, pero 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 provocar un posible problema de seguridad.

Para evitar este problema, el conjunto de algoritmos predeterminado agrega una firma ECDSA (algoritmo de firma digital con curva elíptica) a los registros cifrados. El conjunto de algoritmos predeterminado cifra los campos del registro marcados `ENCRYPT_AND_SIGN` con un algoritmo de cifrado autenticado, el AES-GCM. A continuación, calcula los códigos de autenticación de mensajes basados en hash (HMAC) y las firmas ECDSA asimétricas en los campos del registro marcados con `ENCRYPT_AND_SIGN` y `SIGN_ONLY`. El proceso de descifrado utiliza las firmas para comprobar que un usuario autorizado haya cifrado el registro.

Cuando se utiliza el conjunto de algoritmos predeterminado, el SDK AWS de cifrado de bases de datos genera una clave privada temporal y una clave pública para cada registro cifrado. El SDK AWS de cifrado de bases de datos almacena la clave pública en la descripción del material y descarta la clave privada, por lo que nadie puede crear otra firma que verifique con la clave pública. Como el algoritmo vincula la clave pública a la clave de datos cifrados como datos autenticados adicionales en la descripción del material, un usuario que solo pueda descifrar registros no podrá alterar la clave pública.

El SDK de cifrado AWS de bases de datos siempre incluye la verificación de HMAC. Las firmas digitales ECDSA están habilitadas de forma predeterminada, pero no son obligatorias. Si los usuarios que cifran los datos y los que los descifran tienen el mismo nivel de confianza, podría considerar la posibilidad de utilizar un conjunto de algoritmos que no incluya firmas digitales para mejorar su rendimiento. Para obtener más información sobre cómo seleccionar conjuntos de algoritmos alternativos, consulte Elegir un conjunto de algoritmos.

Cómo funciona el SDK de cifrado de bases de datos de AWS

> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

El SDK de cifrado de bases de datos de AWS proporciona bibliotecas de cifrado del cliente diseñadas específicamente para proteger los datos que se almacenan en las bases de datos. Las bibliotecas incluyen implementaciones seguras que puede ampliar o utilizar sin hacer ningún cambio. Para obtener más información sobre la definición y el uso de componentes personalizados, consulte el repositorio de GitHub para la implementación de la base de datos.

Los flujos de trabajo de esta sección explican cómo el SDK de cifrado de bases de datos de AWS cifra, firma, descifra y verifica los datos de la base de datos. Estos flujos de trabajo describen el proceso básico mediante elementos abstractos y las características predeterminadas. Para obtener más información sobre cómo funciona el SDK de cifrado de bases de datos de AWS con la implementación de la base de datos, consulte el tema Qué está cifrado en la base de datos.

El SDK de cifrado de bases de datos de AWS utiliza el cifrado de sobre para proteger sus datos. Cada registro se cifra con una clave de datos única. La clave de datos se utiliza para obtener una clave de cifrado de datos única para cada campo marcado `ENCRYPT_AND_SIGN` en sus acciones criptográficas. A continuación, las claves de empaquetado que especifique cifran una copia de la clave de datos. Para descifrar el registro cifrado, el SDK de cifrado de bases de datos de AWS utiliza las claves de empaquetado que especifique para descifrar al menos una clave de datos cifrada. A continuación, puede descifrar el texto cifrado y devolver una entrada de texto no cifrado.

Para obtener más información sobre los términos utilizados en el SDK de cifrado de bases de datos de AWS, consulte AWS Conceptos del SDK de encriptación de bases.

Cifrar y firmar

En esencia, el SDK de cifrado de bases de datos de AWS es un encriptador de registros que cifra, firma, verifica y descifra los registros de la base de datos. Recibe información acerca de los registros e instrucciones sobre qué elementos hay que cifrar y firmar. Obtiene los materiales de cifrado, y las instrucciones sobre su uso, desde un proveedor de materiales criptográficos configurado a partir de la clave de empaquetado que especifique.

En el siguiente tutorial se describe cómo el SDK de cifrado de bases de datos de AWS cifra y firma las entradas de datos.

1. El administrador de materiales criptográficos proporciona al SDK de cifrado de bases de datos de AWS claves de cifrado de datos únicas: una clave de datos en texto no cifrado, una copia de la clave de datos cifrada con la clave de empaquetado especificada y una clave MAC.

   > **Note**
   >
   > Puede cifrar la clave de datos con varias claves de empaquetado. Cada una de las claves de empaquetado cifra una copia independiente de la clave de datos. El SDK de cifrado de bases de datos de AWS almacena todas las claves de datos cifrados en la descripción del material. El SDK de cifrado de bases de datos de AWS agrega un nuevo campo (`aws_dbe_head`) al registro que almacena la descripción del material.

Se obtiene una clave MAC para cada copia cifrada de la clave de datos. Las claves MAC no se almacenan en la descripción del material. En su lugar, el método de descifrado utiliza las claves de empaquetado para volver a derivar las claves MAC.

2. El método de cifrado cifra cada campo marcado como `ENCRYPT_AND_SIGN` en las acciones criptográficas especificadas.
3. El método de cifrado deriva un `commitKey` de la clave de datos y lo utiliza para generar un valor de compromiso de clave y, a continuación, descarta la clave de datos.
4. El método de cifrado agrega una descripción del material al registro. La descripción del material contiene las claves de datos cifrados y la información adicional sobre el registro cifrado. Para obtener una lista completa de la información incluida en la descripción del material, consulte Formato de la descripción del material.
5. El método de cifrado utiliza las claves MAC devueltas en el Paso 1 para calcular los valores del código de autenticación de mensajes basado en hash (HMAC) mediante la canonicalización de la descripción del material, el contexto de cifrado y cada campo marcado `ENCRYPT_AND_SIGN` o `SIGN_ONLY` incluido en las acciones criptográficas. Los valores del HMAC se almacenan en un campo nuevo (`aws_dbe_foot`) que el método de cifrado agrega al registro.
6. El método de cifrado calcula una firma ECDSA sobre la canonicalización de la descripción del material, el contexto de cifrado y cada campo marcado con `ENCRYPT_AND_SIGN` o `SIGN_ONLY` y almacena las firmas ECDSA en el campo`aws_dbe_foot`.

> **Note**
>
> Las firmas ECDSA están habilitadas de forma predeterminada, pero no son obligatorias.

7. El método de cifrado almacena el registro cifrado y firmado en la base de datos

Descifrar y verificar

1. El administrador de materiales criptográficos (CMM) proporciona el método de descifrado con los materiales de descifrado almacenados en la descripción del material, incluida la clave de datos de texto no cifrado y la clave MAC asociada.
   - El CMM descifra la clave de datos cifrada mediante las claves de empaquetado del llavero especificado y devuelve la clave de datos en texto no cifrado.

2. El método de descifrado compara y verifica el valor de compromiso clave en la descripción del material.
3. El método de descifrado verifica las firmas en el campo de la firma.

   Identifica qué campos están marcados `ENCRYPT_AND_SIGN` y `SIGN_ONLY` de la lista de campos no autenticados permitidos que haya definido. El método de descifrado utiliza la clave MAC devuelta en el Paso 1 para volver a calcular y comparar los valores HMAC de los campos marcados con `ENCRYPT_AND_SIGN` o `SIGN_ONLY`. A continuación, verifica las firmas del ECDSA mediante la clave pública almacenada en el contexto de cifrado.
4. El método de descifrado usa la clave de datos en texto no cifrado para descifrar cada valor marcado `ENCRYPT_AND_SIGN`. A continuación, el SDK de cifrado de bases de datos de AWS descarta la clave de datos de texto no cifrado.
5. El método de descifrado devuelve el registro en texto no cifrado.

Conjuntos de algoritmos compatibles en el SDK de cifrado de bases de datos de AWS

Se cambió el nombre de nuestra biblioteca de cifrado del cliente por el de SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

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 SDK de cifrado de bases de datos de AWS utiliza un conjunto de algoritmos para cifrar y firmar los campos de la base de datos. El SDK de cifrado de bases de datos de AWS admite dos conjuntos de algoritmos. Todos los conjuntos admitidos utilizan Advanced Encryption Standard (AES) como algoritmo principal y lo combinan con otros algoritmos y valores.

Conjunto de algoritmos predeterminado

Para cifrar los datos sin procesar, el conjunto de algoritmos del SDK de cifrado de bases de datos de AWS usa el algoritmo Advanced Encryption Standard (AES) en Galois/Counter Mode (GCM),

denominado AES-GCM. El SDK de cifrado de bases de datos de AWS admite claves de cifrado de 256 bits. La longitud de la etiqueta de autenticación es siempre de 16 bytes.

De forma predeterminada, el SDK de cifrado de bases de datos de AWS utiliza un conjunto de algoritmos con AES-GCM con una función de extracción y expansión de claves (HKDF) basada en HMAC, asignación de claves, firma simétrica y asimétrica y una clave de cifrado de 256 bits.

El SDK de cifrado de bases de datos de AWS utiliza un conjunto de algoritmos que deriva una clave de datos AES-GCM al proporcionar una clave de cifrado de datos de 256 bits a la función de extracción y expansión de claves (HKDF) basada en HMAC. También deriva una clave MAC para la clave de datos. El SDK de cifrado de bases de datos de AWS utiliza esta clave de datos para obtener una clave de cifrado de datos única para cifrar cada campo. A continuación, el SDK de cifrado de bases de datos de AWS utiliza la clave MAC para calcular un código de autenticación de mensajes basado en hash (HMAC) para cada copia cifrada de la clave de datos y añade una firma con un algoritmo de firma digital de curva elíptica (ECDSA) al registro. Este conjunto de algoritmos también deriva de un compromiso clave: un HMAC que vincula la clave de datos al registro. El valor de compromiso clave es un HMAC que se calcula a partir de la descripción del material y la clave de compromiso, que se obtiene mediante el HKDF mediante un procedimiento similar al de la obtención de la clave de cifrado de datos. A continuación, el valor de compromiso clave se almacena en la descripción del material.

Algoritmo de cifrado	Longitud de la clave de cifrado de datos (en bits)	Algoritmo de firma simétrica	Algoritmo de firma asimétrica.	Compromiso clave
AES-GCM	256	HMAC-SHA-384	ECDSA sobre P384	HKDF con SHA-512

Este conjunto de algoritmos serializa la descripción del material??? y todos los campos marcados `ENCRYPT_AND_SIGN` y `SIGN_ONLY` en las acciones criptográficas y, a continuación, utiliza el HMAC con un algoritmo de función hash criptográfica (SHA-512) para firmar la canonicalización. A continuación, calcula una firma digital ECDSA. Las firmas HMAC y ECDSA se almacenan en un campo nuevo (`aws_dbe_foot`) que el SDK de cifrado de bases de datos de AWS agrega al registro. Las firmas digitales son especialmente útiles cuando la política de autorización permite a un grupo de usuarios cifrar datos y a otro grupo diferente descifrarlos.

El compromiso clave garantiza que cada texto cifrado se descifre en un solo texto no cifrado. Para ello, validan la clave de datos utilizada como entrada en el algoritmo de cifrado. Al cifrar, estos conjuntos de algoritmos obtienen un compromiso clave HMAC. Antes de descifrar, validan que la clave de datos produzca el mismo compromiso de clave HMAC. En caso contrario, el comando de descifrado genera un error.

AES-GCM sin firmas digitales

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. Este conjunto se utiliza cuando los usuarios que cifran los datos y aquellos que los descifran son merecedores de la misma confianza.

Todos los conjuntos de algoritmos del SDK de cifrado de bases de datos de AWS admiten la firma simétrica HMAC-SHA-384. La única diferencia es que el conjunto de algoritmos AES-GCM sin firmas digitales carece de la firma ECDSA, lo que proporciona un nivel adicional de autenticidad y no repudio.

Por ejemplo, si tiene varias claves de empaquetado en el llavero, `wrappingKeyA`, `wrappingKeyB` y `wrappingKeyC`, y usted descifra un registro utilizando `wrappingKeyA`, la firma simétrica HMAC-SHA-384 comprueba que el registro lo haya cifrado un usuario con acceso a `wrappingKeyA`.
Si ha utilizado los algoritmos predeterminados, los HMAC proporcionan la misma verificación y, además`wrappingKeyA`, utilizan la firma ECDSA para garantizar que el registro lo haya cifrado un usuario con permisos de cifrado para ello. `wrappingKeyA`

Para seleccionar el conjunto de algoritmos AES-GCM sin firmas digitales, especifíquelo en la configuración de cifrado.

Uso del SDK de cifrado de bases de datos de AWS con AWS KMS

Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

Para usar el SDK de cifrado de bases de datos de AWS, debe configurar un conjunto de claves y especificar una o más claves de empaquetado. Si no tiene una infraestructura de claves, le recomendamos que use AWS Key Management Service (AWS KMS).

El SDK de cifrado de bases de datos de AWS admite dos tipos de conjuntos de claves AWS KMS. El llavero AWS KMS tradicional utiliza AWS KMS keys para generar, cifrar y descifrar claves de datos. Puede utilizar claves de cifrado simétrico (`SYMMETRIC_DEFAULT`) o asimétricas de RSA KMS. Como el SDK de cifrado de bases de datos de AWS cifra y firma todos los registros con una clave de datos única, el llavero AWS KMS debe requerir cada operación AWS KMS de cifrado y descifrado. Para las aplicaciones que necesitan minimizar el número de llamadasAWS KMS, el SDK de cifrado de bases de datos de AWS también es compatible con el conjunto de claves jerárquico AWS KMS. El conjunto 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 AWS KMSclaves de rama 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. Recomendamos utilizar los AWS KMS llaveros siempre que sea posible.

Para interactuar con AWS KMS, el SDK de cifrado de bases de datos de AWS requiere el módulo AWS KMS del AWS SDK for Java.

Para prepararse para usar el SDK de cifrado de bases de datos de AWS con AWS KMS

1. Cree un Cuenta de AWS Para obtener información sobre cómo hacerlo, consulte el tema Cómo crear y activar una cuenta nueva de Servicios web de Amazon en el Knowledge Center del AWS.
2. Crear un AWS KMS key de clave de cifrado simétrica. Para obtener más información, consulte Creación de claves en la Guía para desarrolladores de AWS Key Management Service.

> **Tip**
>
> Para utilizar la AWS KMS key mediante programación, necesitará el ID de clave o el nombre de recurso de Amazon (ARN) de la AWS KMS key. Para obtener ayuda para encontrar 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 clave de acceso y la clave de acceso secreta para un usuario de IAM o puede utilizar el 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 aconsejamos 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 usuario 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 del identificador de clave de acceso AWS SDK for Javay 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 permite que los SDK de AWS firmen automáticamente solicitudes a AWS. En los ejemplos de código del SDK de cifrado de bases de datos de AWS que interactúan con AWS KMS se da por hecho que ya se ha completado este paso.

Configuración del SDK de cifrado AWS de bases de datos

Se cambió el nombre de nuestra biblioteca de cifrado del lado del cliente por el de SDK de cifrado de AWS bases de datos. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

El SDK AWS de cifrado de bases de datos está diseñado para ser fácil de usar. Si bien el SDK AWS de cifrado de bases de datos 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.

Temas

Selección de las claves de encapsulación

El SDK AWS de cifrado de bases de datos genera una clave de datos simétrica única para cifrar cada campo. No necesita configurar, administrar ni usar las claves de datos. El SDK AWS de cifrado de bases de datos lo hace por usted.

Sin embargo, debe seleccionar una o más claves de encapsulación para cifrar cada clave de datos. El SDK de cifrado de bases de datos de AWS admite claves KMS de cifrado simétrico y claves RSA KMS asimétricas de AWS Key Management Service(AWS KMS). También es compatible con las claves simétricas AES y las claves asimétricas RSA que proporciona en diferentes tamaños. Usted es responsable de la seguridad y durabilidad de las 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, por ejemplo AWS KMS.

Para especificar las claves de encapsulación para el cifrado y el descifrado, utilice un conjunto de claves. Según el tipo de conjunto de claves que utilice, puede especificar una clave de encapsulación

o varias claves de empaquetado del mismo tipo o de diferentes tipos. Si utiliza varias claves de encapsulación para encapsular una clave de datos, cada clave de encapsulación cifrará una copia de la misma clave de datos. Las claves de datos cifradas (una por clave de encapsulación) se guardan en la descripción del material que se almacena junto al campo cifrado. Para descifrar los datos, el SDK de cifrado de AWS bases de datos debe utilizar primero una de sus claves de empaquetado para descifrar una clave de datos cifrada.

Recomendamos utilizar uno de los AWS KMS llaveros siempre que sea posible. El SDK AWS de cifrado de bases de datos proporciona el AWS KMS anillo de claves y el anillo de clavesAWS KMS jerárquico, lo que reduce la cantidad de llamadas realizadas. AWS KMSPara especificar una AWS KMS key en un conjunto de claves, utilice un identificador de clave compatible. AWS KMS Si utiliza el conjunto de claves AWS KMS jerárquico, debe especificar el ARN de clave. Para obtener más información sobre los identificadores clave de una AWS KMS clave, consulte los identificadores clave en la Guía para desarrolladores.AWS Key Management Service

- Al cifrar con un AWS KMS anillo de claves, puede especificar cualquier identificador de clave válido (ARN de clave, nombre de alias, ARN de alias o ID de clave) para una clave KMS de cifrado simétrico. Si usa una clave de RSA KMS asimétrica, debe especificar la clave ARN.

  Si especifica un nombre de alias o un ARN de alias para una clave de KMS al cifrar, el SDK de cifrado de bases de datos de AWS guarda el ARN de clave actualmente asociado a ese alias; no lo guarda. Los cambios en el alias no afectan a la clave KMS utilizada para descifrar las claves de datos.
- De forma predeterminada, el conjunto de AWS KMS claves descifra los registros en modo estricto (en el que se especifican determinadas claves de KMS). Debe usar el ARN de una clave para identificar AWS KMS keys para descifrar.

  Al cifrar con un AWS KMS anillo de claves, el SDK de cifrado de AWS bases de datos almacena la clave ARN de la descripción del material junto con AWS KMS key la clave de datos cifrados. Al descifrar en modo estricto, el SDK de cifrado de AWS bases de datos comprueba que aparezca la misma clave ARN en el anillo de claves antes de intentar utilizar la clave de empaquetado para descifrar la clave de datos cifrados. Si utiliza un identificador de clave diferente, el SDK de cifrado de AWS bases de datos no lo reconocerá ni utilizará AWS KMS key, aunque los identificadores hagan referencia a la misma clave.
- Al descifrar en modo de descubrimiento, no especifique ninguna clave de encapsulación. En primer lugar, el SDK de cifrado de AWS bases de datos intenta descifrar el registro con la clave ARN almacenada en la descripción del material. Si eso no funciona, el SDK de cifrado de AWS bases de datos solicita AWS KMS descifrar el registro con la clave de KMS que lo cifró,

independientemente de quién sea el propietario de esa clave de KMS o de quién tenga acceso a ella.

Para especificar una clave AES sin procesar o un par de claves RSA sin procesar como clave de encapsulación en un conjunto de claves, debe especificar un espacio de nombres y un nombre. Al descifrar, debe utilizar exactamente el mismo espacio de nombres y el mismo nombre para cada clave de encapsulación sin procesar que utilizó al cifrar. Si usa un espacio de nombres o un nombre diferente, el SDK de cifrado de AWS bases de datos no reconocerá ni usará la clave de empaquetado, aunque el material de la clave sea el mismo.

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 el que 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.

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

En el siguiente ejemplo de Java se muestra cómo crear un filtro de detección. Antes de usar el código, sustituya los valores de ejemplo por valores válidos para su partición Cuenta de AWS y.

```
// Create the discovery filter
```

```
DiscoveryFilter discoveryFilter = DiscoveryFilter.builder()
        .partition("aws")
        .accountIds(111122223333)
        .build();
```

Trabajar con bases de datos de varios inquilinos

Con el SDK AWS de cifrado de bases de datos, puede configurar el cifrado del lado del cliente para las bases de datos con un esquema compartido aislando cada inquilino con materiales de cifrado distintos. Al considerar una base de datos de multitenencia, tómese un tiempo para revisar sus requisitos de seguridad y cómo podría afectarlos la multitenencia. Por ejemplo, el uso de una base de datos multiusuario podría afectar a su capacidad de combinar el SDK de cifrado de AWS bases de datos con otra solución de cifrado del lado del servidor.

Si tiene varios usuarios que realizan operaciones de cifrado en su base de datos, puede usar uno de los AWS KMS anillos de claves para proporcionar a cada usuario una clave distinta para utilizarla en sus operaciones criptográficas. Administrar las claves de datos de una solución de cifrado del cliente multitenencia puede resultar complicado. Recomendamos organizar los datos por inquilino siempre que sea posible. Si el inquilino se identifica mediante los valores de la clave principal (por ejemplo, la clave de partición de una tabla de Amazon DynamoDB), la administración de las claves resulta más sencilla.

Puede usar el anillo de AWS KMS claves para aislar a cada inquilino con un anillo de claves distinto y. AWS KMS AWS KMS keysSegún el volumen de AWS KMS llamadas realizadas por inquilino, es posible que desee utilizar el anillo de claves AWS KMS jerárquico para minimizar las llamadas a. AWS KMSEl anillo de clavesAWS KMS 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. Debe utilizar el conjunto de claves jerárquico para implementar un cifrado con capacidad de búsqueda en su base de datos AWS KMS .

Crear balizas firmadas

El SDK AWS de cifrado de bases de datos utiliza balizas estándar y balizas compuestas para proporcionar soluciones de cifrado con capacidad de búsqueda que le permiten buscar registros cifrados sin descifrar toda la base de datos consultada. Sin embargo, el SDK de cifrado AWS de bases de datos también admite balizas firmadas que se pueden configurar completamente a partir de

campos de texto sin formato. `SIGN_ONLY` Las balizas firmadas son un tipo de baliza compuesta que indexan y realizan consultas complejas en los campos `SIGN_ONLY`.

Por ejemplo, si tiene una base de datos de multitenencia, puede que desee crear una baliza firmada que le permita consultar en la base de datos los registros cifrados por la clave de un inquilino específico. Para obtener más información, consulte Consulta de balizas en una base de datos de multitenencia.

Debe utilizar el conjunto de claves AWS KMS jerárquico para crear balizas firmadas.

Para configurar una baliza firmada, proporcione los siguientes valores.

```
List<CompoundBeacon> compoundBeaconList = new ArrayList<>();
CompoundBeacon exampleSignedBeacon = CompoundBeacon.builder()
        .name("signedBeaconName")
        .split(".")
        .signed(signedPartList)
        .constructors(constructorList) // optional
        .build();
compoundBeaconList.add(exampleSignedBeacon);
```

Nombre de la baliza

El nombre que utilizas al consultar la baliza.

El nombre de una baliza firmada no puede ser el mismo nombre que el de un campo sin cifrar. No puede haber dos balizas con el mismo nombre de baliza.

Carácter dividido

El personaje que se usa para separar las partes que componen su baliza firmada.

El carácter dividido no puede aparecer en los valores de texto no cifrado de ninguno de los campos a partir de los que se construye la baliza firmada.

Lista de piezas firmadas

Identifique los `SIGN_ONLY` campos incluidos en la baliza firmada.

Cada parte debe incluir un nombre, una fuente y un prefijo. La fuente es el campo `SIGN_ONLY` que identifica la pieza. La fuente debe ser un nombre de campo o un índice que haga referencia al valor de un campo anidado. Si el nombre de la pieza identifica la fuente, puede omitirla y el SDK de cifrado de AWS bases de datos utilizará automáticamente el nombre como fuente.

Recomendamos especificar la fuente como nombre de la pieza siempre que sea posible. El prefijo puede ser cualquier cadena, pero debe ser único. Dos partes firmadas de una baliza firmada no pueden tener el mismo prefijo. Recomendamos utilizar un valor corto que distinga la parte de otras partes servidas por la baliza firmada. Para simplificar las consultas de balizas, también le recomendamos que identifique una parte con el mismo prefijo en todas las balizas en las que esté incluida y que evite usar el mismo prefijo para identificar diferentes piezas.

```
List<SignedPart> signedPartList = new ArrayList<>);
    SignedPart signedPartExample = SignedPart.builder()
        .name("signedFieldName")
        .prefix("S-")
        .build();
    signedPartList.add(signedPartExample);
```

Lista de constructores (opcional)

Identifique los constructores que definen las diferentes formas en que la baliza firmada puede ensamblar las piezas firmadas.

Si no especifica una lista de constructores, el SDK de cifrado de AWS bases de datos agrupa la baliza firmada con el siguiente constructor predeterminado.

- Todas las piezas firmadas en el orden en que se agregaron a la lista de piezas firmadas
- Todas las piezas son obligatorias

Constructores

Cada constructor es una lista ordenada de piezas del constructor que define una forma de ensamblar la baliza firmada. Las piezas del constructor se unen en el orden en que se agregan a la lista, con cada parte separada por el carácter dividido especificado.

Cada parte del constructor nombra una parte firmada y define si esa parte es obligatoria u opcional dentro del constructor. Por ejemplo, si desea consultar una baliza firmada sobre `Field1`, `Field1.Field2` y`Field1.Field2.Field3`, marcar `Field2` y `Field3` como opcional y crear un constructor.

Cada constructor debe tener como mínimo una pieza requerida. Recomendamos hacer que la primera parte de cada constructor sea obligatoria para poder usar el operador `BEGINS_WITH` en las consultas.

Un constructor tiene éxito si todas las piezas requeridas están presentes en el registro. Al escribir un registro nuevo, la baliza firmada utiliza la lista de constructores para determinar

si la baliza se puede ensamblar a partir de los valores proporcionados. Intente ensamblar la baliza en el orden en que se agregaron los constructores a la lista de constructores y utilice el primer constructor que lo haga correctamente. Si ningún constructor tiene éxito, la baliza no se graba en el registro.

Todos los lectores y redactores deben especificar el mismo orden de constructores para garantizar que los resultados de sus consultas sean correctos.

Utilice los siguientes procedimientos para especificar su propia lista de constructores.

1. Cree una pieza constructora para cada pieza firmada para definir si esa pieza es necesaria o no.

   El nombre de la pieza constructora debe ser el nombre del campo firmado.

   El siguiente ejemplo ilustra cómo crear una pieza de construcción para un campo firmado.

   ```
   ConstructorPart field1ConstructorPart = ConstructorPart.builder()
           .name("Field1")
           .required(true)
           .build();
   ```

2. Cree un constructor para cada una de las formas posibles de ensamblar la baliza firmada utilizando las partes constructoras que creó en el paso 1.

   Por ejemplo, si desea consultar sobre `Field1.Field2.Field3` y `Field4.Field2.Field3`, debe crear dos constructores. Tanto `Field1` como `Field4` pueden ser necesarios porque están definidos en dos constructores distintos.

   ```
   // Create a list for Field1.Field2.Field3 queries
   List<ConstructorPart> field123ConstructorPartList = new ArrayList<>();
   field123ConstructorPartList.add(field1ConstructorPart);
   field123ConstructorPartList.add(field2ConstructorPart);
   field123ConstructorPartList.add(field3ConstructorPart);
   Constructor field123Constructor = Constructor.builder()
           .parts(field123ConstructorPartList)
           .build();
   // Create a list for Field4.Field2.Field1 queries
   List<ConstructorPart> field421ConstructorPartList = new ArrayList<>();
   field421ConstructorPartList.add(field4ConstructorPart);
   field421ConstructorPartList.add(field2ConstructorPart);
   field421ConstructorPartList.add(field1ConstructorPart);
   ```

```
Constructor field421Constructor = Constructor.builder()
        .parts(field421ConstructorPartList)
        .build();
```

3. Cree una lista de constructores que incluya todos los constructores que creó en el Paso 2.

```
List<Constructor> constructorList = new ArrayList<>();
constructorList.add(field123Constructor)
constructorList.add(field421Constructor)
```

4. Especifique el `constructorList` cuando cree su baliza firmada.

Uso de los llaveros

Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

El SDK de cifrado de bases de datos de AWS utiliza llaveros para realizar el cifrado de sobre. Los conjuntos de claves generan, cifran y descifran claves de datos. Según el llavero que se use, se determinará el origen de las claves de datos únicas que protegen cada registro cifrado y las claves de encapsulación que cifran dichas claves de datos. Al cifrar especifica un llavero y al descifrar usa ese mismo llavero u otro llavero.

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

Le recomendamos que utilice un llavero que proteja sus claves de encapsulación y realice operaciones criptográficas dentro de un límite seguro, como el llavero AWS KMS, que utiliza AWS KMS keys que nunca deja AWS Key Management Service() AWS KMS sin cifrar. También puede escribir un llavero que utilice claves de encapsulación que se almacenen en sus módulos de seguridad de hardware (HSM) o que estén protegidos mediante otros servicios de clave maestra.

En este tema, se explica cómo utilizar la característica de llavero del SDK de cifrado de bases de datos de AWS y cómo elegir un llavero.

Temas

Cómo funcionan los llaveros

Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

Al cifrar y firmar un campo de la base de datos, el SDK de cifrado de bases de datos de AWS solicita al conjunto de claves los materiales de cifrado. Este devuelve una clave en texto no cifrado, una copia de dicha clave cifrada por cada una de las claves de encapsulación del llavero y una clave MAC que está asociada a la clave de datos. El SDK de cifrado de bases de datos de AWS utiliza la clave de datos de texto no cifrado para cifrar los datos y, a continuación, elimina la clave de datos de texto no cifrado de la memoria tan pronto como sea posible. A continuación, el SDK de cifrado de bases de datos de AWS agrega una descripción que incluye las claves de datos cifrados y otra información, como las instrucciones de cifrado y firma. El SDK de cifrado de bases de datos de AWS utiliza la clave MAC para calcular los códigos de autenticación de mensajes basados en hash (HMAC) mediante la canonicalización de la descripción del material y de todos los campos marcados con `ENCRYPT_AND_SIGN` o `SIGN_ONLY`.

Al descifrar datos, puede utilizar el mismo llavero que utilizó para cifrar los datos o uno diferente. Para descifrar los datos, un llavero de descifrado debe tener acceso al menos a una clave de empaquetado del llavero de cifrado.

El SDK de cifrado de bases de datos de AWS pasa las claves de datos cifrados de la descripción del material al llavero, y pide al llavero que descifre cualquiera de ellas. Este utiliza sus claves de encapsulación para descifrar una de las claves de datos cifradas y devuelve una clave de datos en texto no cifrado. El SDK de cifrado de bases de datos de AWS utiliza la clave de datos en texto no cifrado para descifrar los datos. Si ninguna de las claves de encapsulación del llavero puede descifrar ninguna de las claves de datos cifradas, se producirá un error en la operación de descifrado.

Puede utilizar un único llavero o además combinar llaveros del mismo tipo o de un tipo distinto en un llavero múltiple. Al cifrar los datos, el llavero múltiple devuelve una copia de la clave de datos cifrada por todas las claves de encapsulación en todos los llaveros que componen el llavero múltiple y una clave MAC que está asociada a la clave de datos. Puede descifrar los datos utilizando un llavero configurado con cualquiera de las claves de encapsulación del llavero múltiple.

Elegir un llavero

Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

Su llavero determina las claves de encapsulación que protegen sus claves de datos y, en última instancia, sus datos. Utilice las claves de encapsulación más seguras que resulten prácticas para su tarea. Siempre que sea posible, utilice las claves de encapsulación que están protegidas por un módulo de seguridad de hardware (HSM) o una infraestructura de administración de claves, como las claves KMS en AWS Key Management Service (AWS KMS) o claves de cifrado en AWS CloudHSM.

El SDK de cifrado de bases de datos de AWS proporciona varios llaveros y configuraciones de llaveros, y usted puede crear sus propios llaveros personalizados. También puede crear un llavero múltiple que incluya uno o más llaveros del mismo tipo o de uno diferente.

Temas

- Llaveros AWS KMS
- AWS KMS Llaveros jerárquicos
- Llaveros de AES sin formato
- Llaveros de RSA sin formato
- Llaveros múltiples

Llaveros AWS KMS

> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

Un AWS KMS llavero utiliza el cifrado simétrico o el RSA asimétrico AWS KMS keys para generar, cifrar y descifrar las claves de datos. AWS Key Management Service(AWS KMS) protege las claves KMS y realiza operaciones criptográficas dentro del límite del FIPS. Le recomendamos que utilice un llavero AWS KMS o un llavero con propiedades de seguridad similares, siempre que sea posible.

También puede usar una clave KMS simétrica de múltiples regiones en un llavero AWS KMS. Para obtener más información y ejemplos sobre el uso de multirregiones AWS KMS keys, consulte Uso de claves de múltiples regiones AWS KMS keys. Para obtener más información sobre las claves de multirregiones, consulte Uso de claves de multirregiones en la Guía para desarrolladores de AWS Key Management Service.

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

- 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 en texto no cifrado que generó la clave generadora. Los llaveros AWS KMS pueden tener cero o más claves adicionales.

Debe tener una clave generadora para cifrar los registros. Cuando un AWS KMS llavero tiene solo una AWS KMS clave, esa clave se usa para generar y cifrar la clave de datos.

Al igual que todos los conjuntos de claves, los llaveros AWS KMS se pueden utilizar de forma independiente o en un llavero múltiple con otros llaveros del mismo tipo o de otro tipo.

Temas

- Permisos necesarios para los llaveros AWS KMS
- Identificación de AWS KMS keys en un llavero de AWS KMS
- Crear un llavero AWS KMS
- Uso de claves de múltiples regiones AWS KMS keys
- Uso de un llavero de detección de AWS KMS
- Uso de un llavero de detección regional de AWS KMS

Permisos necesarios para los llaveros AWS KMS

El SDK de cifrado de bases de datos de AWS no requiere una Cuenta de AWS ni depende de ningún Servicio de AWS. Sin embargo, para usar un AWS KMS llavero, necesita tener un Cuenta de AWS y los siguientes permisos mínimos en el AWS KMS keys del llavero.

- Para cifrar con un llavero AWS KMS, necesita el permiso kms:GenerateDataKey en la clave generadora. Necesita el permiso kms:Encrypt en todas las claves adicionales del llavero AWS KMS.
- Para descifrar con un llavero AWS KMS, necesita el permiso kms:Decrypt en una clave del AWS KMS como mínimo.
- Para cifrar con un conjunto de claves múltiple compuesto por llaveros AWS KMS, necesita el permiso kms:GenerateDataKey en la clave generadora del llavero generador. Necesita el permiso kms:Encrypt en todas las demás claves de los demás llaveros AWS KMS.

Para obtener información detallada acerca de los permisos para AWS KMS keys, consulte Autenticación y control de acceso en la Guía para desarrolladores deAWS Key Management Service.

Identificación de AWS KMS keys en un llavero de AWS KMS

Un llavero AWS KMS puede contener una o más AWS KMS keys. Para especificar una AWS KMS key en un llavero AWS KMS, utilice un identificador de claves de AWS KMS compatible. Los identificadores de clave que puede utilizar para identificar una AWS KMS key de un conjunto de claves varían según la operación y la implementación de lenguaje. Para obtener más información sobre los identificadores de clave de un AWS KMS key, consulte Identificadores de 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.

- Para cifrar con un llavero AWS KMS, puede utilizar una ID de clave, un ARN de clave, un nombre de alias o un ARN de alias para cifrar los datos.

  > **Note**
  >
  > Si especifica un nombre de alias o un ARN de alias para una clave de KMS en un llavero 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 la clave KMS utilizada para descifrar las claves de datos cifrados.

- Para descifrar con un llavero AWS KMS, debe usar el ARN de una clave para identificar AWS KMS keys. Para obtener más información, consulte Selección de las claves de encapsulación.
- En un llavero usado para cifrar y descifrar, debe usar el ARN de una clave para identificar AWS KMS keys.

Al descifrar, el SDK de cifrado de bases de datos de AWS busca en el llavero AWS KMS de una AWS KMS key que pueda descifrar una de las claves de datos cifrados. En concreto, el SDK de cifrado de bases de datos de AWS utiliza el siguiente patrón para cada clave de datos cifrada de la descripción del material.

- El SDK de cifrado de bases de datos de AWS obtiene la clave ARN de la AWS KMS key que cifró la clave de datos de los metadatos de la descripción del material.

- El SDK de cifrado de bases de datos de AWS busca en el llavero de descifrado una AWS KMS key con el ARN de una clave coincidente.
- Si encuentra una AWS KMS key con el ARN de una clave coincidente en el llavero, el SDK de cifrado de bases de datos de AWS solicita al AWS KMS utilizar la clave KMS para descifrar la clave de datos cifrada.
- De lo contrario, pasa a la siguiente clave de datos cifrada, si la hay.

Crear un llavero AWS KMS

Puede configurar cada llavero AWS KMS con una única AWS KMS key o varias AWS KMS keys en la misma o diferente Cuentas de AWS y Regiones de AWS. La AWS KMS key debe ser una clave de cifrado simétrico (`SYMMETRIC_DEFAULT`) o una clave asimétrica RSA KMS. También puede utilizar una clave KMS de múltiples regiones de cifrado simétrico. Puede utilizar uno o varios llaveros AWS KMS en un llavero múltiple.

Puede crear un AWS KMS llavero que cifre y descifre los datos, o puede crear llaveros AWS KMS específicos para cifrar o descifrar. Cuando crea un llavero AWS KMS para cifrar datos, debe especificar una clave generadora, que es una AWS KMS key que se utiliza para generar una clave de datos en texto no cifrado y la cifra. La clave de datos no está relacionada matemáticamente con la clave KMS. Luego, si decide hacerlo, puede especificar más AWS KMS keys que cifren la misma clave de datos en texto no cifrado. Para descifrar un campo cifrado protegido por este llavero, el llavero de descifrado que utilice debe incluir al menos uno de los AWS KMS keys definidos en el llavero o no AWS KMS keys. (Un llavero AWS KMS sin AWS KMS keys se conoce como un llavero AWS KMS de detección.)

Todas las claves de empaquetado de un llavero de cifrado o de varios llaveros deben poder cifrar la clave de datos. Si alguna clave de encapsulación no se cifra, el método de cifrado falla. Como resultado, la persona que llama debe tener los permisos requeridos para todas las claves del llavero. Si utiliza un llavero de detección para cifrar los datos, solo o en un llavero múltiple, la operación de cifrado no se realizará correctamente.

En el siguiente ejemplo de Java, se utiliza un `CreateAwsKmsMrkMultiKeyring` método para crear un AWS KMS llavero con una clave KMS de cifrado simétrico. El método `CreateAwsKmsMrkMultiKeyring` garantiza que el llavero maneje correctamente las claves de una sola región y de múltiples regiones. El ejemplo utiliza una clave ARN para identificar la clave KMS.

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

```
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
final CreateAwsKmsMrkMultiKeyringInput keyringInput =
 CreateAwsKmsMrkMultiKeyringInput.builder()
        .generator(kmsKeyArn)
        .build();
final IKeyring kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

Uso de claves de múltiples regiones AWS KMS keys

Puede utilizar AWS KMS keys de múltiples regiones como claves de empaquetado en el SDK de cifrado de bases de datos de AWS. Si cifra con una clave de múltiples regiones en una Región de AWS, puede descifrar utilizando una clave de múltiples regiones relacionada en una diferente Región de AWS.

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

Para admitir claves de múltiples regiones, el SDK de cifrado de bases de datos de AWS incluye llaveros AWS KMS compatibles con múltiples regiones. El método `CreateAwsKmsMrkMultiKeyring` admite claves de una sola región y de múltiples regiones.

- En el caso de las claves de una sola región, el símbolo compatible con múltiples regiones se comporta igual que el llavero AWS KMS de una sola región. Intenta descifrar el texto cifrado únicamente con la clave de región única que cifró los datos. Para simplificar su AWS KMS experiencia con los llaveros, le recomendamos que utilice el método `CreateAwsKmsMrkMultiKeyring` siempre que utilice una clave KMS de cifrado simétrico.
- En el caso de las claves de múltiples regiones, el símbolo de Múltiples regiones intenta descifrar el texto cifrado con la misma clave de múltiples regiones que cifró los datos o con la clave múltiples regiones relacionada en la región que especifique.

En los llaveros compatibles con múltiples regiones que utilizan más de una clave KMS, puede especificar varias claves de una o múltiples regiones. Sin embargo, solo puede especificar una clave

de cada conjunto de claves de múltiples regiones relacionadas. Si especifica más de un identificador de clave con el mismo ID de clave, se produce un error en la llamada al constructor.

En el siguiente ejemplo de Java, se crea un llavero AWS KMS con una clave KMS de múltiples regiones El ejemplo especifica una clave de múltiples regiones como clave generadora y una clave de región única como clave secundaria

```
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
final CreateAwsKmsMrkMultiKeyringInput createAwsKmsMrkMultiKeyringInput =
    CreateAwsKmsMrkMultiKeyringInput.builder()
            .generator(multiRegionKeyArn)
            .kmsKeyIds(Collections.singletonList(kmsKeyArn))
            .build();
IKeyring awsKmsMrkMultiKeyring =
 matProv.CreateAwsKmsMrkMultiKeyring(createAwsKmsMrkMultiKeyringInput);
```

Cuando utiliza llaveros AWS KMS de múltiples regiones, puede descifrar el texto cifrado en modo estricto o en modo de descubrimiento. Para descifrar el texto cifrado en modo estricto, cree una instancia del símbolo de múltiples regiones con la clave ARN de la clave de múltiples regiones relacionada en la región en la que esté descifrando el texto cifrado. Si especifica la clave ARN de una clave de múltiples regiones relacionada en una región diferente (por ejemplo, la región en la que se cifró el registro), el símbolo de múltiples regiones realizará una llamada entre regiones para esa AWS KMS key.

Al descifrar en modo estricto, el símbolo compatible con múltiples regiones requiere un ARN clave. Solo acepta un ARN de clave de cada conjunto de claves de varias regiones relacionadas.

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 AWS KMS llaveros de detección de una sola región, consulte Uso de un llavero de detección de AWS KMS.)

Si ha cifrado con una clave de múltiples regiones, el símbolo de múltiples regiones del modo de detección intentará descifrarlo utilizando una clave de múltiples regiones relacionada en la región local. Si no existe ninguno, se produce un error en la llamada. En el modo de detección, el SDK de cifrado de bases de datos de AWS no intentará realizar una llamada entre regiones para obtener la clave de múltiples regiones utilizada para el cifrado.

Uso de un llavero de detección de AWS KMS

Al descifrar, se recomienda especificar las claves de empaquetado que puede utilizar el SDK de cifrado de bases de datos de AWS. Para seguir esta práctica recomendada, utilice un llavero AWS KMS de descifrado que limite las claves de encapsulación AWS KMS a las que usted especifique. Sin embargo, también puede crear un AWS KMS llavero de detección; es decir, un llavero AWS KMS que no especifique ninguna clave de encapsulación.

El SDK de cifrado de bases de datos de AWS proporciona un conjunto de claves AWS KMS de detección estándar y un conjunto de claves AWS KMS de detección para claves de múltiples regiones. Para obtener información sobre cómo usar claves de varias regiones con el SDK de cifrado de bases de datos de AWS, consulte Uso de claves de múltiples regiones AWS KMS keys.

Como no especifica ninguna clave de encapsulación, un conjunto de claves de detección no puede cifrar los datos. Si utiliza un llavero de detección para cifrar los datos, solo o en un llavero múltiple, la operación de cifrado no se realizará correctamente.

Al descifrar, un conjunto de claves de detección permite al SDK de cifrado de bases de datos de AWS solicitar a AWS KMS el descifrado de cualquier clave de datos cifrada utilizando la AWS KMS key que la cifró, independientemente de quién sea su propietario o tenga acceso a esa 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 conjunto de claves de AWS KMS detección en un conjunto de claves de descifrado múltiple, el conjunto de claves de detección anula todas las restricciones de claves de KMS especificadas en otros conjuntos de claves del conjunto de claves múltiples. El conjunto de claves múltiples se comporta como el menos restrictivo. Si utiliza un llavero de detección para cifrar datos, solo o en un llavero múltiple, la operación de cifrado no se realizará correctamente

El SDK de cifrado de bases de datos de AWS proporciona un llavero AWS KMS de detección por conveniencia. No obstante, recomendamos que utilice un llavero más limitado siempre que sea posible por los siguientes motivos.

- Autenticidad: un AWS KMS llavero de detección puede utilizar cualquier clave de datos AWS KMS key utilizada en la descripción del material, siempre que la persona que llama tenga permiso

para utilizarla AWS KMS key para descifrarla. Podría no ser la AWS KMS key que el intermediario quiere utilizar. Por ejemplo, una de las claves de datos cifrada podría haberse cifrado con una AWS KMS key menos segura que cualquiera pudiese utilizar.

- Latencia y rendimiento: un conjunto de claves AWS KMS de detección puede ser perceptiblemente más lento que otros, ya que el SDK de cifrado de bases de datos de AWS intenta descifrar todas las claves de datos cifradas, incluidas las AWS KMS keys cifradas en otros Cuentas de AWS y regiones y AWS KMS keys que la persona que llama no tiene permiso para utilizarlas para el descifrado.

Si utiliza un conjunto de claves de detección, le recomendamos que utilice un filtro de detección para limitar las claves KMS que se pueden utilizar a las especificadas Cuentas de AWS y particiones. Para obtener ayuda para encontrar el ID y la partición de su cuenta, consulte Sus identificadores Cuenta de AWS y el formato ARN en el Referencia general de AWS.

El siguiente código Java crea una instancia de un llavero de AWS KMS detección con un filtro de detección que limita las claves de KMS que el SDK de cifrado de bases de datos de AWS puede utilizar a las de la partición de `aws` y 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 `aws-cn` partición. Si las claves KMS están dentro AWS GovCloud (US) Regions, use el valor de la `aws-us-gov` partición. Para todos los demás Regiones de AWS, utilice el valor de la `aws` partición.

```
// 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);
```

Uso de un llavero de detección regional de AWS KMS

Un llavero de AWS KMSdetección regional es un conjunto de claves que no especifica los ARN de las claves de KMS. En su lugar, permite que el SDK de cifrado de bases de datos de AWS descifre utilizando únicamente las claves de KMS, en Regiones de AWS específicos.

Al descifrar con un llavero de detección regional de AWS KMS, el SDK de cifrado de bases de datos de AWS descifra cualquier clave de datos cifrada que se cifró con una AWS KMS key en la región de Región de AWS especificada. Para tener éxito, el intermediario debe disponer permiso de `kms:Decrypt` en al menos una de las AWS KMS keys en el Región de AWS especificado que 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 campos cifrados. Si utiliza un llavero 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 de múltiples regiones 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 llavero de descifrado múltiple, el conjunto de claves de detección regional anula todas las restricciones de claves de KMS especificadas por otros llaveros de los llaveros múltiples. El llavero múltiples se comporta como el menos restrictivo. Un llavero de detección AWS KMS no afecta el cifrado cuando se utiliza solo o en un llavero múltiple.

El llavero de detección regional del SDK de cifrado de bases de datos de AWS intenta descifrar únicamente con las claves de KMS de la región especificada. Cuando se utiliza un conjunto de claves de detección, se configura la región en el cliente AWS KMS. Estas implementaciones del SDK de cifrado de bases de datos de AWS no filtran las claves de KMS por región, pero AWS KMS no permite 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

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

```
// 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);
```

AWS KMS Llaveros jerárquicos

Se cambió el nombre de nuestra biblioteca de cifrado del lado del cliente por el de SDK de cifrado de bases de AWS datos. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

> **Note**
>
> A partir del 24 de julio de 2023, no se admiten las claves de rama creadas durante la versión preliminar para desarrolladores. Cree nuevas claves de rama para seguir usando el almacén de claves de rama que creó durante la versión preliminar para desarrolladores.

Con el conjunto 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 un registro. 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 DynamoDB sirve como almacén de claves de rama 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 campo y cifra cada clave de datos con una clave de encapsulación única derivada de la clave de rama activa. El conjunto de claves jerárquico depende de la jerarquía establecida entre las claves de rama activas y las claves de encapsulación 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 anillo de claves jerárquico realiza una AWS KMS llamada para descifrar la clave de rama y reunir los materiales de la clave de rama 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 consta de dos particiones, una para las operaciones de cifrado y otra para las operaciones de descifrado. La partición de cifrado almacena los materiales de clave de rama recopilados a partir de la clave de rama activa y los reutiliza para todas las operaciones de cifrado hasta que venza el límite de memoria caché. La partición de descifrado almacena los materiales de clave de rama reunidos para otras versiones de claves de rama identificadas en las operaciones de

descifrado. La partición de descifrado puede almacenar varias versiones de materiales de claves de rama activas a la vez. Cuando se configura para usar un proveedor de ID de clave de rama para una base de datos de multitenencia, la partición de cifrado también puede almacenar varias versiones de materiales de clave de rama a la vez. Para obtener más información, consulte Uso del conjunto de claves jerárquico con bases de datos de multitenencia.

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

Temas

Cómo funcionan

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 deAWS 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 conjunto de claves continúa con el paso 5.

2. Si no hay ningún material de clave de rama válido, el conjunto de claves jerárquico consulta el almacén de claves de rama en busca de la clave de rama activa.

   a. El almacén de claves de bifurcación llama AWS KMS para desencriptar la clave de bifurcación activa 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 de rama devuelve la clave de rama en texto no cifrado 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 encapsulación 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 y firmar el registro. Para obtener más información sobre cómo se cifran y firman los registros en el SDK de cifrado de bases de datos de AWS , consulte Encrypt and sign.

Descifrado y verificación

En el siguiente tutorial, se 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 cifrada en el campo de descripción del material del registro 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 jerárquico consulte el almacén de claves de rama para encontrar la clave de rama que coincida con la versión de clave de rama identificada en el paso 2.

    a. El almacén de claves de bifurcación llama AWS KMS para desencriptar 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 de rama devuelve la clave de rama en texto no cifrado 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 encapsulación 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 de texto no cifrado para descifrar y verificar el registro. Para obtener más información sobre cómo se descifran y verifican los registros en el SDK de cifrado de AWS bases de datos, consulte Descifrar y verificar.

Requisitos previos

El SDK AWS de cifrado de bases de datos no requiere ni depende de ninguno. Cuenta de AWS Servicio de AWSSin embargo, el conjunto de claves jerárquico depende de Amazon AWS KMS DynamoDB.

Para utilizar un conjunto de claves jerárquico, necesita un cifrado simétrico con permisos de KMS:Decrypt. AWS KMS key También puede utilizar una clave multirregional de cifrado simétrico. Para obtener información detallada acerca de los permisos para AWS KMS keys, consulte Autenticación y control de acceso en la Guía para desarrolladores deAWS Key Management Service .

Antes de poder crear y usar un conjunto de claves jerárquico, debe crear su almacén de claves de rama y rellenarlo con la primera clave de rama activa.

Paso 1: Configurar un nuevo servicio de almacenamiento de claves

El servicio de almacén de claves ofrece varias operaciones, como `CreateKeyStore` y `CreateKey`, para ayudarle a reunir los requisitos previos del conjunto de claves jerárquico y a gestionar el almacén de claves de su rama.

El siguiente ejemplo de Java crea un servicio de almacén de claves. Debe especificar un nombre de tabla de DynamoDB que sirva como nombre del almacén de claves de rama, un nombre lógico para el almacén de claves de rama y el ARN de clave de KMS que identifica la clave de KMS que protegerá las claves de rama.

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. El nombre del almacén de claves lógicas puede ser el mismo que el nombre de la tabla de DynamoDB, pero no tiene por qué serlo. Se recomienda encarecidamente especificar el nombre de la tabla de DynamoDB como nombre de la tabla lógica cuando configure por primera vez el servicio de almacén de claves. Debe especificar siempre el mismo nombre de tabla lógica. En caso de que el nombre del almacén de claves de la rama cambie después de restaurar la tabla de DynamoDB a partir de una copia de seguridad, el nombre del almacén de claves lógico se asigna al nombre de la tabla de DynamoDB que especifique para garantizar que el conjunto de claves jerárquico pueda seguir accediendo al almacén de claves de la rama.

```
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();
```

Paso 2: llamar a **`CreateKeyStore`** para crear un almacén de claves de rama

La siguiente operación de Java crea el almacén de claves de rama que persistirá y protegerá sus claves de rama.

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

La operación `CreateKeyStore` crea una tabla de DynamoDB con el nombre de tabla que especificó en el Paso 1 y los siguientes valores obligatorios.

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

Paso 3: Llamar a **`CreateKey`** para crear una nueva clave de rama activa

La siguiente operación de Java crea una nueva clave de rama activa con la clave de KMS que especificó en el Paso 1 y agrega la clave de rama activa a la tabla de DynamoDB que creó en el Paso 2.

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 adicionales (AAD) en el contexto de cifrado incluido en la llamada kms:. GenerateDataKeyWithoutPlaintext

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

```
final Map<String, String> additionalEncryptionContext =
 Collections.singletonMap("contextKey",
        "contextValue");

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

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 branch key store",
      "kms-arn" : the KMS key ARN,
      "hierarchy-version" : "1",
      "aws-crypto-ec:contextKey": "contextValue"
   },
   "KeyId": "the KMS key ARN you specified in Step 1",
   "NumberOfBytes": "32"
}
```

> **Note**
>
> La operación `CreateKey` crea una clave de rama activa y una clave de baliza, incluso si no ha configurado la base de datos para el cifrado con capacidad de búsqueda. Ambas claves se almacenan en el almacén de claves de su rama. Para obtener más información, consulte Uso del conjunto de claves jerárquico para el cifrado para búsquedas.

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"
}
```

Crear un conjunto de claves jerárquico

Para inicializar el conjunto de claves jerárquico, debe proporcionar los siguientes valores:

- Un nombre de almacén de claves de rama

  Nombre de la tabla de DynamoDB que creó como almacén de claves de sucursal.

-

  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 valor debe ser mayor que cero. Cuando el TTL límite de caché vence, la entrada se expulsa de la caché local.

- Un identificador de clave de rama

  La `branch-key-id` que identifica la clave de rama activa en su almacén de clave de rama.

  > **Note**
  >
  > Para inicializar el conjunto de claves jerárquico para su uso por varios inquilinos, debe especificar un proveedor de ID de sucursal en lugar de una `branch-key-id`. Para obtener más información, consulte Uso del conjunto de claves jerárquico con bases de datos de multitenencia.

- (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.

El siguiente ejemplo de Java muestra cómo inicializar un conjunto de claves jerárquico con el SDK de cifrado de bases de datos de AWS para el cliente DynamoDB. El siguiente ejemplo especifica un TTL con un límite de caché de 600 segundos.

```
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);
```

Rote la clave de rama activa

Solo puede haber una versión activa para cada clave de rama a la vez. El conjunto de claves jerárquico suele utilizar cada versión de clave de rama activa 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 rama activa no se utilizarán para realizar operaciones de cifrado ni para obtener nuevas claves de encapsulamiento. Sin embargo, aún se pueden consultar y proporcionar claves de encapsulamiento para descifrar las claves de datos que cifraron mientras estaban activas.

Utilice la operación `VersionKey` del servicio de almacenamiento de claves para rotar la clave de rama 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 clave de rama activa actual cuando llame a `VersionKey`.

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

```
        .build()
);
```

Uso del conjunto de claves jerárquico con bases de datos de multitenencia

Puede utilizar la jerarquía de claves establecida entre las claves de rama activas y sus claves de encapsulación derivadas para admitir bases de datos de multitenencia creando una clave de rama para cada inquilino de la base de datos. A continuación, el conjunto de claves jerárquico cifra y firma todos los datos de un inquilino determinado con su clave de rama distinta. Esto le permite almacenar los datos de multitenencia en una única base de datos y aislar los datos de los inquilinos por clave de rama.

Cada inquilino tiene su propia clave de rama que se define mediante una clave única `branch-key-id`. Solo puede haber una versión activa de cada `branch-key-id` a la vez.

Antes de poder inicializar su conjunto de claves jerárquico para su uso multitenencia, debe crear una clave de rama para cada inquilino y crear un proveedor de ID de clave de sucursal. Use el identificador de la clave de rama para crear un nombre amigable para su `branch-key-ids` y así poder reconocer fácilmente el nombre correcto `branch-key-id` para el inquilino. 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.

En primer lugar, siga los pasos 1 y 2 de los procedimientos de Requisitos previos A continuación, utilice los siguientes procedimientos para crear una clave de rama para cada inquilino, crear un proveedor de ID de clave de rama e inicializar su conjunto de claves jerárquicas para su uso por varios inquilinos.

Paso 1: Cree una clave de rama para cada inquilino de la base de datos

Llamar a `CreateKey` de cada inquilino en su base de datos.

La siguiente operación de Java crea dos claves de rama con la clave de KMS que especificó al crear el servicio de almacén de claves y agrega las claves de rama a la tabla de DynamoDB que

creó para que sirva como almacén de claves de rama. La misma clave de KMS debe proteger todas las claves de rama.

```
final String tenant1BranchKey = keystore.CreateKey(
        CreateKeyInput.builder().build()).branchKeyIdentifier();
final String tenant2BranchKey = keystore.CreateKey(
        CreateKeyInput.builder().build()).branchKeyIdentifier();
```

Paso 2: crear un proveedor de ID de sucursal

En el siguiente ejemplo de Java se crean nombres descriptivos para las dos claves de rama creadas en el paso 1 y se pide `CreateDynamoDbEncryptionBranchKeyIdSupplier` la creación de un proveedor de ID de clave de rama con el SDK de cifrado de AWS bases de datos para el cliente DynamoDB.

```
// 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();
```

Paso 3: inicialice su conjunto de claves jerárquico con el proveedor de ID de clave de rama

Para inicializar el conjunto de claves jerárquico, debe proporcionar los siguientes valores:

- Un nombre de almacén de claves de rama
- Un tiempo límite de vida de la memoria caché (TTL)
- Un proveedor de ID de clave de rama

- (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 tipo de caché define el modelo de subprocesamiento. El conjunto de claves jerárquico proporciona tres tipos de caché que admiten bases de datos multiusuario: predeterminada,,. MultiThreaded StormTracking

  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.

  Default (Recommended)

  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 rama, la caché predeterminada evita que varios subprocesos llamen al sistema AWS KMS notificando a un subproceso que la entrada de materiales de clave de rama va a caducar con 10 segundos de antelación. Esto garantiza que solo un subproceso envíe una solicitud AWS KMS para actualizar la caché.

  Para inicializar el conjunto de claves jerárquico con una caché predeterminada, especifique el siguiente valor:

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

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

  La caché predeterminada y la StormTracking caché admiten el mismo modelo de subprocesos, pero solo es necesario especificar la capacidad de entrada para inicializar el conjunto de claves jerárquico con la caché predeterminada. Para personalizaciones de caché más detalladas, utilice la caché. StormTracking

MultiThreaded

La MultiThreaded memoria caché se puede utilizar de forma segura 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 inicializar el conjunto de claves jerárquico con una MultiThreaded memoria 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.

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

StormTracking

La StormTracking memoria caché está diseñada para ser compatible con 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 inicializar el conjunto de claves jerárquico con una 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.
- 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: 20 segundos

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

  Valor predeterminado: 20 milisegundos

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

- (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.

El siguiente ejemplo de Java inicializa un conjunto de claves jerárquico con el proveedor de claves de rama creado en el Paso 2, un TLL con un límite de caché de 600 segundos y un tamaño máximo de caché de 1000. En este ejemplo, se inicializa un conjunto de claves jerárquico con el SDK de cifrado de AWS bases de datos para el cliente DynamoDB.

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

Uso del conjunto de claves jerárquico para el cifrado para búsquedas

El cifrado para búsquedas le permite buscar registros cifrados sin necesidad de descifrar toda la base de datos. Esto se logra indexando el valor de texto no cifrado de un campo cifrado con una baliza. Para implementar un cifrado para búsquedas, debe utilizar un conjunto de claves jerárquico.

La operación `CreateKey` de almacenamiento de claves genera tanto una clave de rama como una clave de baliza. La clave de rama se utiliza en las operaciones de cifrado y descifrado de registros. La clave de baliza se utiliza para generar balizas.

La clave de rama y la clave de baliza están protegidas por lo mismo AWS KMS key que especificó al crear el servicio de almacenamiento de claves. Una vez que la `CreateKey` operación llama AWS KMS para generar la clave de sucursal, llama a kms: GenerateDataKeyWithoutPlaintext una segunda vez para generar la clave de baliza 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 branch key store",
      "kms-arn" : the KMS key ARN,
      "hierarchy-version" : 1
   },
   "KeyId": "the KMS key ARN",
   "NumberOfBytes": "32"
}
```

Tras generar ambas claves, la `CreateKey` operación llama a ddb: TransactWriteItems para escribir dos nuevos elementos que conservarán la clave de rama y la clave de baliza en tu almacén de claves de sucursal.

Al configurar una baliza estándar, el SDK de cifrado de AWS bases de datos consulta la clave de baliza en el almacén de claves de la sucursal. A continuación, utiliza una función de derivación de extract-and-expand claves (HKDF) basada en HMAC para combinar la clave de baliza con el nombre de la baliza estándar y crear la clave HMAC para una baliza determinada.

A diferencia de las claves de rama, solo hay una versión de clave de baliza por `branch-key-id` en un almacén de claves de rama. La clave de la baliza nunca se rota.

Definir la fuente de claves de baliza

Al definir la versión de la baliza para las balizas estándar y compuestas, debe identificar la clave de la baliza y definir un tiempo de vida útil (TTL) límite de caché para los materiales de la clave de la baliza. Los materiales de las claves de baliza se almacenan en una caché local independiente de las claves de rama. El siguiente fragmento muestra cómo definir la `keySource` para la base de datos de un solo inquilino. Identifique la clave de su baliza por el `branch-key-id` que está asociada.

```
keySource(BeaconKeySource.builder()
        .single(SingleKeyStore.builder()
                .keyId(branch-key-id)
                .cacheTTL(6000)
                .build())
        .build())
```

Definición de la fuente de la baliza en una base de multitenencia

Si tiene una base de datos de multitenencia, debe especificar los siguientes valores al configurar la `keySource`.

-

  keyFieldName

  Define el nombre del campo que almacena la clave `branch-key-id` asociada a la baliza utilizada para generar las balizas para un inquilino determinado. El `keyFieldName` puede ser cualquier cadena, pero debe ser única para todos los demás campos de la base de datos. Cuando se escriben nuevos registros en la base de datos, en este campo se almacena la `branch-key-id` de baliza utilizada para generar las balizas de ese registro. Debe incluir este campo en sus consultas de baliza e identificar los materiales clave de baliza adecuados necesarios para volver a calcular la baliza. Para obtener más información, consulte Consulta de balizas en una base de datos de multitenencia.

- CacheTTL

  El tiempo en segundos que se puede utilizar una entrada de materiales clave de baliza en la caché de balizas local antes de que caduque. Este valor debe ser mayor que cero. Cuando el TTL límite de caché vence, la entrada se expulsa de la caché local.

- (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 tipo de caché define el modelo de subprocesamiento. El conjunto de claves jerárquico proporciona tres tipos de caché que admiten bases de datos multiusuario: predeterminada,,. MultiThreaded StormTracking

  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.

  Default (Recommended)

  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 rama, la caché

predeterminada evita que varios subprocesos llamen al sistema AWS KMS notificando a un subproceso que la entrada de materiales de clave de rama va a caducar con 10 segundos de antelación. Esto garantiza que solo un subproceso envíe una solicitud AWS KMS para actualizar la caché.

Para inicializar el conjunto de claves jerárquico con una caché predeterminada, especifique el siguiente valor:

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

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

La caché predeterminada y la StormTracking caché admiten el mismo modelo de subprocesos, pero solo es necesario especificar la capacidad de entrada para inicializar el conjunto de claves jerárquico con la caché predeterminada. Para personalizaciones de caché más detalladas, utilice la caché. StormTracking

MultiThreaded

La MultiThreaded memoria caché se puede utilizar de forma segura 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 inicializar el conjunto de claves jerárquico con una MultiThreaded memoria 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.

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

```
        .build())
```

StormTracking

La StormTracking memoria caché está diseñada para ser compatible con 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 inicializar el conjunto de claves jerárquico con una 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.
- 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: 20 segundos

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

  Valor predeterminado: 20 milisegundos

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

```
keySource(BeaconKeySource.builder()
        .multi(MultiKeyStore.builder()
                .keyFieldName(beaconKeys)
                .cacheTTL(6000)
                .cache(CacheType.builder() // OPTIONAL
                        .Default(DefaultCache.builder()
                        .entryCapacity(100)
                        .build())
                .build())
        .build())
```

Llaveros de AES sin formato

Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

El SDK de cifrado de bases de datos de AWS le permite utilizar una clave simétrica AES que se proporciona como clave de empaquetado para proteger la 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 AES sin procesar cuando necesite proporcionar la clave de encapsulación 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 encapsulación que especifique como matriz de bytes para cifrar claves de datos. Puede especificar una sola clave de encapsulación en cada conjunto de claves de AES sin formato, pero puede incluir varios conjuntos de claves de AES sin formato en cada llavero múltiple.

Nombres y espacios de nombres clave

Para identificar la clave de encapsulación, el llavero de AES sin formato utiliza un nombre de espacio de clave y nombre de clave que usted proporcione. Estos valores no son secretos. Aparecen en texto plano en la descripción del material que el SDK de cifrado de bases de datos de AWS agrega al registro. 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 del. `JceMasterKey`

Si crea diferentes conjuntos de claves para cifrar y descifrar un campo 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 AES sin procesar con el espacio de nombres y `HSM_01` el nombre de la clave `AES_256_012`. A continuación, utilice ese llavero para cifrar algunos datos. Para descifrar esos datos, cree un conjunto de claves AES sin procesar con el mismo espacio de nombres, nombre de clave y material de clave.

En el siguiente ejemplo de Java se muestra cómo crear un llavero AES sin procesar. La `AESWrappingKey` variable representa el material clave que proporciona.

```
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);
```

Llaveros de RSA sin formato

Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

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 llavero 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 RSA sin procesar con una clave pública y una privada, asegúrese de que formen parte del mismo par de claves.

El llavero de RSA sin formato es equivalente a JceMasterKey en el SDK de cifrado de AWS para Java e interactúa con él, cuando se utilizan con claves de cifrado asimétrico de RSA.

> **Note**
>
> El conjunto de claves RSA no admite claves de KMS asimétricas. Para usar claves RSA KMS asimétricas, cree un llavero AWS KMS.

Espacios de nombres y nombres

Para identificar el par de claves, el conjunto de claves de RSA sin formato utiliza un nombre de espacio de clave y nombre de clave que usted proporcione. Estos valores no son secretos. Aparecen

en texto plano en la descripción del material que el SDK de cifrado de bases de datos de AWS agrega al registro. 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 de clave y el nombre de clave son equivalentes a los campos ID de proveedor (o proveedor) e ID de clave del. `JceMasterKey`

Si crea diferentes conjuntos de claves para cifrar y descifrar un registro 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 RSA sin procesar para una clave pública RSA con el espacio de nombres y el nombre de la clave.`HSM_01` `RSA_2048_06` Para descifrar esos datos, cree un conjunto de claves 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 conjuntos de claves RSA sin procesar que se utilizan para el cifrado y el descifrado, o utilizar las funciones de la implementación de su lenguaje que lo especifiquen para usted.

AWS Encryption SDKAdmite los siguientes modos de relleno, sujetos a las limitaciones de cada lenguaje. Recomendamos un modo de relleno OAEP, especialmente el OAEP con SHA-256 y el MGF1 con relleno SHA-256. El modo de relleno PKCS1 solo se admite por motivos de compatibilidad con versiones anteriores.

- OAEP con SHA-1 y MGF1 con relleno SHA-1
- OAEP con SHA-256 y MGF1 con relleno SHA-256
- OAEP con relleno SHA-384 y MGF1 con relleno SHA-384

- OAEP con SHA-512 y MGF1 con relleno SHA-512
- Relleno PKCS1 v1.5

El siguiente ejemplo de Java muestra 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 MGF1 con el modo de relleno SHA-256. `RSAPublicKey`y `RSAPrivateKey`Las variables y representan el material clave que proporciona.

```
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);
```

Llaveros múltiples

> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

Puede combinar llaveros en un llavero múltiple. Un llavero múltiple es un llavero que consta de uno o varios llaveros individuales del mismo tipo o de un tipo distinto. El efecto equivale a utilizar varios llaveros en una serie. Cuando se utiliza un llavero múltiple para cifrar datos, cualquiera de las claves de encapsulación en cualquiera de los llaveros puede descifrar dichos datos.

Cuando crea un llavero múltiple para cifrar datos, designa uno de los llaveros como llavero generador. Los llaveros restantes se conocen como llaveros secundarios. El llavero generador genera y cifra la clave de datos de texto no cifrado. A continuación, todas las claves de encapsulación de todos los llaveros secundarios cifran la misma clave de datos en texto no cifrado. El llavero múltiple devuelve la clave de texto no cifrado y una clave de datos cifrada para cada clave de encapsulación del llavero múltiple. Si el llavero generador es un llavero KMS, la clave

generadora del AWS KMS genera y cifra la clave en texto no cifrado. Luego todas las AWS KMS keys adicionales del llavero AWS KMS y todas las claves de encapsulación de todos los llaveros secundarios del llavero múltiple cifran la misma clave de texto no cifrado.

Al descifrar, el SDK de cifrado de bases de datos de AWS utiliza los llaveros para intentar descifrar una de las claves de datos cifradas. Los llaveros se llaman en el orden en que están especificados en el llavero múltiple. El procesamiento se detiene tan pronto como cualquier clave de cualquier llavero pueda descifrar una clave de datos cifrada.

Para crear un llavero múltiple, en primer lugar instancie los llaveros secundarios. En este ejemplo, utilizamos un llavero AWS KMS de KMS y un llavero de AES sin formato, pero puede combinar cualquier llavero admitido en un llavero múltiple.

```
// 1. Create 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);

// 2. Create the AWS KMS keyring.
final CreateAwsKmsMrkMultiKeyringInput createAwsKmsMrkMultiKeyringInput =
 CreateAwsKmsMrkMultiKeyringInput.builder()
        .generator(kmsKeyArn)
        .build();
IKeyring awsKmsMrkMultiKeyring =
 matProv.CreateAwsKmsMrkMultiKeyring(createAwsKmsMrkMultiKeyringInput);
```

A continuación, cree el llavero múltiple y especifique su llavero generador, si lo hay. En este ejemplo, creamos un conjunto de claves múltiple en el que el AWS KMS es el llavero generador y el conjunto de claves de AES es el conjunto de claves secundario.

```
final CreateMultiKeyringInput createMultiKeyringInput =
 CreateMultiKeyringInput.builder()
        .generator(awsKmsMrkMultiKeyring)
```

```
        .childKeyrings(Collections.singletonList(rawAesKeyring))
        .build();
IKeyring multiKeyring = matProv.CreateMultiKeyring(createMultiKeyringInput);
```

Ahora, puede utilizar el llavero múltiple para cifrar y descifrar datos.

Cifrado para búsquedas

Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

El cifrado para búsquedas le permite buscar registros cifrados sin tener que descifrar toda la base de datos. Esto se logra mediante balizas, que crean un mapa entre el valor de texto no cifrado escrito en un campo y el valor cifrado que realmente está almacenado en la base de datos. El SDK de cifrado de bases de datos de AWS almacena la baliza en un campo nuevo que se agrega al registro. Según el tipo de baliza que utilice, puede realizar búsquedas de coincidencias exactas o consultas complejas más personalizadas en sus datos cifrados.

> **Note**
>
> El cifrado para búsquedas del SDK de cifrado de bases de datos de AWS difiere del cifrado simétrico para búsquedas definido en la investigación académica, como el cifrado simétrico para búsquedas.

Una baliza es una etiqueta de código de autenticación de mensajes basado en hash (HMAC) truncada que crea un mapa entre el texto no cifrado y los valores cifrados de un campo. Al escribir un valor nuevo en un campo cifrado que está configurado para el cifrado para búsquedas, el SDK de cifrado de bases de datos de AWS calcula un HMAC sobre el valor del texto no cifrado. Esta salida del HMAC coincide uno a uno (1:1) con el valor de texto no cifrado de ese campo. La salida del HMAC se trunca para que varios valores de texto no cifrado distintos se asignen a la misma etiqueta HMAC truncada. Estos falsos positivos limitan la capacidad de un usuario no autorizado para identificar información distintiva sobre el valor del texto no cifrado. Al consultar una baliza, el SDK de cifrado de bases de datos de AWS filtra automáticamente estos falsos positivos y devuelve el resultado de la consulta en texto no cifrado.

El número medio de falsos positivos generados por cada baliza viene determinado por la longitud de la baliza restante tras el truncamiento. Si necesita ayuda para determinar la longitud de la baliza adecuada para su implementación, consulte Determinar la longitud de la baliza.

> **Note**
>
> El cifrado para búsquedas está diseñado para implementarse en bases de datos nuevas y despobladas. Cualquier baliza configurada en una base de datos existente solo mapeará los nuevos registros cargados en la base de datos; no hay forma de que una baliza mapee los datos ya existentes.

Temas

¿Las balizas son adecuadas para mi conjunto de datos?

El uso de balizas para realizar consultas sobre datos cifrados reduce los costos de rendimiento asociados a las bases de datos cifradas del cliente. Cuando se utilizan balizas, existe un equilibrio inherente entre la eficacia de las consultas y la cantidad de información que se revela sobre la distribución de los datos. La baliza no altera el estado cifrado del campo. Al cifrar y firmar un campo con el SDK de cifrado de bases de datos de AWS, el valor de texto no cifrado del campo nunca se expone a la base de datos. La base de datos almacena el valor cifrado y la asignación al azar del campo.

Las balizas se almacenan junto a los campos cifrados a partir de los cuales se calculan. Esto significa que, incluso si un usuario no autorizado no puede ver los valores de texto no cifrado de un campo cifrado, podría realizar un análisis estadístico de las balizas para obtener más información sobre la distribución del conjunto de datos y, en casos extremos, identificar los valores de texto no cifrado a los que se asigna una baliza. La forma en que configura su baliza puede mitigar estos riesgos. En particular, elegir la longitud de baliza correcta puede ayudarle a preservar la confidencialidad de su conjunto de datos.

Seguridad en comparación con rendimiento

- Cuanto menor sea la longitud de la baliza, más seguridad se preservará.
- Cuanto mayor sea la longitud de la baliza, más rendimiento se preservará.

Es posible que el cifrado para búsquedas no proporcione los niveles deseados de rendimiento y seguridad para todos los conjunto de datos. Revise su modelo de amenazas, sus requisitos de seguridad y sus necesidades de rendimiento antes de configurar cualquier baliza.

Tenga en cuenta los siguientes requisitos de exclusividad del conjunto de datos al determinar si el cifrado para búsquedas es adecuado para su conjunto de datos.

Distribución

El grado de seguridad que conserva una baliza depende de la distribución del conjunto de datos. Al configurar un campo cifrado para un cifrado que permita realizar búsquedas, el SDK de cifrado de bases de datos de AWS calcula un HMAC a partir de los valores de texto no cifrado escritos en ese campo. Todas las balizas calculadas para un campo determinado se calculan con la misma clave, con la excepción de las bases de datos de multitenencia, que utilizan una clave distinta para cada inquilino. Esto significa que si se escribe el mismo valor de texto no cifrado en el campo varias veces, se crea la misma etiqueta HMAC para cada instancia de ese valor de texto no cifrado.

Debe evitar crear balizas a partir de campos que contengan valores muy comunes. Por ejemplo, considere una base de datos que almacene la dirección de todos los residentes del estado de Illinois. Si crea una baliza a partir del campo `City` cifrado, la baliza calculada sobre «Chicago» estará sobrerrepresentada debido al porcentaje alto de la población de Illinois que vive en Chicago. Incluso si un usuario no autorizado solo puede leer los valores cifrados y los valores de la baliza, podría identificar qué registros contienen datos de los residentes de Chicago si la baliza conserva esta distribución. Para minimizar la cantidad de información distintiva revelada sobre su distribución, debe truncar suficientemente la baliza. La longitud de baliza necesaria para ocultar esta distribución desigual supone unos costos de rendimiento significativos que podrían no satisfacer las necesidades de la aplicación.

Debe analizar detenidamente la distribución del conjunto de datos para determinar en qué medida es necesario truncar las balizas. La longitud de la baliza que queda después del truncamiento se correlaciona directamente con la cantidad de información estadística que se puede identificar sobre su distribución. Es posible que tengas que elegir longitudes de baliza más cortas para minimizar suficientemente la cantidad de información distintiva que se revela sobre tu conjunto de datos.

En casos extremos, no se puede calcular la longitud de una baliza para un conjunto de datos distribuido de forma desigual que equilibre eficazmente el rendimiento y la seguridad. Por ejemplo, no debe construir una baliza a partir de un campo que almacene el resultado de

un examen médico para detectar una enfermedad rara. Como `NEGATIVE` se espera que los resultados sean significativamente más frecuentes en el conjunto de datos, `POSITIVE` los resultados se pueden identificar fácilmente por su poca frecuencia. Es muy difícil ocultar la distribución cuando el campo solo tiene dos valores posibles. Si utiliza una longitud de baliza lo suficientemente corta como para ocultar la distribución, todos los valores de texto no cifrado se asignan a la misma etiqueta HMAC. Si utiliza una longitud de baliza más larga, es obvio qué balizas se asignan a valores `POSITIVE` de texto no cifrado.

Correlación

Le recomendamos encarecidamente que evite construir balizas distintas a partir de campos con valores relacionados entre sí. Las balizas construidas a partir de campos relacionados entre sí requieren longitudes de baliza más cortas para minimizar suficientemente la cantidad de información revelada sobre la distribución de cada conjunto de datos a un usuario no autorizado. Debe analizar detenidamente el conjunto de datos, incluida su entropía y la distribución conjunta de los valores relacionados entre sí, para determinar en qué medida deben truncarse las balizas. Si la longitud de baliza resultante no satisface sus necesidades de rendimiento, es posible que las balizas no sean adecuadas para su conjunto de datos.

Por ejemplo, no debe construir dos balizas independientes a partir de los campos `City` y `ZIPCode`, ya que es probable que el código postal esté asociado a una sola ciudad. Por lo general, los falsos positivos que genera una baliza limitan la capacidad de un usuario no autorizado de identificar información distintiva sobre su conjunto de datos. Sin embargo, la correlación entre los campos `City` y `ZIPCode` significa que un usuario no autorizado puede identificar fácilmente qué resultados son falsos positivos y distinguir los distintos códigos postales.

También debe evitar crear balizas a partir de campos que contengan los mismos valores de texto no cifrado. Por ejemplo, no debe crear una baliza a partir de los campos `mobilePhone` y `preferredPhone` porque es probable que contengan los mismos valores. Si crea balizas distintas a partir de ambos campos, el SDK de cifrado de bases de datos de AWS crea las balizas para cada campo con claves diferentes. Esto da como resultado dos etiquetas HMAC diferentes para el mismo valor de texto no cifrado. Es poco probable que las dos balizas distintas tengan los mismos falsos positivos y un usuario no autorizado podría distinguir números de teléfono diferentes.

Incluso si su conjunto de datos contiene campos relacionados entre sí o tiene una distribución desigual, es posible que pueda construir balizas que preserven la confidencialidad del conjunto de datos mediante longitudes de baliza más cortas. Sin embargo, la longitud de la baliza no garantiza

que cada valor único del conjunto de datos produzca una serie de falsos positivos que minimicen de forma efectiva la cantidad de información distintiva que se revela sobre el conjunto de datos. La longitud de la baliza solo estima el número medio de falsos positivos producidos. Cuanto más desigualmente esté distribuido el conjunto de datos, menos eficaz será la longitud de la baliza para determinar el número medio de falsos positivos producidos.

Considere detenidamente la distribución de los campos a partir de los cuales construye las balizas y considere cuánto necesitará truncar la longitud de la baliza para cumplir con sus requisitos de seguridad. En los siguientes temas de este capítulo, se parte del supuesto de que las balizas están distribuidas uniformemente y no contienen datos relacionados entre sí.

Situación de cifrado para búsquedas

En el ejemplo siguiente, se muestra una solución de cifrado sencilla para búsquedas. En la aplicación, es posible que los campos de ejemplo utilizados en este ejemplo no cumplan con las recomendaciones de unicidad de distribución y correlación para las balizas. Puede utilizar este ejemplo como referencia mientras lee en este capítulo acerca de los conceptos de cifrado para búsquedas.

Considere una base de datos denominada `Employees` que rastrea los datos de los empleados de una empresa. Cada registro de la base de datos contiene campos denominados EmployeeID, LastName, FirstName y Address. Cada campo de la `Employees` base de datos se identifica mediante la clave principal `EmployeeID`.

A continuación, se muestra un ejemplo de un registro de texto no cifrado de la base de datos.

```
{
    "EmployeeID": 101,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
                "Street": "123 Main",
                "City": "Anytown",
                "State": "OH",
                "ZIPCode": 12345
    }
}
```

Si marcó los campos `LastName` y `FirstName` como `ENCRYPT_AND_SIGN` en sus acciones criptográficas, los valores de estos campos se cifrarán localmente antes de cargarlos en la base de

datos. Los datos cifrados que se cargan son completamente asignados al azar y la base de datos no los reconoce como protegidos. Simplemente detecta las entradas de datos típicas. Esto significa que el registro que está realmente almacenado en la base de datos podría tener el siguiente aspecto.

```
{
    "PersonID": 101,
    "LastName": "1d76e94a2063578637d51371b363c9682bad926cbd",
    "FirstName": "21d6d54b0aaabc411e9f9b34b6d53aa4ef3b0a35",
    "Address": {
                "Street": "123 Main",
                "City": "Anytown",
                "State": "OH",
                "ZIPCode": 12345
    }
}
```

Si necesita consultar en la base de datos las coincidencias exactas en el `LastName` campo, configure un indicador estándar denominado LastName para asignar los valores de texto no cifrado escritos en el `LastName` campo a los valores cifrados almacenados en la base de datos.

Esta baliza calcula los HMAC a partir de los valores de texto no cifrado del campo `LastName`. Cada salida del HMAC se trunca para que ya no coincida exactamente con el valor del texto no cifrado. Por ejemplo, el hash completo y el hash truncado `Jones` pueden tener el siguiente aspecto.

Hash completo

`2aa4e9b404c68182562b6ec761fcca5306de527826a69468885e59dc36d0c3f824bdd44cab45526f`

Hash truncado

`b35099d408c833`

Una vez configurada la baliza estándar, puede realizar búsquedas de igualdad en el campo `LastName`. Por ejemplo, si desea buscar a `Jones`, utilice la baliza LastName para realizar la siguiente consulta.

```
LastName = Jones
```

El SDK de cifrado de bases de datos de AWS filtra automáticamente los falsos positivos y devuelve el resultado de la consulta en texto no cifrado.

Balizas

> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

Una baliza es una etiqueta de código de autenticación de mensajes basado en hash (HMAC) truncada que crea un mapa entre el valor de texto no cifrado escrito en un campo y el valor cifrado que está realmente almacenado en la base de datos. La baliza no altera el estado cifrado del campo. La baliza calcula un HMAC sobre el valor de texto no cifrado del campo y lo almacena junto con el valor cifrado. Esta salida del HMAC coincide uno a uno (1:1) con el valor de texto no cifrado de ese campo. La salida del HMAC se trunca para que varios valores de texto no cifrado distintos se asignen a la misma etiqueta HMAC truncada. Estos falsos positivos limitan la capacidad de un usuario no autorizado para identificar información distintiva sobre el valor del texto no cifrado.

Las balizas solo se pueden crear a partir de campos marcados `ENCRYPT_AND_SIGN` o `SIGN_ONLY` en sus acciones criptográficas. La baliza en sí no está firmada ni cifrada. No se puede construir una baliza con campos marcados `DO_NOTHING`.

El tipo de baliza que configure determinará el tipo de consultas que podrá realizar. Existen dos tipos de balizas que admiten el cifrado para búsquedas. Las balizas estándar realizan búsquedas de igualdad. Las balizas compuestas combinan cadenas literales de texto no cifrado y balizas estándar para realizar operaciones complejas de bases de datos. Después de configurar las balizas, debe configurar un índice secundario para cada baliza antes de poder buscar en los campos cifrados. Para obtener más información, consulte Configurar índices secundarios con balizas.

Temas

Balizas estándar

Las balizas estándar son la forma más sencilla de implementar el cifrado para búsquedas en su base de datos. Solo pueden realizar búsquedas de igualdad para un único campo virtual o cifrado. Para obtener información sobre cómo configurar balizas estándar, consulte Configuración de balizas estándar.

El campo a partir del cual se construye una baliza estándar se denomina la fuente de baliza. Identifica la ubicación de los datos que la baliza necesita mapear. La fuente de la baliza puede ser un campo cifrado o un campo virtual. La fuente de baliza de cada baliza estándar debe ser única. No puede configurar dos balizas con la misma fuente de baliza.

Puede crear una baliza estándar que realice búsquedas de igualdad para un único campo cifrado, o puede crear una baliza estándar que realice búsquedas de igualdad en la concatenación de varios campos `ENCRYPT_AND_SIGN` y `SIGN_ONLY` mediante la creación de un campo virtual.

Campos virtuales

Un campo virtual es un campo conceptual creado a partir de uno o más campos de origen. Al crear un campo virtual no se graba un campo nuevo en el registro. El campo virtual no se almacena de forma explícita en la base de datos. Se utiliza en la configuración de baliza estándar para dar instrucciones a la baliza sobre cómo identificar un segmento específico de un campo o concatenar varios campos de un registro para realizar una consulta específica. Un campo virtual requiere al menos un campo cifrado.

> **Note**
>
> El siguiente ejemplo muestra los tipos de transformaciones y consultas que se pueden realizar con un campo virtual. En la aplicación, es posible que los campos de ejemplo utilizados en este ejemplo no cumplan con las recomendaciones de unicidad de distribución y correlación para las balizas.

Por ejemplo, si desea realizar búsquedas de igualdad en la concatenación de los campos `FirstName` y `LastName`, puede crear uno de los siguientes campos virtuales.

- Un campo virtual `NameTag`, construido a partir de la primera letra del campo `FirstName`, seguida del campo `LastName`, todo en minúsculas. Este campo virtual le permite realizar consultas `NameTag=mjones`.
- Un campo virtual `LastFirst`, que se construye a partir del campo `LastName`, seguido del campo `FirstName`. Este campo virtual le permite realizar consultas `LastFirst=JonesMary`.

O bien, si desea realizar búsquedas de igualdad en un segmento específico de un campo cifrado, cree un campo virtual que identifique el segmento que desea consultar.

Por ejemplo, si desea consultar un campo `IPAddress` cifrado con los tres primeros segmentos de la dirección IP, cree el siguiente campo virtual.

- Un campo virtual `IPSegment`, construido a partir de `Segments('.', 0, 3)`. Este campo virtual le permite realizar consultas `IPSegment=192.0.2`. La consulta devuelve todos los registros con un valor `IPAddress` que comienza por «192.0.2».

Los campos virtuales deben ser únicos. No se pueden construir dos campos virtuales a partir exactamente de los mismos campos de origen.

Para obtener ayuda para configurar los campos virtuales y las balizas que los utilizan, consulte Creación de un campo virtual.

Balizas compuestas

Las balizas compuestas crean índices que mejoran el rendimiento de las consultas y permiten realizar operaciones de base de datos más complejas. Puede utilizar balizas compuestas para combinar cadenas literales de texto no cifrado y balizas estándar para realizar consultas complejas en registros cifrados, como consultar dos tipos de registros diferentes desde un único índice o consultar una combinación de campos con una clave de clasificación. Para ver más ejemplos de soluciones de baliza compuesta, consulte Elegir un tipo de baliza.

Las balizas compuestas se pueden construir a partir de balizas estándar o de una combinación de balizas y campos estándar `SIGN_ONLY`. Se construyen a partir de una lista de piezas. Todas las balizas compuestas deben incluir una lista de partes cifradas que identifique los `ENCRYPT_AND_SIGN` campos incluidos en la baliza. Cada `ENCRYPT_AND_SIGN` campo debe identificarse mediante una baliza estándar. Las balizas compuestas más complejas también pueden incluir una lista de partes firmadas que identifique los campos `SIGN_ONLY` de texto no cifrado incluidos en la baliza y una lista de partes constructivas que identifique todas las formas posibles en que la baliza compuesta puede ensamblar los campos.

> **Note**
>
> El SDK de cifrado de bases de datos de AWS también admite balizas firmadas que se pueden configurar completamente a partir de campos `SIGN_ONLY` de texto no cifrado. Las

balizas firmadas son un tipo de baliza compuesta que indexan y realizan consultas complejas en los campos. `SIGN_ONLY` Para obtener más información, consulte Crear balizas firmadas.

Para obtener ayuda para configurar balizas compuestas, consulte Configuración de balizas compuestas.

La forma en que configure su baliza compuesta determina los tipos de consultas que puede realizar. Por ejemplo, puede hacer que algunas partes cifradas y firmadas sean opcionales para permitir una mayor flexibilidad en sus consultas. Para obtener más información sobre los tipos de consultas que pueden realizar las balizas compuestas, consulte Balizas de consulta.

Planificación de balizas

Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

Las balizas están diseñadas para su implementación en bases de datos nuevas y despobladas. Cualquier baliza configurada en una base de datos existente solo mapeará los nuevos registros escritos en la base de datos. Las balizas se calculan a partir del valor de texto no cifrado de un campo. Una vez cifrado el campo, la baliza no tiene forma de mapear los datos existentes. Una vez que haya escrito nuevos registros con una baliza, no puede actualizar la configuración de la baliza. Sin embargo, puede agregar balizas nuevas para los campos nuevos que agregue a su registro.

Para implementar un cifrado que permita realizar búsquedas, debe usar el AWS KMS llavero jerárquico para generar, cifrar y descifrar las claves de datos que se utilizan para proteger sus registros. Para obtener más información, consulte Uso del conjunto de claves jerárquico para el cifrado para búsquedas.

Antes de poder configurar balizas para el cifrado para búsquedas, debe revisar sus requisitos de cifrado, los patrones de acceso a las bases de datos y el modelo de amenazas para determinar cuál es la mejor solución para su base de datos.

El tipo de baliza que configure determina el tipo de consultas que puede realizar. La longitud de la baliza que especifique en la configuración de baliza estándar determina el número esperado

de falsos positivos producidos para una baliza determinada. Recomendamos encarecidamente identificar y planificar los tipos de consultas que debe realizar antes de configurar las balizas. Una vez que haya utilizado una baliza, la configuración no se puede actualizar.

Le recomendamos encarecidamente que revise y complete las siguientes tareas antes de configurar cualquier baliza.

- Determine si las balizas son adecuadas para su conjunto de datos
- Elija un tipo de baliza
- Elija una longitud de baliza
- Elija un nombre de baliza

Recuerde los siguientes requisitos de exclusividad de las balizas al planificar la solución de cifrado para búsquedas para su base de datos.

- Cada baliza estándar debe tener una fuente de baliza única

  No se pueden construir varias balizas estándar a partir del mismo campo virtual o cifrado.

  Sin embargo, se puede usar una única baliza estándar para construir múltiples balizas compuestas.
- Evite crear un campo virtual con campos de origen que se superpongan con las balizas estándar existentes

  La construcción de una baliza estándar a partir de un campo virtual que contiene un campo de origen que se utilizó para crear otra baliza estándar puede reducir la seguridad de ambas balizas.

  Para obtener más información, consulte Aspectos de seguridad para campos virtuales.

Consideraciones para bases de datos de multitenencia

Para consultar las balizas configuradas en una base de datos de multitenencia, debe incluir el campo que almacena la `branch-key-id` asociada al inquilino que cifró el registro en la consulta. Este campo se define al definir la fuente de la clave de la baliza. Para que la consulta se realice correctamente, el valor de este campo debe identificar los materiales clave de baliza adecuados necesarios para volver a calcular la baliza.

Antes de configurar las balizas, debe decidir cómo las va a incluir `branch-key-id` en las consultas. Para obtener más información sobre las diferentes formas en que puede incluirlos `branch-key-id` en sus consultas, visite Consulta de balizas en una base de datos de multitenencia.

Elección de un tipo de baliza

> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

Con el cifrado que permite realizar búsquedas, puede buscar registros cifrados asignando con una baliza los valores de texto no cifrado de un campo cifrado. El tipo de baliza que configure determina el tipo de consultas que puede realizar.

Recomendamos encarecidamente identificar y planificar los tipos de consultas que debe realizar antes de configurar las balizas. Después de configurar las balizas, debe configurar un índice secundario para cada baliza antes de poder buscar en los campos cifrados. Para obtener más información, consulte Configurar índices secundarios con balizas.

Las balizas crean un mapa entre el valor de texto no cifrado escrito en un campo y el valor cifrado que está realmente almacenado en la base de datos. No se pueden comparar los valores de dos balizas estándar, aunque contengan el mismo texto no cifrado subyacente. Las dos balizas estándar generarán dos etiquetas HMAC diferentes para los mismos valores de texto no cifrado. Como resultado, las balizas estándar no pueden realizar las siguientes consultas.

- *beacon1* = *beacon2*
- *beacon1* IN (*beacon2*)
- *value* IN (*beacon1*, *beacon2*, ...)
- CONTAINS(*beacon1*, *beacon2*)

Solo puede realizar las consultas anteriores si compara las partes firmadas de las balizas compuestas, con la excepción del `CONTAINS` operador, que puede utilizar con las balizas compuestas para identificar el valor total de un campo cifrado o firmado que contenga la baliza ensamblada. Al comparar partes firmadas, si lo desea, puede incluir el prefijo de una parte cifrada, pero no puede incluir el valor cifrado de un campo. Para obtener más información sobre los tipos de consultas que pueden realizar las balizas estándar y compuestas, consulte Consulta de balizas.

Tenga en cuenta las siguientes soluciones de cifrado con capacidad de búsqueda al revisar los patrones de acceso a la base de datos. Los siguientes ejemplos definen qué baliza se debe configurar para satisfacer los diferentes requisitos de cifrado y consulta.

Balizas estándar

Las balizas estándar solo pueden realizar búsquedas de igualdad. Puede utilizar balizas estándar para llevar a cabo las siguientes consultas.

Consulte un único campo cifrado

Si desea identificar los registros que contienen un valor específico para un campo cifrado, cree un indicador estándar.

Ejemplos

Para el siguiente ejemplo, considere una base de datos denominada `UnitInspection` que rastrea los datos de inspección de una planta de producción. Cada registro de la base de datos contiene campos denominados `work_id`, `inspection_date`, `inspector_id_last4` y `unit`. El ID completo del inspector es un número comprendido entre 0 y 99.999.999. Sin embargo, para garantizar que el conjunto de datos se distribuya de manera uniforme, `inspector_id_last4` solo almacena los últimos cuatro dígitos del ID del inspector. Cada campo de la base de datos se identifica mediante la clave principal`work_id`. Los campos `inspector_id_last4` y `unit` están marcados `ENCRYPT_AND_SIGN` en las acciones criptográficas.

A continuación, se muestra un ejemplo de una entrada de texto no cifrado en la `UnitInspection` base de datos.

```
{
    "work_id": "1c7fcff3-6e74-41a8-b7f7-925dc039830b",
    "inspection_date": 2023-06-07,
    "inspector_id_last4": 8744,
    "unit": 229304973450
}
```

Consulte un único campo cifrado de un registro

Si es necesario cifrar el campo `inspector_id_last4`, pero aun así necesita consultarlo para obtener coincidencias exactas, cree una baliza estándar a partir del campo `inspector_id_last4`. A continuación, utilice la baliza estándar para crear un

índice secundario. Puede utilizar este índice secundario para realizar consultas en el `inspector_id_last4` campo cifrado.

Para obtener ayuda para configurar balizas estándar, consulte Configuración de balizas estándar.

Consulte un campo virtual

Un campo virtual es un campo conceptual creado a partir de uno o más campos de origen. Si desea realizar búsquedas de igualdad para un segmento específico de un campo cifrado o realizar búsquedas de igualdad en la concatenación de varios campos, cree una baliza estándar a partir de un campo virtual. Todos los campos virtuales deben incluir al menos un campo de origen cifrado.

Ejemplos

Los siguientes ejemplos crean campos virtuales para la `Employees` base de datos. A continuación, se muestra un ejemplo de un registro de texto no cifrado de la `Employees` base de datos.

```
{
    "EmployeeID": 101,
    "SSN": 000-00-0000,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
                "Street": "123 Main",
                "City": "Anytown",
                "State": "OH",
                "ZIPCode": 12345
    }
}
```

Consulte un segmento de un campo cifrado

En este ejemplo, el campo `SSN` está cifrado.

Si desea consultar el campo `SSN` con los últimos cuatro dígitos de un número de seguro social, cree un campo virtual que identifique el segmento que desea consultar.

Un `Last4SSN` campo virtual, creado a partir de `Suffix(4)`, le permite realizar consultas`Last4SSN=0000`. Utilice este campo virtual para construir una baliza estándar. A continuación, utilice la baliza estándar para crear un índice secundario. Puede utilizar este índice

secundario para realizar consultas en el campo virtual. Esta consulta devuelve todos los registros con un valor SSN que termina en los últimos cuatro dígitos que especificó.

Consulte la concatenación de varios campos

> **Note**
>
> El siguiente ejemplo muestra los tipos de transformaciones y consultas que se pueden realizar con un campo virtual. En la aplicación, es posible que los campos de ejemplo utilizados en este ejemplo no cumplan con las recomendaciones de unicidad de distribución y correlación para las balizas.

Si desea realizar búsquedas de igualdad en una concatenación de los campos `FirstName` y `LastName`, puede crear un campo `NameTag` virtual, construido a partir de la primera letra del campo `FirstName`, seguida del campo `LastName`, todo en minúsculas. Utilice este campo virtual para construir una baliza estándar. A continuación, utilice la baliza estándar para crear un índice secundario. Puede utilizar este índice secundario para realizar consultas `NameTag=mjones` en el campo virtual.

Al menos uno de los campos de origen debe estar cifrado. Bien sea `FirstName` o `LastName` puede estar cifrado, o ambos pueden estar cifrados. Todos los campos de origen de texto no cifrado deben marcarse como `SIGN_ONLY` en sus acciones criptográficas.

Para obtener ayuda para configurar los campos virtuales y las balizas que los utilizan, consulte Creación de un campo virtual.

Balizas compuestas

Las balizas compuestas crean un índice a partir de cadenas literales de texto no cifrado y balizas estándar para realizar operaciones complejas de bases de datos. Puede utilizar balizas compuestas para realizar las siguientes consultas.

Consulte una combinación de campos cifrados en un único índice

Si necesita consultar una combinación de campos cifrados en un único índice, cree una baliza compuesta que combine las balizas estándar individuales creadas para cada campo cifrado para formar un índice único.

Tras configurar la baliza compuesta, puede crear un índice secundario que especifique la baliza compuesta como clave de partición para realizar consultas de coincidencia exacta o con una clave de clasificación para realizar consultas más complejas. Los índices secundarios que especifican la baliza compuesta como clave de clasificación pueden realizar consultas de coincidencia exacta y consultas complejas más personalizadas.

Ejemplos

Para los siguientes ejemplos, considere una base de datos denominada `UnitInspection` que rastrea los datos de inspección de una instalación de producción. Cada registro de la base de datos contiene campos denominados `work_id`, `inspection_date`, `inspector_id_last4` y `unit`. El ID completo del inspector es un número comprendido entre 0 y 99.999.999. Sin embargo, para garantizar que el conjunto de datos se distribuya de manera uniforme, `inspector_id_last4` solo almacena los últimos cuatro dígitos del ID del inspector. Cada campo de la base de datos se identifica mediante la clave principal`work_id`. Los campos `inspector_id_last4` y `unit` están marcados `ENCRYPT_AND_SIGN` en las acciones criptográficas.

A continuación, se muestra un ejemplo de una entrada de texto no cifrado en la `UnitInspection` base de datos.

```
{
    "work_id": "1c7fcff3-6e74-41a8-b7f7-925dc039830b",
    "inspection_date": 2023-06-07,
    "inspector_id_last4": 8744,
    "unit": 229304973450
}
```

Realice búsquedas de igualdad en una combinación de campos cifrados

Si desea consultar en la `UnitInspection` base de datos las coincidencias exactas en `inspector_id_last4.unit`, cree primero balizas estándar distintas para los campos `inspector_id_last4` y `unit`. A continuación, cree una baliza compuesta a partir de las dos balizas estándar.

Después de configurar la baliza compuesta, cree un índice secundario que especifique la baliza compuesta como clave de partición. Utilice este índice secundario para buscar coincidencias exactas en `inspector_id_last4.unit`. Por ejemplo, puede consultar esta baliza para encontrar una lista de las inspecciones que un inspector realizó en una unidad determinada.

Realice consultas complejas en una combinación de campos cifrados

Si desea consultar la `UnitInspection` base de datos en `inspector_id_last4` y `inspector_id_last4.unit`, primero, cree balizas estándar distintas para los campos `inspector_id_last4` y `unit`. A continuación, cree una baliza compuesta a partir de las dos balizas estándar.

Después de configurar la baliza compuesta, cree un índice secundario que especifique la baliza compuesta como clave de clasificación. Utilice este índice secundario para consultar en la `UnitInspection` base de datos las entradas que comiencen por un inspector determinado o consulte la base de datos para obtener una lista de todas las unidades dentro de un rango de ID de unidades específico que fueron inspeccionadas por un inspector determinado. También puede realizar búsquedas de coincidencias exactas en `inspector_id_last4.unit`.

Para obtener ayuda para configurar balizas compuestas, consulte Configuración de balizas compuestas.

Consulte una combinación de campos cifrados y de texto no cifrado en un único índice

Si necesita consultar una combinación de campos cifrados y de texto no cifrado en un solo índice, cree un indicador compuesto que combine balizas estándar individuales y campos de texto no cifrado para formar un índice único. Los campos de texto no cifrado que se utilizan para construir la baliza compuesta deben estar marcados `SIGN_ONLY` en sus acciones criptográficas.

Después de configurar la baliza compuesta, puede crear un índice secundario que especifique la baliza compuesta como clave de partición para realizar consultas de coincidencia exacta o con una clave de clasificación para realizar consultas más complejas. Los índices secundarios que especifican la baliza compuesta como clave de clasificación pueden realizar consultas de coincidencia exacta y consultas complejas más personalizadas.

Ejemplos

Para los siguientes ejemplos, considere una base de datos denominada `UnitInspection` que rastrea los datos de inspección de una instalación de producción. Cada registro de la base de datos contiene campos denominados `work_id`, `inspection_date`, `inspector_id_last4` y `unit`. El ID completo del inspector es un número comprendido entre 0 y 99.999.999. Sin embargo, para garantizar que el conjunto de datos se distribuya de manera uniforme, `inspector_id_last4` solo almacena los últimos cuatro dígitos del ID del inspector. Cada campo de la base de datos se

identifica mediante la clave principalwork_id. Los campos inspector_id_last4 y unit están marcados ENCRYPT_AND_SIGN en las acciones criptográficas.

A continuación, se muestra un ejemplo de una entrada de texto no cifrado en la UnitInspection base de datos.

```
{
    "work_id": "1c7fcff3-6e74-41a8-b7f7-925dc039830b",
    "inspection_date": 2023-06-07,
    "inspector_id_last4": 8744,
    "unit": 229304973450
}
```

Realice búsquedas de igualdad en una combinación de campos

Si desea consultar en la base de datos de UnitInspection las inspecciones realizadas por un inspector específico en una fecha específica, cree primero una baliza estándar para el campo inspector_id_last4. El campo inspector_id_last4 está marcado ENCRYPT_AND_SIGN en las acciones criptográficas. Todas las partes cifradas requieren su propia baliza estándar. El campo inspection_date está marcado SIGN_ONLY y no requiere una baliza estándar. A continuación, cree una baliza compuesta desde el campoinspection_date y la baliza inspector_id_last4 estándar.

Después de configurar la baliza compuesta, cree un índice secundario que especifique la baliza compuesta como clave de partición. Utilice este índice secundario para consultar en las bases de datos los registros que coincidan exactamente con un inspector y una fecha de inspección determinados. Por ejemplo, puede consultar la base de datos para obtener una lista de todas las inspecciones que el inspector cuya ID termina en 8744 realizó en una fecha específica.

Realice consultas complejas en una combinación de campos

Si desea consultar en la base de datos las inspecciones realizadas dentro de un intervalo de inspection_date, o consultar en la base de datos las inspecciones realizadas en un determinado ámbito inspection_date limitado por inspector_id_last4 o inspector_id_last4.unit, primero, cree balizas estándar distintas para los campos inspector_id_last4 y unit. A continuación, cree una baliza compuesta a partir del campo inspection_date de texto no cifrado y de las dos balizas estándar.

Después de configurar la baliza compuesta, cree un índice secundario que especifique la baliza compuesta como clave de clasificación. Utilice este índice secundario para realizar consultas

sobre las inspecciones realizadas en fechas específicas por un inspector específico. Por ejemplo, puede consultar la base de datos para obtener una lista de todas las unidades inspeccionadas en la misma fecha. O bien, puede consultar la base de datos para obtener una lista de todas las inspecciones realizadas en una unidad específica en un intervalo determinado de fechas de inspección.

Para obtener ayuda para configurar balizas compuestas, consulte Configuración de balizas compuestas.

Elegir la longitud de una baliza

> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

Al escribir un valor nuevo en un campo cifrado que está configurado para el cifrado para búsquedas, el SDK de cifrado de bases de datos de AWS calcula un HMAC sobre el valor del texto no cifrado. Esta salida del HMAC coincide uno a uno (1:1) con el valor de texto no cifrado de ese campo. La salida del HMAC se trunca para que varios valores de texto no cifrado distintos se asignen a la misma etiqueta HMAC truncada. Estas colisiones, o falsos positivos, limitan la capacidad de un usuario no autorizado de identificar información distintiva sobre el valor del texto no cifrado.

El número medio de falsos positivos generados por cada baliza viene determinado por la longitud de la baliza restante tras el truncamiento. Solo es necesario definir la longitud de la baliza al configurar las balizas estándar. Las balizas compuestas utilizan las longitudes de baliza de las balizas estándar con las que están construidas.

La baliza no altera el estado cifrado del campo. Sin embargo, cuando se utilizan balizas, existe un equilibrio inherente entre la eficacia de las consultas y la cantidad de información que se revela sobre la distribución de los datos.

El objetivo del cifrado con capacidad de búsqueda es reducir los costos de rendimiento asociados a las bases de datos cifradas del cliente mediante el uso de balizas para realizar consultas sobre datos cifrados. Las balizas se almacenan junto a los campos cifrados a partir de los cuales se calculan. Esto significa que pueden revelar información distintiva sobre la distribución de su conjunto de datos. En casos extremos, un usuario no autorizado podría analizar la información revelada sobre su

distribución y utilizarla para identificar el valor de texto no cifrado de un campo. Elegir la longitud de baliza correcta puede ayudar a mitigar estos riesgos y preservar la confidencialidad de la distribución.

Revise su modelo de amenazas para determinar el nivel de seguridad que necesita. Por ejemplo, cuantas más personas tengan acceso a su base de datos, pero que no deberían tener acceso a los datos en texto no cifrado, más querrá proteger la confidencialidad de la distribución de su conjunto de datos. Para aumentar la confidencialidad, una baliza debe generar más falsos positivos. El aumento de la confidencialidad reduce el rendimiento de las consultas.

Seguridad en comparación con rendimiento

- Una longitud de baliza demasiado larga produce muy pocos falsos positivos y podría revelar información distintiva sobre la distribución del conjunto de datos.
- Una longitud de baliza demasiado corta produce demasiados falsos positivos y aumenta el costo de rendimiento de las consultas, porque requiere un análisis más amplio de la base de datos.

Al determinar la longitud de baliza adecuada para su solución, debe encontrar una longitud que preserve adecuadamente la seguridad de sus datos sin afectar el rendimiento de las consultas más de lo estrictamente necesario. El grado de seguridad que conserva una baliza depende de la distribución del conjunto de datos y de la correlación de los campos a partir de los cuales se construyen las balizas. En los temas siguientes, se parte del supuesto de que las balizas están distribuidas de manera uniforme y no contienen datos relacionados entre sí.

Temas

Calcular la longitud de la baliza

La longitud de la baliza se define en bits y se refiere al número de bits de la etiqueta HMAC que se conservan tras el truncamiento. La longitud de baliza recomendada varía en función de la distribución del conjunto de datos, la presencia de valores relacionados entre sí y los requisitos específicos de seguridad y rendimiento. Si su conjunto de datos está distribuido de manera uniforme, puede usar las siguientes ecuaciones y procedimientos para ayudarse a identificar la mejor longitud de baliza para su implementación. Estas ecuaciones solo estiman el número promedio de falsos positivos que producirá la baliza, pero no garantizan que cada valor único del conjunto de datos produzca un número específico de falsos positivos.

> **Note**
>
> La eficacia de estas ecuaciones depende de la distribución del conjunto de datos. Si su conjunto de datos no está distribuido uniformemente, consulte ¿Las balizas son adecuadas para mi conjunto de datos?.
> En general, cuanto más lejos esté el conjunto de datos de una distribución uniforme, más necesitará acortar la longitud de la baliza.

1.

Calcula la población

La población es el número esperado de valores únicos en el campo a partir del cual se construye la baliza estándar, no el número total esperado de valores almacenados en el campo. Por ejemplo, considere un `Room` campo cifrado que identifique la ubicación de las reuniones de los empleados. Se espera que el `Room` campo almacene 100 000 valores en total, pero solo hay 50 salas diferentes que los empleados pueden reservar para reuniones. Esto significa que la población es de 50 porque solo hay 50 valores únicos posibles que se pueden almacenar en el `Room` campo.

> **Note**
>
> Si la baliza estándar se construye a partir de un campo virtual, la población utilizada para calcular la longitud de la baliza es el número de combinaciones únicas creadas por el campo virtual.

Al estimar la población, asegúrese de tener en cuenta el crecimiento proyectado del conjunto de datos. Una vez que haya escrito nuevos registros con la baliza, no podrá actualizar la longitud de la baliza. Revise su modelo de amenazas y cualquier solución de base de datos existente para crear una estimación del número de valores únicos que espera que este campo almacene en los próximos cinco años.

Su población no tiene por qué ser precisa. En primer lugar, identifique el número de valores únicos en su base de datos actual o calcule el número de valores únicos que espera almacenar durante el primer año. A continuación, utilice las siguientes preguntas para determinar el crecimiento proyectado de los valores únicos en los próximos cinco años.

- ¿Espera que los valores únicos se multipliquen por 10?
- ¿Espera que los valores únicos se multipliquen por 100?
- ¿Espera que los valores únicos se multipliquen por 1000?

La diferencia entre los valores únicos 50 000 y 60 000 no es significativa y ambos darán como resultado la misma longitud de baliza recomendada. Sin embargo, la diferencia entre los valores únicos 50 000 y 500 000 afectará considerablemente la longitud de baliza recomendada.

Considere la posibilidad de revisar los datos públicos sobre la frecuencia de los tipos de datos más comunes, como los códigos postales o los apellidos. Por ejemplo, hay 41 707 códigos postales en los Estados Unidos. La población que utilice debe ser proporcional a su propia base de datos. Si el `ZIPCode` campo de la base de datos incluye datos de todos los Estados Unidos, puede definir su población como 41 707, incluso si el `ZIPCode` campo no tiene actualmente 41 707 valores únicos. Si el `ZIPCode` campo de la base de datos solo incluye datos de un estado y siempre incluirá datos de un solo estado, entonces puede definir su población como el número total de códigos postales de ese estado en lugar de 41 704.

2. Calcule el rango recomendado para el número esperado de colisiones

   Para determinar la longitud de baliza adecuada para un campo determinado, primero debe identificar un rango adecuado para el número esperado de colisiones. El número esperado de colisiones representa el número promedio esperado de valores únicos de texto no cifrado que se asignan a una etiqueta HMAC concreta. El número esperado de falsos positivos para un valor único de texto no cifrado es uno menos que el número esperado de colisiones.

   Recomendamos que el número esperado de colisiones sea mayor o igual a dos e inferior a la raíz cuadrada de la población. Las siguientes ecuaciones solo funcionan si la población tiene 16 o más valores únicos.

   ```
   2 ≤ number of collisions < √(Population)
   ```

   Si el número de colisiones es inferior a dos, la baliza producirá muy pocos falsos positivos. Recomendamos dos como número mínimo de colisiones esperadas porque significa que, en promedio, cada valor único del campo generará al menos un falso positivo al asignarlo a otro valor único.

3. Calcule el rango recomendado para las longitudes de baliza

Tras identificar el número mínimo y máximo de colisiones esperadas, utilice la siguiente ecuación para identificar un rango de longitudes de baliza adecuadas.

$$\text{number of collisions} = \text{Population} * 2^{-(\text{beacon length})}$$

En primer lugar, calcule la longitud de la baliza, donde el número de colisiones esperadas es igual a dos (el número mínimo recomendado de colisiones esperadas).

$$2 = \text{Population} * 2^{-(\text{beacon length})}$$

A continuación, calcule la longitud de la baliza, donde el número esperado de colisiones es igual a la raíz cuadrada de tu población (el número máximo recomendado de colisiones esperadas).

$$\sqrt{(\text{Population})} = \text{Population} * 2^{-(\text{beacon length})}$$

Recomendamos redondear el resultado que produce esta ecuación a la longitud de baliza más corta. Por ejemplo, si la ecuación produce una longitud de baliza de 15,6, recomendamos redondear ese valor a 15 bits en lugar de redondearlo al alza a 16 bits.

4. Elija una longitud de baliza

   Estas ecuaciones solo identifican un rango recomendado de longitudes de baliza para su campo. Recomendamos utilizar una longitud de baliza más corta para preservar la seguridad del conjunto de datos siempre que sea posible. Sin embargo, la longitud de la baliza que utilice realmente viene determinada por el modelo de amenaza. Tenga en cuenta sus requisitos de rendimiento al revisar su modelo de amenazas para determinar la mejor longitud de baliza para su campo.

   El uso de una longitud de baliza más corta reduce el rendimiento de las consultas, mientras que el uso de una longitud de baliza más larga reduce la seguridad. En general, si el conjunto de datos está distribuido de forma desigual, o si se construyen balizas distintas a partir de campos relacionados entre sí, es necesario utilizar longitudes de baliza más cortas para minimizar la cantidad de información revelada sobre la distribución de los conjunto de datos.

   Si revisa su modelo de amenazas y decide que cualquier información distintiva revelada sobre la distribución de un campo no representa una amenaza para su seguridad general, puede optar por utilizar una longitud de baliza superior al rango recomendado que calculó. Por ejemplo, si ha

calculado el rango recomendado de longitudes de baliza para un campo de 9 a 16 bits, puede optar por utilizar una longitud de baliza de 24 bits para evitar cualquier pérdida de rendimiento.

Elija la longitud de la baliza con cuidado. Una vez que haya escrito nuevos registros con la baliza, no podrá actualizar la longitud de la baliza.

Ejemplo

Considere una base de datos que marcara el `unit` campo como `ENCRYPT_AND_SIGN` parte de las acciones criptográficas. Para configurar una baliza estándar para el `unit` campo, necesitamos determinar el número esperado de falsos positivos y la longitud de la baliza para el `unit` campo.

1. Haga un estimado de la población

   Tras revisar nuestro modelo de amenazas y nuestra solución de base de datos actual, esperamos que el `unit` campo acabe teniendo 100 000 valores únicos.

   Esto significa que la Población = 100 000.

2. Calcule el rango recomendado para el número esperado de colisiones.

   Para este ejemplo, el número esperado de colisiones debe estar entre 2 y 316.

   ```
   2 ≤ number of collisions < √(Population)
   ```

   a.
   ```
   2 ≤ number of collisions < √(100,000)
   ```

   b.
   ```
   2 ≤ number of collisions < 316
   ```

3. Calcule el rango recomendado para la longitud de la baliza.

   Para este ejemplo, la longitud de la baliza debe estar entre 9 y 16 bits.

   $$\text{number of collisions} = \text{Population} * 2^{-(\text{beacon length})}$$

   a. Calcule la longitud de la baliza cuando el número esperado de colisiones sea igual al mínimo identificado en el Paso 2.

   $$2 = 100{,}000 * 2^{-(\text{beacon length})}$$

Longitud de la baliza = 15,6 o 15 bits

b. Calcule la longitud de la baliza cuando el número esperado de colisiones sea igual al máximo identificado en el Paso 2.

$$316 = 100{,}000 * 2^{-(\text{beacon length})}$$

Longitud de la baliza = 8,3 u 8 bits

4. Determine la longitud de baliza adecuada para sus requisitos de seguridad y rendimiento.

Por cada bit inferior a 15, el costo de rendimiento y la seguridad se duplican.

- 16 bits
  - En promedio, cada valor único se asignará a otras 1,5 unidades.
  - Seguridad: dos registros con la misma etiqueta HMAC truncada tienen un 66% de probabilidades de tener el mismo valor de texto no cifrado.
  - Rendimiento: una consulta recuperará 15 registros por cada 10 registros que realmente haya solicitado.
- 14 bits
  - En promedio, cada valor único se asignará a otras 6,1 unidades.
  - Seguridad: dos registros con la misma etiqueta HMAC truncada tienen un 33% de probabilidades de tener el mismo valor de texto no cifrado.
  - Rendimiento: una consulta recuperará 30 registros por cada 10 registros que realmente haya solicitado.

Elegir un nombre de baliza

Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

Cada baliza se identifica con un nombre de baliza único. Una vez configurada una baliza, el nombre de la baliza es el nombre que se utiliza para consultar un campo cifrado. El nombre de una baliza puede ser el mismo nombre que un campo cifrado o un campo virtual, pero no puede ser el mismo

nombre que un campo no cifrado. Dos balizas diferentes no pueden tener el mismo nombre de baliza.

Para ver ejemplos que muestran cómo nombrar y configurar balizas, consulte Configuración de balizas.

Denominación de baliza estándar

Al nombrar las balizas estándar, recomendamos encarecidamente que el nombre de la baliza se refiera a la fuente de la baliza siempre que sea posible. Esto significa que el nombre de la baliza y el nombre del campo cifrado o virtual a partir del cual se construye la baliza estándar son los mismos. Por ejemplo, si va a crear una baliza estándar para un campo cifrado denominado `LastName`, el nombre de la baliza también debería ser `LastName`.

Si el nombre de la baliza es el mismo que el de la fuente de la baliza, puede omitir la fuente de la baliza en la configuración y el SDK de cifrado de bases de datos de AWS utilizará automáticamente el nombre de la baliza como la fuente de la baliza.

Configuración de las balizas

> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

Existen dos tipos de balizas que admiten el cifrado para búsquedas. Las balizas estándar realizan búsquedas de igualdad. Son la forma más sencilla de implementar el cifrado para búsquedas en su base de datos. Las balizas compuestas combinan cadenas literales de texto no cifrado y balizas estándar para realizar consultas más complejas.

Las balizas están diseñadas para su implementación en bases de datos nuevas y despobladas. Cualquier baliza configurada en una base de datos existente solo mapeará los nuevos registros escritos en la base de datos. Las balizas se calculan a partir del valor de texto no cifrado de un campo. Una vez cifrado el campo, la baliza no tiene forma de mapear los datos existentes. Una vez que haya escrito nuevos registros con una baliza, no puede actualizar la configuración de la baliza. Sin embargo, puede agregar balizas nuevas para los campos nuevos que agregue a su registro.

Una vez determinados los patrones de acceso, la configuración de las balizas debería ser el segundo paso de la implementación de la base de datos. A continuación, después de configurar todas las balizas, debe crear un llavero jerárquico AWS KMS, definir la versión de las balizas, configurar un índice secundario para cada baliza, definir las acciones criptográficas y configurar la base de datos y el cliente del SDK de cifrado de bases de datos de AWS. Para obtener más información, consulte Utilizar balizas.

Para facilitar la definición de la versión de baliza, recomendamos crear listas de balizas estándar y compuestas. Agregue cada baliza que cree a la lista de balizas estándar o compuestas correspondiente a medida que las vaya configurando.

Temas

Configuración de balizas estándar

Las balizas estándar son la forma más sencilla de implementar el cifrado para búsquedas en su base de datos. Solo pueden realizar búsquedas de igualdad para un único campo virtual o cifrado.

Ejemplo de sintaxis de configuración

```
List<StandardBeacon> standardBeaconList = new ArrayList<>();
StandardBeacon exampleStandardBeacon = StandardBeacon.builder()
        .name("beaconName")
        .length(beaconLengthInBits)
        .loc("fieldName") // optional
        .build();
standardBeaconList.add(exampleStandardBeacon);
```

Para configurar una baliza estándar, proporcione los siguientes valores.

Nombre de la baliza

El nombre que se utiliza al consultar un campo cifrado.

El nombre de un indicador puede ser el mismo nombre que un campo cifrado o un campo virtual, pero no puede ser el mismo nombre que un campo no cifrado. Siempre que sea posible,

recomendamos encarecidamente utilizar el nombre del campo cifrado o el campo virtual con el que se construye la baliza estándar. Dos balizas diferentes no pueden tener el mismo nombre de baliza. Para obtener ayuda para determinar el mejor nombre de baliza para su implementación, consulte Elegir un nombre de baliza.

Longitud de la baliza

El número de bits del valor hash de la baliza que se conservan tras el truncamiento.

La longitud de la baliza determina el número medio de falsos positivos producidos por una baliza determinada. Para obtener más información y ayuda para determinar la longitud de baliza adecuada para su implementación, consulte Determinar la longitud de la baliza.

Fuente de la baliza (opcional)

El campo a partir del cual se construye una baliza estándar.

La fuente de la baliza debe ser un nombre de campo o un índice que haga referencia al valor de un campo anidado. Si el nombre de la baliza es el mismo que el de la fuente de la baliza, puede omitir la fuente de la baliza en la configuración y el SDK de cifrado de bases de datos de AWS utilizará automáticamente el nombre de la baliza como fuente de la baliza.

Creación de un campo virtual

Para crear un campo virtual, debe proporcionar un nombre para el campo virtual y una lista de los campos de origen. El orden en que se agregan los campos de origen a la lista de piezas virtuales determina el orden en que se concatenan para crear el campo virtual. El siguiente ejemplo concatena dos campos de origen en su totalidad para crear un campo virtual.

```
List<VirtualPart> virtualPartList = new ArrayList<>();
    virtualPartList.add(sourceField1);
    virtualPartList.add(sourceField2);

VirtualField virtualFieldName = VirtualField.builder()
    .name("virtualFieldName")
    .parts(virtualPartList)
    .build();

List<VirtualField> virtualFieldList = new ArrayList<>();
    virtualFieldList.add(virtualFieldName);
```

Para crear un campo virtual con un segmento específico de un campo de origen, debe definir esa transformación antes de agregar el campo de origen a la lista de piezas virtuales.

Aspectos de seguridad para campos virtuales

Las balizas no alteran el estado cifrado del campo. Sin embargo, cuando se utilizan balizas, existe un equilibrio inherente entre la eficacia de las consultas y la cantidad de información que se revela sobre la distribución de los datos. La forma en que configura su baliza determina el nivel de seguridad que preserva esa baliza.

Evite crear un campo virtual con campos de origen que se superpongan con las balizas estándar existentes. La creación de campos virtuales que incluyan un campo de origen que ya se haya utilizado para crear una baliza estándar puede reducir el nivel de seguridad de ambas balizas. La medida en que se reduzca la seguridad depende del nivel de entropía agregado por los campos de origen adicionales. El nivel de entropía está determinado por la distribución de valores únicos en el campo de origen adicional y el número de bits que el campo de origen adicional aporta al tamaño total del campo virtual.

Puede usar la población y la longitud de la baliza para determinar si los campos de origen de un campo virtual preservan la seguridad del conjunto de datos. La población es el número esperado de valores únicos en un campo. Su población no tiene por qué ser precisa. Para obtener ayuda para estimar la población de un campo, consulte Estimar la población.

Considere el siguiente ejemplo al revisar la seguridad de sus campos virtuales.

- Beacon1 se construye a partir de `FieldA`. `FieldA`tiene una población superior a $2^{\text{(longitud de Beacon1)}}$.
- Beacon2 se construye a partir de `VirtualField`, que se construye a partir de `FieldA`, `FieldB`, `FieldC` y `FieldD`. Juntos, `FieldB`, `FieldC` y `FieldD` tienen una población superior a 2^N

Beacon2 preserva la seguridad tanto de Beacon1 como de Beacon2 si se cumplen las siguientes afirmaciones:

```
N ≥ (Beacon1 length)/2
```

y

```
N ≥ (Beacon2 length)/2
```

Configuración de balizas compuestas

Las balizas compuestas combinan cadenas literales de texto no cifrado y balizas estándar para realizar operaciones complejas de bases de datos, como consultar dos tipos de registros diferentes desde un único índice o consultar una combinación de campos con una clave de clasificación. Las balizas compuestas se pueden construir a partir de los campos `ENCRYPT_AND_SIGN` y `SIGN_ONLY`. Debe crear una baliza estándar para cada campo cifrado incluido en la baliza compuesta.

Ejemplo de sintaxis de configuración

```
List<CompoundBeacon> compoundBeaconList = new ArrayList<>();
    CompoundBeacon exampleCompoundBeacon = CompoundBeacon.builder()
        .name("compoundBeaconName")
        .split(".")
        .encrypted(encryptedPartList)
        .signed(signedPartList) // optional
        .constructors(constructorList) // optional
        .build();
    compoundBeaconList.add(exampleCompoundBeacon);
```

Para configurar una baliza compuesta, proporcione los siguientes valores.

Nombre de la baliza

El nombre que se utiliza al consultar un campo cifrado.

El nombre de un indicador puede ser el mismo nombre que un campo cifrado o un campo virtual, pero no puede ser el mismo nombre que un campo no cifrado. No puede haber dos balizas con el mismo nombre de baliza. Para obtener ayuda para determinar el mejor nombre de baliza para su implementación, consulte Elegir un nombre de baliza.

Carácter dividido

El personaje que se usa para separar las partes que conforman su baliza compuesta.

El carácter dividido no puede aparecer en los valores de texto no cifrado de ninguno de los campos a partir de los que se construye la baliza compuesta.

Lista de piezas cifradas

Identifica los campos `ENCRYPT_AND_SIGN` incluidos en la baliza compuesta.

Cada pieza debe incluir un nombre y un prefijo. El nombre de la pieza debe ser el nombre de la baliza estándar construida a partir del campo cifrado. El prefijo puede ser cualquier cadena, pero debe ser único. Una parte cifrada no puede tener el mismo prefijo que una parte firmada. Recomendamos utilizar un valor corto que distinga la pieza de otras partes servidas por la baliza compuesta. Para simplificar las consultas de balizas, también le recomendamos que identifique una parte con el mismo prefijo en todas las balizas en las que esté incluida y que evite usar el mismo prefijo para identificar diferentes piezas.

```
List<EncryptedPart> encryptedPartList = new ArrayList<>);
    EncryptedPart encryptedPartExample = EncryptedPart.builder()
        .name("standardBeaconName")
        .prefix("E-")
        .build();
    encryptedPartList.add(encryptedPartExample);
```

Lista de piezas firmadas (opcional)

Identifique los `SIGN_ONLY` campos incluidos en la baliza compuesta.

Cada parte debe incluir un nombre, una fuente y un prefijo. La fuente es el campo `SIGN_ONLY` que identifica la pieza. La fuente debe ser un nombre de campo o un índice que haga referencia al valor de un campo anidado. Si el nombre de la pieza identifica la fuente, puede omitirla y el SDK de cifrado de bases de datos de AWS utilizará automáticamente el nombre como su fuente. Recomendamos especificar la fuente como nombre de la pieza siempre que sea posible. El prefijo puede ser cualquier cadena, pero debe ser único. Una pieza firmada no puede tener el mismo prefijo que una parte cifrada. Recomendamos utilizar un valor corto que distinga la pieza de otras partes servidas por la baliza compuesta. Para simplificar las consultas de balizas, también le recomendamos que identifique una parte con el mismo prefijo en todas las balizas en las que esté incluida y que evite usar el mismo prefijo para identificar diferentes piezas.

```
List<SignedPart> signedPartList = new ArrayList<>);
    SignedPart signedPartExample = SignedPart.builder()
        .name("signedFieldName")
        .prefix("S-")
        .build();
    signedPartList.add(signedPartExample);
```

Lista de constructores (opcional)

Identifica los constructores que definen las diferentes formas en que la baliza compuesta puede ensamblar las piezas cifradas y firmadas.

Si no especifica una lista de constructores, el SDK de cifrado de bases de datos de AWS ensambla la baliza compuesta con el siguiente constructor predeterminado.

- Todas las piezas firmadas en el orden en que se agregaron a la lista de piezas firmadas
- Todas las piezas cifradas en el orden en que se agregaron a la lista de piezas cifradas
- Todas las piezas son obligatorias

Constructores

Cada constructor es una lista ordenada de piezas del constructor que define una forma en la que se puede ensamblar la baliza compuesta. Las piezas del constructor se unen en el orden en que se agregan a la lista, con cada parte separada por el carácter dividido especificado.

Cada parte del constructor nombra una parte cifrada o una parte firmada y define si esa parte es obligatoria u opcional en el constructor. Por ejemplo, si desea consultar una baliza compuesta sobre `Field1`, `Field1.Field2` y `Field1.Field2.Field3`, marque `Field2` y `Field3` como opcional y cree un constructor.

Cada constructor debe tener como mínimo una pieza requerida. Recomendamos hacer que la primera parte de cada constructor sea obligatoria para poder usar el operador `BEGINS_WITH` en las consultas.

Un constructor tiene éxito si todas las piezas requeridas están presentes en el registro. Al escribir un registro nuevo, la baliza compuesta utiliza la lista de constructores para determinar si la baliza se puede ensamblar a partir de los valores proporcionados. Intente ensamblar la baliza en el orden en que se agregaron los constructores a la lista de constructores y utilice el primer constructor que lo haga correctamente. Si ningún constructor tiene éxito, la baliza no se graba en el registro.

Todos los lectores y redactores deben especificar el mismo orden de constructores para garantizar que los resultados de sus consultas sean correctos.

Utilice los siguientes procedimientos para especificar su propia lista de constructores.

1. Cree una parte constructora para cada pieza cifrada y firmada para definir si esa pieza es necesaria o no.

El nombre de la pieza constructora debe ser el nombre de la baliza estándar o el campo firmado que representa.

```
ConstructorPart field1ConstructorPart = ConstructorPart.builder()
        .name("Field1")
        .required(true)
        .build();
```

2. Cree un constructor para cada forma posible de ensamblaje de la baliza compuesta utilizando las piezas constructoras que creó en el Paso 1.

   Por ejemplo, si desea consultar sobre `Field1.Field2.Field3` y `Field4.Field2.Field3`, debe crear dos constructores. Tanto `Field1` como `Field4` pueden ser necesarios porque están definidos en dos constructores distintos.

```
// Create a list for Field1.Field2.Field3 queries
List<ConstructorPart> field123ConstructorPartList = new ArrayList<>();
field123ConstructorPartList.add(field1ConstructorPart);
field123ConstructorPartList.add(field2ConstructorPart);
field123ConstructorPartList.add(field3ConstructorPart);
Constructor field123Constructor = Constructor.builder()
        .parts(field123ConstructorPartList)
        .build();
// Create a list for Field4.Field2.Field1 queries
List<ConstructorPart> field421ConstructorPartList = new ArrayList<>();
field421ConstructorPartList.add(field4ConstructorPart);
field421ConstructorPartList.add(field2ConstructorPart);
field421ConstructorPartList.add(field1ConstructorPart);
Constructor field421Constructor = Constructor.builder()
        .parts(field421ConstructorPartList)
        .build();
```

3. Cree una lista de constructores que incluya todos los constructores que creó en el Paso 2.

```
List<Constructor> constructorList = new ArrayList<>();
constructorList.add(field123Constructor)
constructorList.add(field421Constructor)
```

4. Especifica el `constructorList` cuando creó su baliza firmada.

Configuraciones de ejemplo

> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

En los ejemplos siguientes, se muestra cómo configurar balizas estándar y compuestas. Las siguientes configuraciones no proporcionan longitudes de baliza. Para obtener ayuda para determinar la longitud adecuada de la baliza para su configuración, consulte Elegir longitud de una baliza.

Para ver ejemplos de código completos que muestran cómo configurar y usar balizas, consulte el directorio searchableencryption del repositorio aws-database-encryption-sdk-dynamodb-java en GitHub.

Temas

Balizas estándar

Si desea consultar las coincidencias exactas en el campo `inspector_id_last4`, cree una baliza estándar con la siguiente configuración.

```
List<StandardBeacon> standardBeaconList = new ArrayList<>();
StandardBeacon exampleStandardBeacon = StandardBeacon.builder()
        .name("inspector_id_last4")
        .length(beaconLengthInBits)
        .build();
standardBeaconList.add(exampleStandardBeacon);
```

Balizas compuestas

Si desea consultar la `UnitInspection` base de datos sobre `inspector_id_last4` y `inspector_id_last4.unit`, cree una baliza compuesta con la siguiente configuración. Esta baliza compuesta solo requiere partes cifradas.

```
// 1. Create standard beacons for the inspector_id_last4 and unit fields.
    List<StandardBeacon> standardBeaconList = new ArrayList<>);
    StandardBeacon inspectorBeacon = StandardBeacon.builder()
        .name("inspector_id_last4")
        .length(beaconLengthInBits)
        .build();
    standardBeaconList.add(inspectorBeacon);
    StandardBeacon unitBeacon = StandardBeacon.builder()
        .name("unit")
        .length(beaconLengthInBits)
        .build();
    standardBeaconList.add(unitBeacon);
// 2. Define the encrypted parts.
    List<EncryptedPart> encryptedPartList = new ArrayList<>);
// Each encrypted part needs a name and prefix
// The name must be the name of the standard beacon
// The prefix must be unique
// For this example we use the prefix "I-" for "inspector_id_last4"
// and "U-" for "unit"
    EncryptedPart encryptedPartInspector = EncryptedPart.builder()
        .name("inspector_id_last4")
        .prefix("I-")
        .build();
    encryptedPartList.add(encryptedPartInspector);
    EncryptedPart encryptedPartUnit = EncryptedPart.builder()
        .name("unit")
        .prefix("U-")
        .build();
    encryptedPartList.add(encryptedPartUnit);
// 3. Create the compound beacon.
// This compound beacon only requires a name, split character,
// and list of encrypted parts
    CompoundBeacon inspectorUnitBeacon = CompoundBeacon.builder()
        .name("inspectorUnitBeacon")
        .split(".")
        .sensitive(encryptedPartList)
        .build();
```

Uso de balizas

> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

Las balizas permiten buscar registros cifrados sin necesidad de descifrar toda la base de datos consultada. Las balizas están diseñadas para su implementación en bases de datos nuevas y despobladas. Cualquier baliza configurada en una base de datos existente solo mapeará los nuevos registros escritos en la base de datos. Las balizas se calculan a partir del valor de texto no cifrado de un campo. Una vez cifrado el campo, la baliza no tiene forma de mapear los datos existentes. Una vez que haya escrito nuevos registros con una baliza, no puede actualizar la configuración de la baliza. Sin embargo, puede agregar balizas nuevas para los campos nuevos que agrega a su registro.

Tras configurar las balizas, debe completar los siguientes pasos antes de empezar a rellenar la base de datos y a realizar consultas en las balizas.

1. Cree un AWS KMS llavero jerárquico

   Para utilizar el cifrado con capacidad de búsqueda, debe utilizar el AWS KMSllavero jerárquico para generar, cifrar y descifrar las claves de datos que se utilizan para proteger sus registros.

   Después de configurar sus balizas, reúna los requisitos previos del llavero jerárquico y cree su llavero jerárquico.

   Para obtener más información sobre por qué se requiere el llavero jerárquico, consulte Uso del llavero jerárquico para el cifrado con capacidad de búsqueda.

2.

   Defina la versión de baliza

   Especifique `keyStore`, `keySource`, una lista de todas las balizas estándar que configuró, una lista de todas las balizas compuestas que configuró y una versión de la baliza. Debe especificar la versión 1 de la baliza. Para obtener orientación sobre cómo definir su `keySource`, consulte Definir la fuente de claves de baliza.

El siguiente ejemplo de Java define la versión de baliza para una base de datos de un solo inquilino. Para obtener ayuda para definir la versión de baliza para una base de datos de multitenencia, consulte Cifrado con capacidad de búsqueda para bases de datos de multitenencia.

```
List<BeaconVersion> beaconVersions = new ArrayList<>();
    beaconVersions.add(
        BeaconVersion.builder()
            .standardBeacons(standardBeaconList)
            .compoundBeacons(compoundBeaconList)
            .version(1) // MUST be 1
            .keyStore(branchKeyStoreName)
            .keySource(BeaconKeySource.builder()
                .single(SingleKeyStore.builder()
                    .keyId(branch-key-id)
                    .cacheTTL(6000)
                    .build())
                .build())
            .build()
    );
```

3. Configure los índices secundarios

   Después de configurar las balizas, debe configurar un índice secundario que refleje cada baliza antes de poder buscar en los campos cifrados. Para obtener más información, consulte Configurar índices secundarios con balizas.

4. Defina sus acciones criptográficas

   Todos los campos utilizados para construir una baliza estándar deben estar marcados `ENCRYPT_AND_SIGN`. Todos los demás campos utilizados para construir balizas deben estar marcados `SIGN_ONLY`.

5. Configure un cliente SDK de cifrado de bases de datos de AWS

   Para configurar un cliente SDK de cifrado de bases de datos de AWS que proteja los elementos de la tabla de DynamoDB, consulte la biblioteca de cifrado del cliente de Java para DynamoDB.

Balizas de consulta

El tipo de baliza que configure determinará el tipo de consultas que podrá realizar. Las balizas estándar utilizan expresiones de filtro para realizar búsquedas de igualdad. Las balizas compuestas combinan cadenas literales de texto no cifrado y balizas estándar para realizar consultas complejas. Al consultar datos cifrados, se busca por el nombre de la baliza.

No se pueden comparar los valores de dos balizas estándar, aunque contengan el mismo texto no cifrado subyacente. Las dos balizas estándar generarán dos etiquetas HMAC diferentes para los mismos valores de texto no cifrado. Como resultado, las balizas estándar no pueden realizar las siguientes consultas.

- *beacon1* = *beacon2*
- *beacon1* IN (*beacon2*)
- *value* IN (*beacon1*, *beacon2*, ...)
- CONTAINS(*beacon1*, *beacon2*)

Las balizas compuestas pueden realizar las siguientes consultas.

- BEGINS_WITH(*a*), dónde *a* refleja el valor total del campo por el que comienza la baliza compuesta ensamblada. No puede utilizar el BEGINS_WITH operador para identificar un valor que comience con una subcadena determinada. Sin embargo, puede usarBEGINS_WITH(*S_*), donde *S_* refleja el prefijo de una parte por la que comienza la baliza compuesta ensamblada.
- CONTAINS(*a*), donde *a* refleja el valor total de un campo que contiene la baliza compuesta ensamblada. No puede usar el CONTAINS operador para identificar un registro que contenga una subcadena concreta o un valor dentro de un conjunto.

  Por ejemplo, no puede realizar una consulta CONTAINS(*path*, "*a*" en la que *a* se refleje el valor de un conjunto.
- Puede comparar partes firmadas de balizas compuestas. Al comparar partes firmadas, si lo desea, puede agregar el prefijo de una parte cifrada a una o más partes firmadas, pero no puede incluir el valor de un campo cifrado en ninguna consulta.

  Por ejemplo, puede comparar las partes firmadas y realizar consultas en *signedField1* = *signedField2* o *value* IN (*signedField1*, *signedField2*, ...).

También puede comparar las partes firmadas y el prefijo de una parte cifrada consultando *signedField1.A_* = *signedField2.B_*.

- *field* BETWEEN *a* AND *b*, donde *a* y *b* son partes firmadas. Si lo desea, puede añadir el prefijo de una parte cifrada a una o más partes firmadas, pero no puede incluir el valor de un campo cifrado en ninguna consulta.

Debe incluir el prefijo de cada parte que incluya en una consulta en una baliza compuesta. Por ejemplo, si ha creado una baliza compuesta, `compoundBeacon`, a partir de dos campos, `encryptedField` y `signedField`, debe incluir los prefijos configurados para esas dos partes cuando consulte la baliza.

```
compoundBeacon = E_encryptedFieldValue.S_signedFieldValue
```

Cifrado con capacidad de búsqueda para bases de datos multitenencia

Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

Para implementar un cifrado con capacidad de búsqueda en su base de datos, debe utilizar un AWS KMS llavero jerárquico. El AWS KMS llavero jerárquico genera, cifra y descifra las claves de datos utilizadas para proteger sus registros. También crea la clave de baliza que se utiliza para generar balizas. Cuando se utiliza el llavero AWS KMS jerárquico con bases de datos multitenencia, hay una clave de rama y una clave de baliza distintas para cada inquilino. Para consultar datos cifrados en una base de datos de multitenencia, debe identificar los materiales clave de baliza utilizados para generar la baliza que está consultando.

Al definir la versión de baliza para una base de datos de multitenencia, especifique una lista de todas las balizas estándar que configuró, una lista de todas las balizas compuestas que configuró, una versión de baliza y una `keySource`. Debe definir la fuente de claves de baliza como una `MultiKeyStore`, e incluir una`keyFieldName`, el tiempo de vida de la caché de claves de baliza local y el tamaño máximo de la caché de claves de baliza local.

Si configuró alguna baliza firmada, debe incluirla en su `compoundBeaconList`. Las balizas firmadas son un tipo de baliza compuesta que indexan y realizan consultas complejas en los campos `SIGN_ONLY`.

```
List<BeaconVersion> beaconVersions = new ArrayList<>();
    beaconVersions.add(
        BeaconVersion.builder()
                .standardBeacons(standardBeaconList)
                .compoundBeacons(compoundBeaconList)
                .version(1) // MUST be 1
                .keyStore(branchKeyStoreName)
                .keySource(BeaconKeySource.builder()
                        .multi(MultiKeyStore.builder()
                                .keyFieldName(keyField)
                                .cacheTTL(6000)
                                .maxCacheSize(10)
                        .build())
                .build())
        .build()
    );
```

keyFieldName

El `keyFieldName` define el nombre del campo que almacena la `branch-key-id` asociada a la baliza utilizada para generar las balizas para un inquilino determinado.

Cuando se escriben nuevos registros en la base de datos, en este campo se almacena la `branch-key-id` de baliza utilizada para generar las balizas de ese registro.

De forma predeterminada, el `keyField` es un campo conceptual que no se almacena de forma explícita en la base de datos. El SDK de cifrado de bases de datos de AWS identifica la `branch-key-id` de la clave de datos cifrados en la descripción del material y almacena el valor en el formato conceptual `keyField` para que pueda consultarlo en las balizas compuestas y balizas firmadas. Como la descripción del material está firmada, lo conceptual `keyField` se considera una parte firmada.

También puede incluir el`keyField` en sus acciones criptográficas como un campo `SIGN_ONLY` para almacenar explícitamente el campo en la base de datos. Si lo hace, debe incluir manualmente el `branch-key-id` en la `keyField` cada vez que escriba un registro en la base de datos.

Consulta de balizas en una base de datos de multitenencia

Para consultar una baliza, debe incluir la `keyField` en la consulta los materiales clave necesarios para volver a calcular la baliza. Debe especificar la `branch-key-id` asociada a la baliza utilizada para generar las balizas para un registro. No puede especificar el nombre amigable que identifica al `branch-key-id` del inquilino en el proveedor de ID de clave de rama. Puede incluir el `keyField` en sus consultas de las siguientes maneras.

Balizas compuestas

Ya sea que las almacene de forma explícita `keyField` en sus registros o no, puede incluir las `keyField` directamente en sus balizas compuestas como una parte firmada. La parte `keyField` firmada debe ser obligatoria.

Por ejemplo, si desea construir una baliza compuesta, `compoundBeacon`, a partir de dos campos, `encryptedField` y `signedField`, también debe incluir la baliza `keyField` como parte firmada. Esto permite realizar la siguiente consulta en `compoundBeacon`.

```
compoundBeacon = E_encryptedFieldValue.S_signedFieldValue.K_branch-key-id
```

Balizas firmadas

El SDK de cifrado de bases de datos de AWS utiliza balizas estándar y compuestas para proporcionar soluciones de cifrado con capacidad de búsqueda. Estas balizas deben incluir al menos un campo cifrado. Sin embargo, el SDK de cifrado de bases de datos de AWS también admite balizas firmadas que se pueden configurar completamente a partir de campos `SIGN_ONLY` de texto no cifrado.

Las balizas firmadas se pueden construir a partir de una sola pieza. Ya sea que las almacene de forma explícita `keyField` en sus registros o no, puede crear una baliza firmada a partir de ella `keyField` y utilizarla para crear consultas compuestas que combinen una consulta en la baliza `keyField` firmada con una consulta en una de las otras balizas. Por ejemplo, puede realizar la siguiente consulta.

```
keyField = K_branch-key-id AND compoundBeacon =
 E_encryptedFieldValue.S_signedFieldValue
```

Si necesita ayuda para configurar las balizas firmadas, consulte Crear balizas firmadas

Realice consultas directamente en el **`keyField`**

Si lo especificó `keyField` en sus acciones criptográficas y almacenó el campo de forma explícita en su registro, puede crear una consulta compuesta que combine una consulta de su baliza con una consulta de`keyField`. Puede optar por realizar una consulta directamente en el `keyField` si desea consultar una baliza estándar. Por ejemplo, puede realizar la siguiente consulta.

```
keyField = branch-key-id AND standardBeacon = S_standardBeaconValue
```

SDK de cifrado de bases de datos de AWS para DynamoDB

Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

El SDK de cifrado de bases de datos de AWS para DynamoDB es una biblioteca de software que le permite incluir el cifrado del cliente en su diseño de Amazon DynamoDB. El SDK de cifrado de bases de datos de AWS para DynamoDB proporciona cifrado a nivel de atributos y le permite especificar qué elementos cifrar y qué elementos incluir en las firmas para garantizar la autenticidad de los datos. El cifrado de sus datos en tránsito y en reposo confidenciales ayuda a garantizar que los datos de texto no cifrado no estén disponibles para ningún tercero, incluido AWS.

> **Note**
>
> Los siguientes temas se centran en la versión 3. x de la biblioteca de cifrado del cliente de Java para DynamoDB.
> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. El SDK de cifrado de bases de datos de AWS sigue siendo compatible con las versiones antiguas de DynamoDB Encryption Client.

En DynamoDB una tabla es una colección de elementos. Cada elemento es una colección de atributos. Cada atributo tiene un nombre y un valor. El SDK de cifrado de bases de datos de AWS para DynamoDB cifra los valores de los atributos. A continuación, calcula una firma sobre los atributos. Puede especificar qué valores de atributo cifrar y cuáles incluir en la firma en las acciones criptográficas???.

Los temas de este capítulo proporcionan información general sobre el SDK de cifrado de bases de datos de AWS para DynamoDB, incluidos los campos que se cifran, instrucciones sobre la instalación y configuración del cliente y ejemplos de Java para ayudarle a empezar.

Temas

- cifrado del cliente o del lado del servidor
- ¿Qué campos se cifran y se firman?

- Java
- Cliente de cifrado de DynamoDB antiguo

cifrado del cliente o del lado del servidor

> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

El SDK de cifrado de bases de datos de AWS para DynamoDB admite el cifrado del cliente, que permite cifrar los datos de la tabla antes de enviarlos a su base de datos. Sin embargo, admite una característica de cifrado en reposo que cifra de modo transparente la tabla cuando se almacena en disco y la descifra cuando accede a la tabla.

Las herramientas que elija dependen de la confidencialidad de sus datos y de los requisitos de seguridad de su aplicación. Puede usar tanto el SDK de cifrado de bases de datos de AWS para DynamoDB como el cifrado en reposo. Cuando envía elementos cifrados y firmados a , no reconoce los elementos como protegidos. Detecta elementos de tabla típicos con valores de atributo binario.

Cifrado del lado del servidor en reposo

DynamoDB admite el cifrado en reposo, una característica de cifrado del servidor en la que DynamoDB cifra de modo transparente las tablas cuando la tabla se almacena en disco y las descifra cuando se accede a los datos de la tabla.

Cuando usa un SDK de AWS para interactuar con DynamoDB, los datos se cifran en tránsito a través de una conexión HTTPS automáticamente, se descifran en el punto de conexión de DynamoDB y, a continuación, se vuelven a cifrar antes de almacenarse en DynamoDB.

- Cifrado de forma predeterminada. DynamoDB cifra y descifra de forma transparente todas las tablas cuando se escriben. No existe la opción de habilitar o deshabilitar el cifrado en reposo.
- DynamoDB crea y administra las claves criptográficas. La clave única de cada tabla está protegida por un AWS KMS key que nunca se deja AWS Key Management Service (AWS KMS) sin cifrar. De forma predeterminada, DynamoDB utiliza una Clave propiedad de AWS en la cuenta de servicio de DynamoDB, pero puede elegir una Clave administrada de AWS o una clave gestionada por el cliente en su cuenta para proteger algunas o todas sus tablas.

- Todos los datos de la tabla se cifran en disco. Cuando una tabla cifrada se guarda en disco, cifra todos los datos de la tabla, incluida la clave principal y los índices secundarios locales y globales. Si la tabla tiene una clave de clasificación, algunas de las claves de ordenación que marcan los límites del rango se almacenan en texto no cifrado en los metadatos de la tabla.
- Los objetos relacionados con las tablas también están cifrados. El cifrado en reposo protege los flujos de , las tablas globales y las copias de seguridad cada vez que se escriben en medios duraderos.
- Sus elementos se descifran cuando accede a ellos. Cuando accede a la tabla, DynamoDB descifra la parte de la tabla que incluye el elemento de destino y le devuelve el elemento de texto no cifrado.

SDK de cifrado de bases de datos de AWS para DynamoDB

El cifrado del cliente proporciona protección integral para sus datos, en tránsito y en reposo, desde su origen al almacenamiento en DynamoDB. Sus datos de texto no cifrado no se exponen nunca a terceros, incluido AWS. Puede utilizar el SDK de cifrado de bases de datos de AWS para DynamoDB con nuevas tablas de DynamoDB o migrar las tablas existentes de Amazon DynamoDB a la versión 3. x de la biblioteca de cifrado del cliente de Java para DynamoDB.

- Sus datos se protegen en tránsito y en reposo. Nunca se exponen a terceros, incluido AWS.
- Puede firmar sus elementos de tabla. Puede dirigir el SDK de cifrado de bases de datos de AWS para DynamoDB para calcular una firma sobre todo o parte de un elemento de tabla, incluidos los atributos de clave principal y el nombre de la tabla. Esta firma permite detectar cambios no autorizados en el elemento en general, incluida la adición o eliminación de atributos o el cambio de valores de atributo.
- Para determinar cómo se protegen los datos, seleccione un conjunto de claves. Su llavero determina las claves de encapsulación que protegen sus claves de datos y, en última instancia, sus datos. Utilice las claves de encapsulación más seguras que resulten prácticas para su tarea.
- El SDK de cifrado de bases de datos de AWS para DynamoDB no cifra toda la tabla. Usted elige qué atributos se cifrarán en sus elementos. El SDK de cifrado de bases de datos de AWS para DynamoDB no cifra un elemento completo. No cifra nombres de atributo o los nombres o valores de los atributos de clave principal (clave de partición y clave de clasificación).

AWS Encryption SDK

Si va a cifrar los datos que almacena en DynamoDB, le recomendamos el SDK de cifrado de bases de datos de AWS para DynamoDB.

La AWS Encryption SDK es una biblioteca de cifrado del cliente que le ayuda a cifrar y descifrar datos genéricos. Aunque puede proteger cualquier tipo de datos, no se ha diseñado para funcionar con datos estructurados, como registros de base de datos. A diferencia del SDK de cifrado de bases de datos de AWS para DynamoDB, el AWS Encryption SDK no puede proporcionar comprobación de integridad de nivel de elemento y no tiene lógica para reconocer atributos o evitar el cifrado de claves principales.

Si utiliza el AWS Encryption SDK para cifrar cualquier elemento de su tabla, recuerde que no es compatible con el SDK de cifrado de bases de datos de AWS para DynamoDB. No puede cifrar con una biblioteca y descifrar con la otra.

¿Qué campos se cifran y se firman?

> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. Esta guía para desarrolladores sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

El SDK de cifrado de bases de datos de AWS para DynamoDB es una biblioteca de cifrado del cliente que se diseñó especialmente para las aplicaciones de Amazon DynamoDB. Amazon DynamoDB almacena los datos en tablas, que son un conjunto de elementos. Cada elemento es una colección de atributos. Cada atributo tiene un nombre y un valor. El SDK de cifrado de bases de datos de AWS para DynamoDB cifra los valores de los atributos. A continuación, calcula una firma sobre los atributos. Puede especificar qué valores de atributo cifrar y cuáles incluir en la firma.

El cifrado protege la confidencialidad del valor de atributo. La firma proporciona integridad de todos los atributos firmados y de sus relaciones entre sí y proporciona autenticación. Le permite detectar cambios no autorizados en el elemento en general, incluida la adición o eliminación de atributos o la sustitución de un valor cifrado por otro.

En un elemento cifrado, algunos datos permanecen en texto no cifrado, incluido el nombre de la tabla, todos los nombres de atributo, los valores de atributo que no cifra y los nombres y valores de los atributos de la clave principal (clave de partición y clave de clasificación). No almacene información confidencial en estos campos.

Para obtener más información sobre el funcionamiento del SDK de cifrado de bases de datos de AWS para DynamoDB, consulte Cómo funciona el SDK de cifrado de bases de datos de AWS.

> **Note**
>
> Todas las menciones a las acciones de atributos en los temas del SDK de cifrado de bases de datos de AWS para DynamoDB se refieren a acciones criptográficas.

Temas

- Cifrado de valores de atributos
- Firma del elemento

Cifrado de valores de atributos

El SDK de cifrado de bases de datos de AWS para DynamoDB cifra los valores (pero no el nombre o el tipo de atributo) de los atributos que especifique. Para determinar los valores de atributo que se cifran, utilice acciones de atributo.

Por ejemplo, este elemento incluye los atributos `example` y `test`.

```
'example': 'data',
'test': 'test-value',
...
```

Si cifra el atributo `example`, pero no cifra el atributo `test`, el resultado tendrá el siguiente aspecto. El valor de atributo `example` cifrado son datos binarios, en lugar de una cadena.

```
'example': Binary(b"'b\x933\x9a+s\xf1\xd6a\xc5\xd5\x1aZ\xed\xd6\xce\xe9X\xf0T\xcb\x9fY
\x9f\xf3\xc9C\x83\r\xbb\\"),
'test': 'test-value'
...
```

Los atributos de clave principal (clave de partición y clave de clasificación) de cada elemento deben permanecer en texto no cifrado porque DynamoDB los utiliza para buscar el elemento en la tabla. Deben estar firmados, pero no cifrados.

El SDK de cifrado de bases de datos de AWS para DynamoDB identifica los atributos clave principal y garantiza que sus valores estén firmados, pero no cifrados. Y, si identifica la clave principal y, a continuación, intenta cifrarla, el cliente generará una excepción.

El cliente guarda la descripción del material en un nuevo atributo (`aws_dbe_head`) que agrega al elemento. La descripción del material describe cómo se cifró y firmó el elemento. El cliente utiliza esta información para verificar y descifrar el elemento. El campo que almacena la descripción del material no está cifrado.

Firma del elemento

Tras cifrar los valores de los atributos especificados, el SDK de cifrado de bases de datos de AWS para DynamoDB calcula los códigos de autenticación de mensajes basados en hash (HMAC) y una firma digital mediante la canonicalización de la descripción del material, el contexto de cifrado y cada campo marcado `ENCRYPT_AND_SIGN` o `SIGN_ONLY` en las acciones de los atributos. Las firmas ECDSA están habilitadas de forma predeterminada, pero no son obligatorias. El cliente guarda las HMAC y firmas en un nuevo atributo (`aws_dbe_foot`) que agrega al elemento.

Java

Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

En este tema se explica cómo instalar y usar la versión 3. x de la biblioteca de cifrado del cliente de Java para DynamoDB. Para obtener detalles sobre la programación con el SDK de cifrado de bases de datos de AWS para DynamoDB, consulte el directorio de ejemplos del repositorio aws-database-encryption-sdk-dynamodb-java en GitHub.

> **Note**
>
> Los siguientes temas se centran en la versión 3. x de la biblioteca de cifrado del cliente de Java para DynamoDB.
> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. El SDK de cifrado de bases de datos de AWS sigue siendo compatible con las versiones antiguas de DynamoDB Encryption Client.

Temas

Requisitos previos

Antes de instalar la versión 3. x de la biblioteca de cifrado del cliente de Java para DynamoDB, 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).

AWS SDK for Java 2.x

El SDK de cifrado de bases de datos de AWS para DynamoDB requiere el módulo Cliente mejorado de DynamoDB del AWS SDK for Java 2.x. Puede instalar todo el SDK o solo este módulo.

Para obtener información sobre cómo actualizar su versión deAWS SDK for Java, consulte Migración de la versión 1.x a la 2.x del. AWS SDK for Java

El AWS SDK for Java está disponible en Apache Maven. Puede declarar una dependencia para todo AWS SDK for Java el `dynamodb-enhanced` módulo o solo para él.

Instálelo AWS SDK for Java con Apache Maven

- Para importar todo AWS SDK for Java como una dependencia declárelo en el archivo `pom.xml`.

- Para crear una dependencia solo para el módulo Amazon DynamoDB en el AWS SDK for Java, siga las instrucciones para especificar módulos concretos. Establece el `groupId` para y el para.`software.amazon.awssdk` `artifactID` `dynamodb-enhanced`

  > **Note**
  >
  > Si usa el llavero AWS KMS o el llavero jerárquico AWS KMS, también necesita crear una dependencia para el módulo AWS KMS. Establece el `groupId` para y el para.`software.amazon.awssdk` `artifactID` `kms`

Instalación

Puede instalar la versión 3. x de la biblioteca de cifrado del cliente de Java para DynamoDB de las siguientes maneras.

Con Apache Maven

El Cliente de encriptación de Amazon DynamoDB para Java está disponible en Apache Maven con la siguiente definición de dependencias.

```
<dependency>
  <groupId>software.amazon.cryptography</groupId>
  <artifactId>aws-database-encryption-sdk-dynamodb</artifactId>
  <version>version-number</version>
</dependency>
```

Uso de Gradle Kotlin

Puede usar Gradle para declarar una dependencia en el Cliente de encriptación de Amazon DynamoDB para Java añadiendo lo siguiente a la sección de dependencias de su proyecto de Gradle.

```
implementation("software.amazon.cryptography:aws-database-encryption-sdk-
dynamodb:version-number")
```

Manualmente

Para instalar la biblioteca de cifrado del cliente de Java para DynamoDB, clone o descargue el repositorio de GitHub aws-database-encryption-sdk-dynamodb-java.

Después de instalar el SDK, comienza consultando el código de ejemplo de esta guía y el directorio de ejemplos del repositorio aws-database-encryption-sdk-dynamodb-java en GitHub.

Usar la biblioteca de cifrado del cliente de Java para DynamoDB

> Se cambió el nombre de nuestra biblioteca de cifrado del lado del cliente por el de SDK de cifrado de AWS bases de datos. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

En este tema se explican algunas de las funciones y clases de ayuda de la versión 3.x de la biblioteca de cifrado del cliente de Java para DynamoDB.

Para obtener más información sobre la programación con la biblioteca de cifrado del lado del cliente de Java para DynamoDB, consulte los ejemplos de Java, el directorio examples del repositorio - dynamodb-java en. aws-database-encryption-sdk GitHub

Temas

- Encriptadores de elementos
- Acciones de atributos en el SDK de cifrado AWS de bases de datos para DynamoDB
- Configuración de cifrado en el SDK de cifrado de bases de datos de AWS para DynamoDB
- Cifrado con capacidad de búsqueda en DynamoDB

Encriptadores de elementos

En esencia, el SDK de cifrado AWS de bases de datos para DynamoDB es un cifrador de elementos. Puede utilizar la versión 3.x de la biblioteca de cifrado del cliente de Java para DynamoDB para cifrar, firmar, verificar y descifrar los elementos de la tabla de DynamoDB de las siguientes maneras.

El cliente mejorado de DynamoDB

Puede configurar el cliente mejorado de DynamoDB con el `DynamoDbEncryptionInterceptor` para cifrar y firmar automáticamente los elementos del lado del cliente con sus solicitudes `PutItem` de DynamoDB. Con el cliente mejorado de DynamoDB, puede definir las acciones de sus atributos mediante una clase de datos anotada. Recomendamos utilizar el cliente mejorado de DynamoDB siempre que sea posible.

> **Note**
>
> El SDK AWS de cifrado de bases de datos no admite anotaciones en atributos anidados.

API de bajo nivel de DynamoDB

Puede configurar la API de DynamoDB de bajo nivel para cifrar y firmar automáticamente los elementos `DynamoDbEncryptionInterceptor` del lado del cliente con sus solicitudes de DynamoDB. `PutItem`

El nivel inferior **`DynamoDbItemEncryptor`**

El nivel inferior cifra y firma o descifra y verifica `DynamoDbItemEncryptor` directamente los elementos de la tabla sin llamar a DynamoDB. No realiza DynamoDB ni `PutItem` solicitudes`GetItem`. Por ejemplo, puede usar el nivel inferior `DynamoDbItemEncryptor` para descifrar y verificar directamente un elemento de DynamoDB que ya haya recuperado.

El nivel inferior `DynamoDbItemEncryptor` no admite el cifrado con capacidad de búsqueda.

Acciones de atributos en el SDK de cifrado AWS de bases de datos para DynamoDB

Las acciones de atributo determinan qué valores de atributo se cifran y se firman, cuáles solo se firman y cuáles se omiten.

Si utiliza la API de DynamoDB de bajo nivel o el `DynamoDbItemEncryptor` nivel inferior, debe definir manualmente las acciones de los atributos. Si usa el cliente mejorado de DynamoDB, puede definir manualmente las acciones de sus atributos o puede usar una clase de datos anotada para generar un. `TableSchema` Para simplificar el proceso de configuración, se recomienda utilizar una clase de datos anotada. Cuando utiliza una clase de datos anotada, solo tiene que modelar el objeto una vez.

> **Note**
>
> Tras definir las acciones de los atributos, debe definir qué atributos se excluyen de las firmas. Para facilitar la adición de nuevos atributos sin firmar en el futuro, recomendamos elegir un prefijo distinto (como “`:`“) para identificar los atributos sin firmar. Incluya este prefijo en el nombre del atributo para todos los atributos marcados `DO_NOTHING` al definir el esquema y las acciones de atributos de DynamoDB.

Utilice una clase de datos anotada

Utilice una clase de datos anotada para especificar las acciones de sus atributos con el cliente mejorado de DynamoDB y. `DynamoDbEncryptionInterceptor` El SDK de cifrado de bases de datos de AWS para DynamoDB utiliza las anotaciones de atributo estándar de DynamoDB https://sdk.amazonaws.com/java/api/latest/software/amazon/awssdk/enhanced/dynamodb/mapper/annotations/package-summary.htmlque definen el tipo de atributo para determinar cómo proteger un atributo. De forma predeterminada, todos los atributos están cifrados y firmados, excepto las claves principales, que están firmadas, pero no cifradas.

Consulte SimpleClass.java en el repositorio aws-database-encryption-sdk -dynamodb-java GitHub para obtener más información sobre las anotaciones del cliente mejorado de DynamoDB.

De forma predeterminada, los atributos de la clave principal están firmados pero no cifrados (`SIGN_ONLY`) y todos los demás atributos están cifrados y firmados `ENCRYPT_AND_SIGN`(). Para especificar las excepciones, utilice las anotaciones de cifrado que se definen en la biblioteca de cifrado del cliente de Java para DynamoDB. Por ejemplo, si desea que un atributo concreto solo esté firmado, utilice la `@DynamoDbEncryptionSignOnly` anotación. Si desea que un atributo concreto no esté firmado ni cifrado (`DO_NOTHING`), utilice la anotación `@DynamoDbEncryptionDoNothing`.

> **Note**
>
> El SDK de cifrado de AWS bases de datos no admite anotaciones en atributos anidados.

En el siguiente ejemplo, se muestran las anotaciones utilizadas para definir las acciones de los atributos.

```
@DynamoDbBean
public class SimpleClass {

    private String partitionKey;
    private int sortKey;
    private String attribute1;
    private String attribute2;
    private String attribute3;

    @DynamoDbPartitionKey
    @DynamoDbAttribute(value = "partition_key")
    public String getPartitionKey() {
        return this.partitionKey;
```

```
    }

    public void setPartitionKey(String partitionKey) {
        this.partitionKey = partitionKey;
    }

    @DynamoDbSortKey
    @DynamoDbAttribute(value = "sort_key")
    public int getSortKey() {
        return this.sortKey;
    }

    public void setSortKey(int sortKey) {
        this.sortKey = sortKey;
    }

    public String getAttribute1() {
        return this.attribute1;
    }

    public void setAttribute1(String attribute1) {
        this.attribute1 = attribute1;
    }

    @DynamoDbEncryptionSignOnly
    public String getAttribute2() {
        return this.attribute2;
    }

    public void setAttribute2(String attribute2) {
        this.attribute2 = attribute2;
    }

    @DynamoDbEncryptionDoNothing
    public String getAttribute3() {
        return this.attribute3;
    }

    @DynamoDbAttribute(value = ":attribute3")
    public void setAttribute3(String attribute3) {
        this.attribute3 = attribute3;
    }
}
```

Utilice la clase de datos anotada para crearla tal y `TableSchema` como se muestra en el siguiente fragmento.

```
final TableSchema<SimpleClass> tableSchema = TableSchema.fromBean(SimpleClass.class);
```

Defina manualmente las acciones de sus atributos

Para especificar las acciones de atributo cuando se utiliza el DynamoDBEncryptor directamente, cree un objeto `Map` en el que las parejas de nombre-valor representen nombres de atributo y las acciones especificadas.

Especifique `ENCRYPT_AND_SIGN` si desea cifrar y firmar un atributo. Especifique `SIGN_ONLY` firmar, pero no cifrar, un atributo. No se puede cifrar un atributo sin firmarlo también. Especifique `DO_NOTHING` que se omita un atributo.

```
final Map<String, CryptoAction> attributeActionsOnEncrypt = new HashMap<>();
// The partition attribute must be SIGN_ONLY
attributeActionsOnEncrypt.put("partition_key", CryptoAction.SIGN_ONLY);
// The sort attribute must be SIGN_ONLY
attributeActionsOnEncrypt.put("sort_key", CryptoAction.SIGN_ONLY);
attributeActionsOnEncrypt.put("attribute1", CryptoAction.ENCRYPT_AND_SIGN);
attributeActionsOnEncrypt.put("attribute2", CryptoAction.SIGN_ONLY);
attributeActionsOnEncrypt.put(":attribute3", CryptoAction.DO_NOTHING);
```

Configuración de cifrado en el SDK de cifrado de bases de datos de AWS para DynamoDB

Al utilizar el SDK de cifrado AWS de bases de datos, debe definir explícitamente una configuración de cifrado para la tabla de DynamoDB. Los valores necesarios en la configuración de cifrado dependen de si ha definido las acciones de los atributos manualmente o con una clase de datos anotada.

El siguiente fragmento define una configuración de cifrado de tablas de DynamoDB mediante el cliente mejorado de DynamoDB y los atributos no firmados permitidos definidos por un prefijo `TableSchema`distinto.

```
final Map<String, DynamoDbEnhancedTableEncryptionConfig> tableConfigs = new
 HashMap<>();
tableConfigs.put(ddbTableName,
        DynamoDbEnhancedTableEncryptionConfig.builder()
```

```
            .logicalTableName(ddbTableName)
            .keyring(kmsKeyring)
            .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
            .schemaOnEncrypt(tableSchema)
            // Optional: only required if you use beacons
            .search(SearchConfig.builder()
                    .writeVersion(1) // MUST be 1
                    .versions(beaconVersions)
                    .build())
            .build());
```

Nombre de la tabla lógica

Un nombre de tabla lógico para la tabla de DynamoDB.

El nombre de la tabla lógica está enlazado criptográficamente a todos los datos almacenados en la tabla para simplificar las operaciones de restauración de DynamoDB. Se recomienda encarecidamente especificar el nombre de la tabla de DynamoDB como nombre de la tabla lógica cuando defina por primera vez la configuración de cifrado. Debe especificar siempre el mismo nombre de tabla lógica. Para que el descifrado se realice correctamente, el nombre de la tabla lógica debe coincidir con el nombre especificado en el cifrado. En caso de que el nombre de la tabla de DynamoDB cambie después de restaurar la tabla de DynamoDB a partir de una copia de seguridad, el nombre de la tabla lógica garantiza que la operación de descifrado siga reconociendo la tabla.

Atributos sin signo permitidos

Los atributos marcados `DO_NOTHING` en tus acciones de atributos.

Los atributos no firmados permitidos indican al cliente qué atributos están excluidos de las firmas. El cliente asume que todos los demás atributos están incluidos en la firma. A continuación, al descifrar un registro, el cliente determina qué atributos debe verificar y cuáles debe ignorar de los atributos no firmados permitidos que especificó. No puede eliminar un atributo de los atributos no firmados permitidos.

Puede definir los atributos no firmados permitidos de forma explícita mediante la creación de una matriz que enumere todos sus `DO_NOTHING` atributos. También puedes especificar un prefijo distinto al asignar un nombre a tus `DO_NOTHING` atributos y usar el prefijo para indicar al cliente qué atributos no están firmados. Recomendamos encarecidamente especificar un prefijo distinto porque simplifica el proceso de añadir un nuevo `DO_NOTHING` atributo en el futuro. Para obtener más información, consulte Actualización de su modelo de datos.

Si no especifica un prefijo para todos los `DO_NOTHING` atributos, puede configurar una `allowedUnsignedAttributes` matriz que enumere de forma explícita todos los atributos que el cliente debería esperar que no estén firmados cuando los encuentre al descifrarlos. Solo debe definir de forma explícita los atributos no firmados permitidos si es absolutamente necesario.

Configuración de búsqueda (Opcional)

`SearchConfig`Define la versión de baliza.

`SearchConfig`Debe especificarse para utilizar balizas firmadas o cifradas con capacidad de búsqueda.

Conjunto de algoritmos (opcional)

El `algorithmSuiteId` define qué conjunto de algoritmos utiliza el SDK de cifrado de bases de datos de AWS .

A menos que especifique explícitamente un conjunto de algoritmos alternativo, el SDK de cifrado AWS de bases de datos utiliza el conjunto de algoritmos predeterminado. El conjunto de algoritmos predeterminado 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 SDK AWS de cifrado de bases de datos, consulteConjuntos de algoritmos compatibles en el SDK de cifrado de bases de datos de AWS.

Para seleccionar el conjunto de algoritmos AES-GCM sin firmas digitales, incluya el siguiente fragmento en la configuración de cifrado de la tabla.

```
.algorithmSuiteId(

 DBEAlgorithmSuiteId.ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY_ECDSA_P384_SYMSIG_HMAC_SHA384)
```

Cifrado con capacidad de búsqueda en DynamoDB

Para configurar las tablas de Amazon DynamoDB para el cifrado con capacidad de búsqueda, debe utilizar el conjunto de claves deAWS KMS jerárquico para generar, cifrar y descifrar las claves de datos utilizadas para proteger los elementos. Debe usar el cliente mejorado de DynamoDB o la API de DynamoDB de bajo nivel para cifrar, firmar, verificar y descifrar los elementos de la tabla. El nivel

inferior no admite el cifrado con capacidad de búsqueda. `DynamoDBItemEncryptor` También debe incluir `SearchConfig`en la configuración de cifrado de la tabla.

Después de configurar las balizas, debe configurar un índice secundario que refleje cada baliza antes de poder buscar en los atributos cifrados.

Configurar índices secundarios con balizas

Al configurar una baliza estándar o compuesta, el SDK de cifrado de AWS bases de datos añade el `aws_dbe_b_` prefijo al nombre de la baliza para que el servidor pueda identificarlas fácilmente. Por ejemplo, si nombra una baliza compuestacompoundBeacon, el nombre completo de la baliza es realmente. `aws_dbe_b_compoundBeacon` Si desea configurar índices secundarios que incluyan una baliza estándar o compuesta, debe incluir el `aws_dbe_b_` prefijo al identificar el nombre de la baliza.

Claves de partición y claves de clasificación

No puede cifrar los valores de la clave principal. Sus claves de partición y clasificación deben serlo. `SIGN_ONLY` Sus valores de la clave principal no pueden ser una baliza estándar o compuesta.

Sus valores de la clave principal pueden ser balizas firmadas. Si ha configurado balizas firmadas distintas para cada uno de los valores de la clave principal, debe especificar el nombre del atributo que identifica el valor de la clave principal como el nombre de la baliza firmada. Sin embargo, el SDK AWS de cifrado de bases de datos no añade el `aws_dbe_b_` prefijo a las balizas firmadas. Aunque haya configurado balizas firmadas distintas para los valores de la clave principal, solo tendrá que especificar los nombres de los atributos de los valores de la clave principal al configurar un índice secundario.

Índices secundarios locales

La clave de clasificación de un índice secundario local puede ser una baliza.

Si especifica una baliza para la clave de clasificación, el tipo debe ser Cadena. Si especifica una baliza estándar o compuesta para la clave de clasificación, debe incluir el `aws_dbe_b_` prefijo al especificar el nombre de la baliza. Si especifica una baliza firmada, especifique el nombre de la baliza sin ningún prefijo.

Índices secundarios globales

Tanto la partición como las claves de clasificación de un índice secundario global pueden ser balizas.

Si especifica un indicador para la partición o la clave de clasificación, el tipo debe ser Cadena. Si especifica una baliza estándar o compuesta para la clave de clasificación, debe incluir el prefijo `aws_dbe_b_` al especificar el nombre de la baliza. Si especifica una baliza firmada, especifique el nombre de la baliza sin ningún prefijo.

Proyecciones de atributos

Una proyección es el conjunto de atributos que se copia de una tabla en un índice secundario. La clave de partición y la clave de clasificación de la tabla siempre se proyectan en el índice; puede proyectar otros atributos para admitir los requisitos de consulta de la aplicación. DynamoDB ofrece tres opciones diferentes para las proyecciones de atributos`KEYS_ONLY`:`INCLUDE`, y. `ALL`

Si utiliza la proyección de atributos INCLUDE para buscar en una baliza, debe especificar los nombres de todos los atributos a partir de los que se construye la baliza y el nombre de la baliza con el `aws_dbe_b_` prefijo. Por ejemplo, si ha configurado una baliza compuesta`compoundBeacon`, desde `field1``field2`, y`field3`, debe especificar`aws_dbe_b_compoundBeacon`, `field1``field2`, y `field3` en la proyección.

Un índice secundario global solo puede usar los atributos especificados explícitamente en la proyección, pero un índice secundario local puede usar cualquier atributo.

Ejemplos de Java

> Se cambió el nombre de nuestra biblioteca de cifrado del lado del cliente por el de SDK de cifrado de AWS bases de datos. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

Los siguientes ejemplos muestran cómo utilizar la biblioteca de cifrado del cliente de Java para DynamoDB para proteger los elementos de tabla en su aplicación. Puedes encontrar más ejemplos (y añadir los tuyos propios) en el directorio de ejemplos del repositorio aws-database-encryption-sdk - dynamodb-java en. GitHub

Los siguientes ejemplos muestran cómo configurar la biblioteca de cifrado del cliente de Java para DynamoDB en una tabla de Amazon DynamoDB nueva y sin rellenar. Si desea configurar las tablas de Amazon DynamoDB existentes para el cifrado del cliente, consulte. Agregar la versión 3.x a una tabla existente

Temas

- Uso del cliente mejorado de DynamoDB
- API de bajo nivel de DynamoDB
- Usando el nivel inferior DynamoDbItemEncryptor

Uso del cliente mejorado de DynamoDB

En el siguiente ejemplo, se muestra cómo utilizar el cliente mejorado de DynamoDB y el `DynamoDbEncryptionInterceptor` con un conjunto de claves deAWS KMS para cifrar los elementos de la tabla de DynamoDB como parte de las llamadas a la API de DynamoDB.

Puede utilizar cualquier conjunto de claves compatible con el cliente mejorado de DynamoDB, pero le recomendamos que utilice uno de los anillos de AWS KMS claves siempre que sea posible.

Consulte el ejemplo de código completo: .java EnhancedPutGetExample

Paso 1: Crea el llavero AWS KMS

El siguiente ejemplo se utiliza `CreateAwsKmsMrkMultiKeyring` para crear un AWS KMS anillo de claves con una clave KMS de cifrado simétrico. El método `CreateAwsKmsMrkMultiKeyring` garantiza que el conjunto de claves maneje correctamente las claves de una sola región y de múltiples regiones.

```
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
final CreateAwsKmsMrkMultiKeyringInput keyringInput =
 CreateAwsKmsMrkMultiKeyringInput.builder()
        .generator(kmsKeyId)
        .build();
final IKeyring kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

Paso 2: crear un esquema de tabla a partir de la clase de datos anotados

En el siguiente ejemplo, se utiliza la clase de datos anotada para crear la `TableSchema`.

En este ejemplo, se supone que las acciones de clase y atributo de datos anotados se definieron mediante .java. SimpleClass Para obtener más información sobre cómo anotar las acciones de los atributos, consulte Utilice una clase de datos anotada.

> **Note**
>
> El SDK AWS de cifrado de bases de datos no admite anotaciones en atributos anidados.

```
final TableSchema<SimpleClass> schemaOnEncrypt =
 TableSchema.fromBean(SimpleClass.class);
```

Paso 3: defina qué atributos se excluyen de las firmas

En el ejemplo siguiente, se supone que todos los atributos `DO_NOTHING` comparten el prefijo distinto `":"` y se utiliza el prefijo para definir los atributos no firmados permitidos. El cliente asume que cualquier nombre de atributo con el prefijo "`:`" está excluido de las firmas. Para obtener más información, consulte Atributos sin signo permitidos.

```
final String unsignedAttrPrefix = ":";
```

Paso 4: crear el contexto de cifrado

En el siguiente ejemplo, se define un mapa `tableConfigs` que representa la configuración de cifrado de la tabla de DynamoDB.

En este ejemplo, se especifica el nombre de la tabla de DynamoDB como nombre de la tabla lógica. Se recomienda encarecidamente especificar el nombre de la tabla de DynamoDB como nombre de la tabla lógica cuando defina por primera vez la configuración de cifrado. Para obtener más información, consulte Configuración de cifrado en el SDK de cifrado de bases de datos de AWS para DynamoDB.

> **Note**
>
> Para utilizar balizas firmadas o cifrado con capacidad de búsqueda, también debe incluirlos `SearchConfig`en la configuración de cifrado.

```
final Map<String, DynamoDbEnhancedTableEncryptionConfig> tableConfigs = new
 HashMap<>();
tableConfigs.put(ddbTableName,
    DynamoDbEnhancedTableEncryptionConfig.builder()
```

```
        .logicalTableName(ddbTableName)
        .keyring(kmsKeyring)
        .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
        .schemaOnEncrypt(tableSchema)
        .build());
```

Paso 5: Cree la **`DynamoDbEncryptionInterceptor`**

En el siguiente ejemplo, se crea un nuevo `DynamoDbEncryptionInterceptor` con la `tableConfigs` del Paso 4.

```
final DynamoDbEncryptionInterceptor interceptor =
    DynamoDbEnhancedClientEncryption.CreateDynamoDbEncryptionInterceptor(
        CreateDynamoDbEncryptionInterceptorInput.builder()
            .tableEncryptionConfigs(tableConfigs)
            .build()
    );
```

Paso 6: Crear un nuevo cliente AWS SDK de DynamoDB

En el siguiente ejemplo, se crea un nuevo cliente AWS SDK de DynamoDB mediante **`interceptor`** el paso 5.

```
final DynamoDbClient ddb = DynamoDbClient.builder()
        .overrideConfiguration(
                ClientOverrideConfiguration.builder()
                       .addExecutionInterceptor(interceptor)
                       .build())
        .build();
```

Paso 7: Crear el cliente mejorado de DynamoDB y crear una tabla

En el siguiente ejemplo, se crea el cliente mejorado DynamoDB mediante el cliente del SDK DynamoDB de AWS creado en el Paso 6 y se crea una tabla con la clase de datos anotados.

```
final DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
        .dynamoDbClient(ddb)
        .build();
final DynamoDbTable<SimpleClass> table = enhancedClient.table(ddbTableName,
 tableSchema);
```

Paso 8: Cifrar y guardar un elemento de la tabla

En el siguiente ejemplo, se coloca un elemento en la tabla de DynamoDB mediante el cliente mejorado de DynamoDB. El elemento se cifra y se firma en el lado del cliente antes de enviarlo a DynamoDB.

```
final SimpleClass item = new SimpleClass();
item.setPartitionKey("EnhancedPutGetExample");
item.setSortKey(0);
item.setAttribute1("encrypt and sign me!");
item.setAttribute2("sign me!");
item.setAttribute3("ignore me!");

table.putItem(item);
```

API de bajo nivel de DynamoDB

En el siguiente ejemplo, se muestra cómo utilizar la API de DynamoDB de bajo nivel con un conjunto de claves deAWS KMS para cifrar y firmar automáticamente los elementos del lado del cliente con las solicitudes de DynamoDB de `PutItem`.

Puede utilizar cualquier llavero compatible, pero le recomendamos que utilice uno de los AWS KMS llaveros siempre que sea posible.

Consulta el ejemplo de código completo: .java BasicPutGetExample

Paso 1: Crea el llavero AWS KMS

El siguiente ejemplo se utiliza `CreateAwsKmsMrkMultiKeyring` para crear un AWS KMS anillo de claves con una clave KMS de cifrado simétrico. El método `CreateAwsKmsMrkMultiKeyring` garantiza que el conjunto de claves maneje correctamente las claves de una sola región y de múltiples regiones.

```
final MaterialProviders matProv = MaterialProviders.builder()
         .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
         .build();
final CreateAwsKmsMrkMultiKeyringInput keyringInput =
 CreateAwsKmsMrkMultiKeyringInput.builder()
        .generator(kmsKeyId)
        .build();
```

```
final IKeyring kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

Paso 2: configurar las acciones de sus atributos

En el siguiente ejemplo, se define un mapa `attributeActionsOnEncrypt` que representa ejemplos de acciones de atributos para un elemento de la tabla.

```
final Map<String, CryptoAction> attributeActionsOnEncrypt = new HashMap<>();
// The partition attribute must be SIGN_ONLY
attributeActionsOnEncrypt.put("partition_key", CryptoAction.SIGN_ONLY);
// The sort attribute must be SIGN_ONLY
attributeActionsOnEncrypt.put("sort_key", CryptoAction.SIGN_ONLY);
attributeActionsOnEncrypt.put("attribute1", CryptoAction.ENCRYPT_AND_SIGN);
attributeActionsOnEncrypt.put("attribute2", CryptoAction.SIGN_ONLY);
attributeActionsOnEncrypt.put(":attribute3", CryptoAction.DO_NOTHING);
```

Paso 3: defina qué atributos se excluyen de las firmas

En el ejemplo siguiente, se supone que todos los atributos `DO_NOTHING` comparten el prefijo distinto `":"` y se utiliza el prefijo para definir los atributos no firmados permitidos. El cliente asume que cualquier nombre de atributo con el prefijo “`:`” está excluido de las firmas. Para obtener más información, consulte Atributos sin signo permitidos.

```
final String unsignedAttrPrefix = ":";
```

Paso 4: definir la configuración de cifrado de la tabla de DynamoDB

El siguiente ejemplo define un mapa `tableConfigs` que representa la configuración de cifrado de esta tabla de DynamoDB.

En este ejemplo, se especifica el nombre de la tabla de DynamoDB como nombre de la tabla lógica. Se recomienda encarecidamente especificar el nombre de la tabla de DynamoDB como nombre de la tabla lógica cuando defina por primera vez la configuración de cifrado. Para obtener más información, consulte Configuración de cifrado en el SDK de cifrado de bases de datos de AWS para DynamoDB.

> **Note**
>
> Para utilizar balizas firmadas o cifrado con capacidad de búsqueda, también debe incluir `SearchConfig` en la configuración de cifrado.

```
final Map<String, DynamoDbTableEncryptionConfig> tableConfigs = new HashMap<>();
final DynamoDbTableEncryptionConfig config = DynamoDbTableEncryptionConfig.builder()
        .logicalTableName(ddbTableName)
        .partitionKeyName("partition_key")
        .sortKeyName("sort_key")
        .attributeActionsOnEncrypt(attributeActionsOnEncrypt)
        .keyring(kmsKeyring)
        .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
        .build();
tableConfigs.put(ddbTableName, config);
```

Paso 5: Crear el **`DynamoDbEncryptionInterceptor`**

En el siguiente ejemplo, se crea el `DynamoDbEncryptionInterceptor` con la `tableConfigs` del Paso 4.

```
DynamoDbEncryptionInterceptor interceptor = DynamoDbEncryptionInterceptor.builder()
        .config(DynamoDbTablesEncryptionConfig.builder()
                .tableEncryptionConfigs(tableConfigs)
                .build())
        .build();
```

Paso 6: Crear un nuevo cliente AWS SDK de DynamoDB

En el siguiente ejemplo, se crea un nuevo cliente AWS SDK de DynamoDB mediante **`interceptor`** el paso 5.

```
final DynamoDbClient ddb = DynamoDbClient.builder()
        .overrideConfiguration(
                ClientOverrideConfiguration.builder()
                       .addExecutionInterceptor(interceptor)
                       .build())
        .build();
```

Paso 7: Cifrar y firmar un elemento de la tabla de DynamoDB

En el siguiente ejemplo, se define un mapa `item` que representa un elemento de tabla de ejemplo y se coloca el elemento en la tabla de DynamoDB. El elemento se cifra y se firma en el lado del cliente antes de enviarlo a DynamoDB.

```
final HashMap<String, AttributeValue> item = new HashMap<>();
```

```
item.put("partition_key", AttributeValue.builder().s("BasicPutGetExample").build());
item.put("sort_key", AttributeValue.builder().n("0").build());
item.put("attribute1", AttributeValue.builder().s("encrypt and sign me!").build());
item.put("attribute2", AttributeValue.builder().s("sign me!").build());
item.put(":attribute3", AttributeValue.builder().s("ignore me!").build());

final PutItemRequest putRequest = PutItemRequest.builder()
        .tableName(ddbTableName)
        .item(item)
        .build();

final PutItemResponse putResponse = ddb.putItem(putRequest);
```

Usando el nivel inferior DynamoDbItemEncryptor

En el siguiente ejemplo, se muestra cómo utilizar el nivel inferior `DynamoDbItemEncryptor` con un conjunto de claves deAWS KMS para cifrar y firmar directamente los elementos de la tabla. `DynamoDbItemEncryptor` No coloca el elemento en la tabla de DynamoDB.

Puede utilizar cualquier conjunto de claves compatible con el cliente mejorado de DynamoDB, pero le recomendamos que utilice uno de los anillos de AWS KMS claves siempre que sea posible.

> **Note**
>
> El nivel inferior `DynamoDbItemEncryptor` no admite el cifrado con capacidad de búsqueda. Úselo `DynamoDbEncryptionInterceptor` con la API de DynamoDB de bajo nivel o con el cliente mejorado de DynamoDB para utilizar el cifrado con capacidad de búsqueda.

Consulte el ejemplo de código completo: .java ItemEncryptDecryptExample

Paso 1: Crea el llavero AWS KMS

El siguiente ejemplo se utiliza `CreateAwsKmsMrkMultiKeyring` para crear un AWS KMS anillo de claves con una clave KMS de cifrado simétrico. El método `CreateAwsKmsMrkMultiKeyring` garantiza que el conjunto de claves maneje correctamente las claves de una sola región y de múltiples regiones.

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

```
         .build();
final CreateAwsKmsMrkMultiKeyringInput keyringInput =
 CreateAwsKmsMrkMultiKeyringInput.builder()
        .generator(kmsKeyId)
        .build();
final IKeyring kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

Paso 2: configurar las acciones de sus atributos

En el siguiente ejemplo, se define un mapa `attributeActionsOnEncrypt` que representa ejemplos de acciones de atributos para un elemento de la tabla.

```
final Map<String, CryptoAction> attributeActionsOnEncrypt = new HashMap<>();
// The partition attribute must be SIGN_ONLY
attributeActionsOnEncrypt.put("partition_key", CryptoAction.SIGN_ONLY);
// The sort attribute must be SIGN_ONLY
attributeActionsOnEncrypt.put("sort_key", CryptoAction.SIGN_ONLY);
attributeActionsOnEncrypt.put("attribute1", CryptoAction.ENCRYPT_AND_SIGN);
attributeActionsOnEncrypt.put("attribute2", CryptoAction.SIGN_ONLY);
attributeActionsOnEncrypt.put(":attribute3", CryptoAction.DO_NOTHING);
```

Paso 3: defina qué atributos se excluyen de las firmas

En el ejemplo siguiente, se supone que todos los atributos `DO_NOTHING` comparten el prefijo distinto **":"** y se utiliza el prefijo para definir los atributos no firmados permitidos. El cliente asume que cualquier nombre de atributo con el prefijo “**:**” está excluido de las firmas. Para obtener más información, consulte Atributos sin signo permitidos.

```
final String unsignedAttrPrefix = ":";
```

Paso 4: defina la configuración de **`DynamoDbItemEncryptor`**

En el siguiente ejemplo, se consulta la configuración de `DynamoDbItemEncryptor`.

En este ejemplo, se especifica el nombre de la tabla de DynamoDB como nombre de la tabla lógica. Se recomienda encarecidamente especificar el nombre de la tabla de DynamoDB como nombre de la tabla lógica cuando defina por primera vez la configuración de cifrado. Para obtener más información, consulte Configuración de cifrado en el SDK de cifrado de bases de datos de AWS para DynamoDB.

```
final DynamoDbItemEncryptorConfig config = DynamoDbItemEncryptorConfig.builder()
```

```
        .logicalTableName(ddbTableName)
        .partitionKeyName("partition_key")
        .sortKeyName("sort_key")
        .attributeActionsOnEncrypt(attributeActionsOnEncrypt)
        .keyring(kmsKeyring)
        .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
        .build();
```

Paso 5: Crear el **`DynamoDbItemEncryptor`**

En el siguiente ejemplo, se crea un nuevo `DynamoDbItemEncryptor`, con la `config` del Paso 4.

```
final DynamoDbItemEncryptor itemEncryptor = DynamoDbItemEncryptor.builder()
        .DynamoDbItemEncryptorConfig(config)
        .build();
```

Paso 6: cifrar y firmar directamente un elemento de la tabla

En el siguiente ejemplo, se cifra y firma directamente un elemento mediante el `DynamoDbItemEncryptor`. `DynamoDbItemEncryptor` no coloca el elemento en la tabla de DynamoDB.

```
final Map<String, AttributeValue> originalItem = new HashMap<>();
originalItem.put("partition_key",
 AttributeValue.builder().s("ItemEncryptDecryptExample").build());
originalItem.put("sort_key", AttributeValue.builder().n("0").build());
originalItem.put("attribute1", AttributeValue.builder().s("encrypt and sign
 me!").build());
originalItem.put("attribute2", AttributeValue.builder().s("sign me!").build());
originalItem.put(":attribute3", AttributeValue.builder().s("ignore me!").build());

final Map<String, AttributeValue> encryptedItem = itemEncryptor.EncryptItem(
        EncryptItemInput.builder()
                .plaintextItem(originalItem)
                .build()
).encryptedItem();
```

Actualización de su modelo de datos

Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

Al configurar la biblioteca de cifrado del cliente de Java para DynamoDB, proporciona acciones de atributos. Al cifrar, el SDK de cifrado de bases de datos de AWS utiliza las acciones de atributos para identificar qué atributos cifrar y firmar, qué atributos firmar (pero no cifrar) y cuáles ignorar. También se definen los atributos no firmados permitidos para indicar explícitamente al cliente qué atributos están excluidos de las firmas. Al descifrar, el SDK de cifrado de bases de datos de AWS utiliza los atributos no firmados permitidos que usted definió para identificar qué atributos no están incluidos en las firmas. Las acciones de atributo no se guardan en el elemento cifrado y el SDK de cifrado de bases de datos de AWS no actualiza automáticamente las acciones de atributo.

Elija cuidadosamente sus acciones de atributo. En caso de duda, use Encrypt and sign. Una vez que haya utilizado el SDK de cifrado de bases de datos de AWS para proteger sus elementos, no podrá cambiar un atributo `ENCRYPT_AND_SIGN` o `SIGN_ONLY` existente a `DO_NOTHING`. Puede hacer los siguientes cambios.

- Agregue atributos y nuevos `ENCRYPT_AND_SIGN``SIGN_ONLY`
- Elimine `ENCRYPT_AND_SIGN` los `SIGN_ONLY``DO_NOTHING` atributos existentes
- Cambie un `ENCRYPT_AND_SIGN` atributo existente a `SIGN_ONLY`
- Cambie un atributo existente `SIGN_ONLY` a `ENCRYPT_AND_SIGN`
- Añada un atributo `DO_NOTHING` nuevo

Consideraciones sobre el cifrado con capacidad de búsqueda

Antes de actualizar el modelo de datos, considere detenidamente cómo podrían afectar las actualizaciones a las balizas que haya creado a partir de los atributos. Una vez que haya escrito nuevos registros con una baliza, no puede actualizar la configuración de la baliza. No puede actualizar las acciones de los atributos asociadas a los atributos que utilizó para construir balizas. Si elimina un atributo existente y su baliza asociada, no podrá consultar los registros existentes con esa baliza. Puede crear balizas nuevas para los campos nuevos que añada a su registro, pero no puede actualizar las balizas existentes para incluir el nuevo campo.

Agregue atributos y nuevos **ENCRYPT_AND_SIGNSIGN_ONLY**

Para agregar un nuevo atributo `ENCRYPT_AND_SIGN` o un atributo`SIGN_ONLY`, defina el nuevo atributo en las acciones de sus atributos.

No puede eliminar un atributo `DO_NOTHING` existente y volver a añadirlo como un atributo `ENCRYPT_AND_SIGN` o `SIGN_ONLY`.

Uso de una clase de datos anotada

Si ha definido las acciones de los atributos con una`TableSchema`, añada el nuevo atributo a la clase de datos anotada. Si no especificas una anotación de acción de atributo para el nuevo atributo, el cliente cifrará y firmará el nuevo atributo de forma predeterminada (a menos que el atributo forme parte de la clave principal). Si solo quiere firmar el nuevo atributo, debe añadir el nuevo atributo con la `@DynamoDBEncryptionSignOnly` anotación.

Uso de un objeto de modelo

Si ha definido manualmente las acciones de los atributos, añada el nuevo atributo a las acciones de los atributos del modelo de objetos y especifique `ENCRYPT_AND_SIGN` o `SIGN_ONLY` como la acción de atributo.

Elimine **ENCRYPT_AND_SIGN** los **SIGN_ONLYDO_NOTHING** atributos existentes

Si decide que ya no necesita un atributo, puede dejar de escribir datos en ese atributo o puede eliminarlos formalmente de las acciones de sus atributos. Cuando dejas de escribir nuevos datos en un atributo, el atributo sigue apareciendo en las acciones de tus atributos. Esto puede resultar útil si necesita volver a utilizar el atributo en el futuro. Si eliminas formalmente el atributo de tus acciones de atributos, no lo eliminas de tu conjunto de datos. Su conjunto de datos seguirá conteniendo elementos que incluyan ese atributo.

Para eliminar formalmente un atributo existente `ENCRYPT_AND_SIGN`, `SIGN_ONLY` o `DO_NOTHING` actualice las acciones de sus atributos.

Si elimina un atributo `DO_NOTHING`, no debe eliminarlo de los atributos no firmados permitidos. Aunque ya no escriba valores nuevos en ese atributo, el cliente necesitará saber que el atributo no está firmado para poder leer los elementos existentes que lo contienen.

Uso de una clase de datos anotada

Si ha definido las acciones de los atributos con una `TableSchema`, elimine el atributo de la clase de datos anotada.

Uso de un objeto de modelo

Si definió manualmente las acciones de los atributos, elimine el atributo de las acciones de los atributos del modelo de objetos.

Cambie un **ENCRYPT_AND_SIGN** atributo existente a **SIGN_ONLY**

Para cambiar un atributo existente `ENCRYPT_AND_SIGN` a `SIGN_ONLY`, debe actualizar las acciones del atributo. Tras implementar la actualización, el cliente podrá verificar y descifrar los valores existentes escritos en el atributo, pero solo firmará los nuevos valores escritos en el atributo.

Uso de una clase de datos anotada

Si ha definido las acciones de los atributos con una `TableSchema`, actualice el atributo existente para incluir la anotación `@DynamoDBEncryptionSignOnly` en la clase de datos anotada.

Uso de un objeto de modelo

Si ha definido manualmente las acciones de atributo, actualice la acción de atributo asociada al atributo existente de `ENCRYPT_AND_SIGN` a `SIGN_ONLY` en su modelo de objetos.

Cambie un atributo existente **SIGN_ONLY** a **ENCRYPT_AND_SIGN**

Para cambiar un atributo existente `SIGN_ONLY` a `ENCRYPT_AND_SIGN`, debe actualizar las acciones del atributo. Tras implementar la actualización, el cliente podrá comprobar los valores existentes escritos en el atributo y cifrará y firmará los nuevos valores escritos en el atributo.

Uso de una clase de datos anotada

Si ha definido las acciones de sus atributos con una `TableSchema`, elimine la anotación `@DynamoDBEncryptionSignOnly` del atributo existente `SIGN_ONLY`.

Uso de un objeto de modelo

Si ha definido manualmente las acciones de atributo, actualice la acción de atributo asociada al atributo de `SIGN_ONLY` a `ENCRYPT_AND_SIGN` en su modelo de objetos.

Añada un atributo **DO_NOTHING** nuevo

Para reducir el riesgo de errores al añadir un atributo `DO_NOTHING` nuevo, le recomendamos que especifique un prefijo distinto al asignar un nombre a los atributos `DO_NOTHING` y, a continuación, utilizar ese prefijo para definir los atributos no [firmados permitidos].

No puede eliminar un atributo `ENCRYPT_AND_SIGN` o un atributo `SIGN_ONLY` existente de la clase de datos anotada y, a continuación, volver a añadir el atributo como un atributo `DO_NOTHING`. Solo puede agregar atributos `DO_NOTHING` completamente nuevos.

Los pasos que siga para añadir un atributo `DO_NOTHING` nuevo dependerán de si ha definido los atributos no firmados permitidos de forma explícita en una lista o con un prefijo.

Utilizar un prefijo de atributos no firmados permitido

Si ha definido las acciones de los atributos con un `TableSchema`, añada el atributo `DO_NOTHING` nuevo a la clase de datos anotada con la anotación `@DynamoDBEncryptionDoNothing`. Si ha definido manualmente las acciones de los atributos, actualice las acciones de los atributos para incluir el nuevo atributo. Asegúrese de configurar explícitamente el nuevo atributo con la acción de atributo `DO_NOTHING`. Debe incluir el mismo prefijo distinto en el nombre del nuevo atributo.

Utilizar una lista de atributos no firmados permitidos

1. Añada el atributo `DO_NOTHING` nuevo a la lista de atributos no firmados permitidos e implemente la lista actualizada.
2. Implemente el cambio desde el paso 1.

   No puede continuar con el paso 3 hasta que el cambio se haya propagado a todos los hosts que necesiten leer estos datos.
3. Añada el atributo `DO_NOTHING` nuevo a las acciones de sus atributos.

   a. Si ha definido las acciones de los atributos con un `TableSchema`, añada el atributo `DO_NOTHING` nuevo a la clase de datos anotada con la anotación `@DynamoDBEncryptionDoNothing`.

   b. Si ha definido manualmente las acciones de los atributos, actualice las acciones de los atributos para incluir el nuevo atributo. Asegúrese de configurar explícitamente el nuevo atributo con la acción de atributo `DO_NOTHING`.
4. Implemente el cambio desde el paso 3.

Configurar una tabla de DynamoDB existente para usar el SDK de cifrado de bases de datos de AWS para DynamoDB

Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

Con la versión 3. x de la biblioteca de cifrado del cliente de Java para DynamoDB, puede configurar las tablas de Amazon DynamoDB existentes para el cifrado del cliente. En este tema se proporcionan instrucciones sobre los tres pasos que debe seguir para añadir la versión 3. x a una tabla de DynamoDB existente y rellenada.

Requisitos previos

Versión 3. x de la biblioteca de cifrado del cliente de Java para DynamoDB requiere el cliente mejorado de DynamoDB incluido en. AWS SDK for Java 2.x Si aun usa DynamoDBMapper, debe migrar para AWS SDK for Java 2.x usar el cliente mejorado de DynamoDB.

Siga las instrucciones para migrar de la versión 1.x a la 2.x de. AWS SDK for Java

A continuación, siga las instrucciones para empezar a utilizar la API de cliente mejorada de DynamoDB.

Antes de configurar la tabla para que utilice la biblioteca de cifrado del cliente de Java para DynamoDB, debe generar una `TableSchema`mediante una clase de datos anotada y crear un cliente mejorado.

Paso 1: prepararse para leer y escribir elementos cifrados

Complete los siguientes pasos para preparar su cliente del SDK de cifrado de bases de datos de AWS para leer y escribir elementos cifrados. Tras implementar los siguientes cambios, el cliente seguirá leyendo y escribiendo elementos de texto no cifrado. No cifrará ni firmará ningún elemento nuevo escrito en la tabla, pero podrá descifrar los elementos cifrados en cuanto aparezcan. Estos cambios preparan al cliente para empezar a cifrar nuevos elementos. Los siguientes cambios deben implementarse en cada lector antes de continuar con el siguiente paso.

1. Defina las acciones de sus atributos

   Actualice la clase de datos anotada para incluir acciones de atributos que definan qué valores de atributo se cifrarán y firmarán, cuáles solo se firmarán y cuáles se ignorarán.

   Consulte el archivo SimpleClass.java en el repositorio aws-database-encryption-sdk-dynamodb-java de GitHub para obtener más información sobre las anotaciones del cliente mejorado de DynamoDB.

   De forma predeterminada, los atributos de la clave principal están firmados pero no cifrados () y todos los demás atributos están cifrados y firmados (). `SIGN_ONLY` `ENCRYPT_AND_SIGN` Para especificar las excepciones, utiliza las anotaciones de cifrado que se definen en la biblioteca de cifrado del cliente de Java para DynamoDB. Por ejemplo, si desea que un atributo concreto sea de solo firmar, utilice únicamente la anotación `@DynamoDbEncryptionSignOnly`. Si desea que un atributo concreto no esté firmado ni cifrado (`DO_NOTHING`), utilice la anotación `@DynamoDbEncryptionDoNothing`.

   Para ver anotaciones de ejemplo, consulte Utilice una clase de datos anotada.

2. Defina qué atributos se excluirán de las firmas

   En el ejemplo siguiente se supone que todos los atributos `DO_NOTHING` comparten el prefijo distinto “`:`“ y se utiliza el prefijo para definir los atributos no firmados permitidos. El cliente asumirá que cualquier nombre de atributo con el prefijo "`:`" está excluido de las firmas. Para obtener más información, consulte Atributos sin signo permitidos.

   ```
   final String unsignedAttrPrefix = ":";
   ```

3. Cree un llavero

   El siguiente ejemplo crea un AWS KMS llavero. El llavero AWS KMS utiliza un cifrado simétrico o un RSA asimétrico AWS KMS keys para generar, cifrar y descifrar las claves de datos.

   En este ejemplo, se utiliza `CreateMrkMultiKeyring` para crear un llavero AWS KMS con una clave de KMS de cifrado simétrico. El método `CreateAwsKmsMrkMultiKeyring` garantiza que el llavero maneje correctamente las claves de una sola región y de múltiples regiones.

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

```
final CreateAwsKmsMrkMultiKeyringInput keyringInput =
 CreateAwsKmsMrkMultiKeyringInput.builder()
        .generator(kmsKeyId)
        .build();
final IKeyring kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

4. Definir la configuración de cifrado de la tabla de DynamoDB

   El siguiente ejemplo define un `tableConfigs` mapa que representa la configuración de cifrado de esta tabla de DynamoDB.

   En este ejemplo, se especifica el nombre de la tabla de DynamoDB como nombre de la tabla lógica???. Se recomienda encarecidamente especificar el nombre de la tabla de DynamoDB como nombre de la tabla lógica cuando defina por primera vez la configuración de cifrado. Para obtener más información, consulte Configuración de cifrado en el SDK de cifrado de bases de datos de AWS para DynamoDB.

```
final Map<String, DynamoDbTableEncryptionConfig> tableConfigs = new HashMap<>();
final DynamoDbTableEncryptionConfig config = DynamoDbTableEncryptionConfig.builder()
        .logicalTableName(ddbTableName)
        .partitionKeyName("partition_key")
        .sortKeyName("sort_key")
        .schemaOnEncrypt(tableSchema)
        .keyring(kmsKeyring)
        .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
        .build();
tableConfigs.put(ddbTableName, config);
```

5. Crear el **`DynamoDbEncryptionInterceptor`**

   En el siguiente ejemplo, se crea un nuevo `DynamoDbEncryptionInterceptor` con la `tableConfigs` del Paso 3. Debe especificarlo `FORCE_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT` como modificación de texto no cifrado. Esta política sigue leyendo y escribiendo elementos de texto no cifrado, lee los elementos cifrados y prepara al cliente para escribir elementos cifrados.

```
DynamoDbEncryptionInterceptor interceptor = DynamoDbEncryptionInterceptor.builder()
        .config(DynamoDbTablesEncryptionConfig.builder()
                .tableEncryptionConfigs(tableConfigs)

 .plaintextOverride(PlaintextOverride.FORCE_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT)
                .build())
```

```
        .build();
```

Paso 2: escribir elementos cifrados y firmados

Actualice la política de texto no cifrado de su `DynamoDbEncryptionInterceptor` configuración para permitir que el cliente escriba elementos cifrados y firmados. Tras implementar el siguiente cambio, el cliente cifrará y firmará los nuevos elementos en función de las acciones de atributos que configuró en el paso 1. El cliente podrá leer los elementos en texto no cifrado y los elementos cifrados y firmados.

Antes de continuar con el Paso 3, debe cifrar y firmar todos los elementos de texto no cifrado existentes en la tabla. No existe una métrica o consulta única que pueda ejecutar para cifrar rápidamente los elementos de texto no cifrado existentes. Utilice el proceso que mejor se adapte a su sistema. Por ejemplo, puede utilizar un proceso asíncrono que escanee lentamente la tabla y reescriba los elementos mediante las acciones de los atributos y la configuración de cifrado que haya definido. Para identificar los elementos de texto no cifrado de la tabla, se recomienda buscar todos los elementos que no contengan los atributos `aws_dbe_head` y `aws_dbe_foot` que el SDK de cifrado de bases de datos de AWS agrega a los elementos cuando están cifrados y firmados.

En el siguiente ejemplo, se crea el `DynamoDbEncryptionInterceptor` con la misma `tableConfigs` del Paso 1. Debe actualizar la anulación de texto no cifrado con `FORBID_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT`. Esta política sigue leyendo los elementos de texto no cifrado, pero también lee y escribe los elementos cifrados.

```
DynamoDbEncryptionInterceptor interceptor = DynamoDbEncryptionInterceptor.builder()
        .config(DynamoDbTablesEncryptionConfig.builder()
                .tableEncryptionConfigs(tableConfigs)

 .plaintextOverride(PlaintextOverride.FORBID_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT)
                .build())
        .build();
```

Paso 3: Lee solo los elementos cifrados y firmados

Una vez cifrados y firmados todos los elementos, actualice la modificación del texto no cifrado de la `DynamoDbEncryptionInterceptor` configuración para que el cliente solo pueda leer y escribir los elementos cifrados y firmados. Tras implementar el siguiente cambio, el cliente cifrará y firmará los nuevos elementos en función de las acciones de atributos que configuró en el paso 1. El cliente solo podrá leer los elementos cifrados y firmados.

En el siguiente ejemplo, se crea el `DynamoDbEncryptionInterceptor` con la misma `tableConfigs` del Paso 1. Puede actualizar la anulación de texto no cifrado con `FORBID_WRITE_PLAINTEXT_FORBID_READ_PLAINTEXT` o eliminar la política de texto no cifrado de su configuración. De forma predeterminada, el cliente solo lee y escribe los elementos cifrados y firmados.

```
DynamoDbEncryptionInterceptor interceptor = DynamoDbEncryptionInterceptor.builder()
        .config(DynamoDbTablesEncryptionConfig.builder()
                .tableEncryptionConfigs(tableConfigs)
                 // Optional: you can also remove the plaintext policy from your
 configuration

 .plaintextOverride(PlaintextOverride.FORBID_WRITE_PLAINTEXT_FORBID_READ_PLAINTEXT)
                .build())
        .build();
```

Migre a la versión 3.x de la biblioteca de cifrado del cliente de Java para DynamoDB

> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

Versión 3. x de la biblioteca de cifrado del cliente de Java para DynamoDB es una importante reescritura de la 2. base de código x. Incluye numerosas actualizaciones, como un nuevo formato de datos estructurados, una compatibilidad mejorada de multitenencia, cambios de esquema fluidos y compatibilidad con el cifrado para búsquedas. En este tema se proporciona orientación sobre cómo migrar el código a la versión 3. x.

Migración de la versión 1.x a la versión 2.x

Migre a la versión 2. x antes de migrar a la versión 3. x. Versión 2. x cambió el símbolo del proveedor más reciente de `MostRecentProvider` a`CachingMostRecentProvider`. Si actualmente usa la versión 1. x de la biblioteca de cifrado del cliente de Java para DynamoDB con el `MostRecentProvider` símbolo, debe actualizar el nombre del símbolo en el código a. `CachingMostRecentProvider` Para obtener más información, consulte Actualizaciones del proveedor más reciente.

Migración de la versión 2.x a la versión 3.x

En los siguientes procedimientos se describe cómo migrar el código desde la versión 2. x a la versión 3. x de la biblioteca de cifrado del cliente de Java para DynamoDB.

Paso 1. Prepárese para leer los elementos en el nuevo formato

Complete los siguientes pasos para preparar su cliente del SDK AWS de cifrado de bases de datos para leer los elementos en el nuevo formato. Tras implementar los siguientes cambios, el cliente seguirá comportándose del mismo modo que en la versión 2. x. Su cliente seguirá leyendo y escribiendo los elementos de la versión 2. formato x, pero estos cambios preparan al cliente para leer los elementos en el nuevo formato.

Actualice su versión AWS SDK for Java a la 2.x

Versión 3. x de la biblioteca de cifrado del cliente de Java para DynamoDB requiere el cliente mejorado de DynamoDB. El cliente mejorado de DynamoDB reemplaza al DynamoDBMapper utilizado en versiones anteriores. Para usar el cliente mejorado, debe usar el. AWS SDK for Java 2.x

Siga las instrucciones para migrar de la versión 1.x a la 2.x de. AWS SDK for Java

Para obtener más información sobre los AWS SDK for Java 2.x módulos necesarios, consulte. Requisitos previos

Configure su cliente para que lea los elementos cifrados por las versiones antiguas

Los siguientes procedimientos proporcionan una descripción general de los pasos que se muestran en el siguiente ejemplo de código.

1. Cree un llavero.

   Los administradores de llaveros y materiales criptográficos sustituyen a los proveedores de materiales criptográficos utilizados en las versiones anteriores de la biblioteca de cifrado del cliente de Java para DynamoDB.

   > **Important**
   >
   > Las claves de empaquetado que especifique al crear un conjunto de claves deben ser las mismas claves de empaquetado que utilizó con su proveedor de materiales criptográficos en la versión 2. x.

2. Crea un esquema de tabla sobre tu clase anotada.

   En este paso se definen las acciones de atributos que se utilizarán cuando comience a escribir elementos en el nuevo formato.

   Para obtener instrucciones sobre el uso del nuevo cliente mejorado de DynamoDB, consulte Generar un `TableSchema` en la Guía para desarrolladores de AWS SDK for Java.

   En el siguiente ejemplo, se supone que actualizó la clase anotada a partir de la versión 2. x utilizando las nuevas anotaciones de acciones de atributos. Para obtener más información sobre cómo anotar las acciones de los atributos, consulte. Utilice una clase de datos anotada
3. Defina qué atributos se excluyen de la firma.
4. Configure un mapa explícito de las acciones de los atributos configuradas en su clase modelada de la versión 2.x.

   En este paso, se definen las acciones de atributo utilizadas para escribir los elementos en el formato anterior.
5. Configure el `DynamoDBEncryptor` que utilizó en la versión 2. x de la biblioteca de cifrado del cliente de Java para DynamoDB.
6. Configure el comportamiento heredado.
7. Crear una `DynamoDbEncryptionInterceptor`.
8. Crear un cliente del SDK DynamoDB de AWS.
9. Cree la clase modelada `DynamoDBEnhancedClient` y cree una tabla con ella.

   Para obtener más información sobre el cliente mejorado de DynamoDB, consulte crear un cliente mejorado.

```
public class MigrationExampleStep1 {

    public static void MigrationStep1(String kmsKeyId, String ddbTableName, int
 sortReadValue) {
        // 1. Create a Keyring.
        //    This example creates an AWS KMS Keyring that specifies the
        //    same kmsKeyId previously used in the version 2.x configuration.
        //    It uses the 'CreateMrkMultiKeyring' method to create the
        //    keyring, so that the keyring can correctly handle both single
        //    region and Multi-Region KMS Keys.
        //    Note that this example uses the AWS SDK for Java v2 KMS client.
```

```
        final MaterialProviders matProv = MaterialProviders.builder()
                .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
                .build();
        final CreateAwsKmsMrkMultiKeyringInput keyringInput =
CreateAwsKmsMrkMultiKeyringInput.builder()
                .generator(kmsKeyId)
                .build();
        final IKeyring kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);

        // 2. Create a Table Schema over your annotated class.
        //    For guidance on using the new attribute actions
        //    annotations, see SimpleClass.java in the
        //    aws-database-encryption-sdk-dynamodb-java GitHub repository.
        //    All primary key attributes must be signed but not encrypted
        //    (SIGN_ONLY) and by default all non-primary key attributes
        //    are encrypted and signed (ENCRYPT_AND_SIGN).
        //    If you want a particular non-primary key attribute to be signed but
        //    not encrypted, use the 'DynamoDbEncryptionSignOnly' annotation.
        //    If you want a particular attribute to be neither signed nor encrypted
        //    (DO_NOTHING), use the 'DynamoDbEncryptionDoNothing' annotation.
        final TableSchema<SimpleClass> schemaOnEncrypt =
TableSchema.fromBean(SimpleClass.class);

        // 3. Define which attributes the client should expect to be excluded
        //    from the signature when reading items.
        //    This value represents all unsigned attributes across the entire
        //    dataset.
        final List<String> allowedUnsignedAttributes = Arrays.asList("attribute3");

        // 4. Configure an explicit map of the attribute actions configured
        //    in your version 2.x modeled class.
        final Map<String, CryptoAction> legacyActions = new HashMap<>();
        legacyActions.put("partition_key", CryptoAction.SIGN_ONLY);
        legacyActions.put("sort_key", CryptoAction.SIGN_ONLY);
        legacyActions.put("attribute1", CryptoAction.ENCRYPT_AND_SIGN);
        legacyActions.put("attribute2", CryptoAction.SIGN_ONLY);
        legacyActions.put("attribute3", CryptoAction.DO_NOTHING);

        // 5. Configure the DynamoDBEncryptor that you used in version 2.x.
        final AWSKMS kmsClient = AWSKMSClientBuilder.defaultClient();
        final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kmsClient,
kmsKeyId);
        final DynamoDBEncryptor oldEncryptor = DynamoDBEncryptor.getInstance(cmp);
```

```
        // 6. Configure the legacy behavior.
        //    Input the DynamoDBEncryptor and attribute actions created in
        //    the previous steps. For Legacy Policy, use
        //    'FORCE_LEGACY_ENCRYPT_ALLOW_LEGACY_DECRYPT'. This policy continues to
read
        //    and write items using the old format, but will be able to read
        //    items written in the new format as soon as they appear.
        final LegacyOverride legacyOverride = LegacyOverride
                .builder()
                .encryptor(oldEncryptor)
                .policy(LegacyPolicy.FORCE_LEGACY_ENCRYPT_ALLOW_LEGACY_DECRYPT)
                .attributeActionsOnEncrypt(legacyActions)
                .build();

        // 7. Create a DynamoDbEncryptionInterceptor with the above configuration.
        final Map<String, DynamoDbEnhancedTableEncryptionConfig> tableConfigs = new
HashMap<>();
        tableConfigs.put(ddbTableName,
                DynamoDbEnhancedTableEncryptionConfig.builder()
                        .logicalTableName(ddbTableName)
                        .keyring(kmsKeyring)
                        .allowedUnsignedAttributes(allowedUnsignedAttributes)
                        .schemaOnEncrypt(tableSchema)
                        .legacyOverride(legacyOverride)
                        .build());
        final DynamoDbEncryptionInterceptor interceptor =
                DynamoDbEnhancedClientEncryption.CreateDynamoDbEncryptionInterceptor(
                        CreateDynamoDbEncryptionInterceptorInput.builder()
                                .tableEncryptionConfigs(tableConfigs)
                                .build()
                );

        // 8. Create a new AWS SDK DynamoDb client using the
        //    interceptor from Step 7.
        final DynamoDbClient ddb = DynamoDbClient.builder()
                .overrideConfiguration(
                        ClientOverrideConfiguration.builder()
                                .addExecutionInterceptor(interceptor)
                                .build())
                .build();

        // 9. Create the DynamoDbEnhancedClient using the AWS SDK DynamoDb client
        //    created in Step 8, and create a table with your modeled class.
        final DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
```

```
                .dynamoDbClient(ddb)
                .build();
        final DynamoDbTable<SimpleClass> table = enhancedClient.table(ddbTableName,
 tableSchema);
    }
}
```

Paso 2. Escriba los elementos en el nuevo formato

Una vez implementados los cambios del paso 1 en todos los lectores, complete los siguientes pasos para configurar el cliente del SDK de cifrado de bases de datos de AWS para que escriba los elementos en el nuevo formato. Tras implementar los siguientes cambios, el cliente seguirá leyendo los elementos en el formato anterior y empezará a escribir y leer los elementos en el nuevo formato.

Los siguientes procedimientos proporcionan una descripción general de los pasos que se muestran en el siguiente ejemplo de código.

1. Siga configurando el conjunto de claves, el esquema de la tabla`allowedUnsignedAttributes`, las acciones de los atributos heredados y `DynamoDBEncryptor` tal y como hizo en el paso 1.
2. Actualice su comportamiento anterior para escribir solo elementos nuevos con el nuevo formato.
3. Crear una `DynamoDbEncryptionInterceptor`
4. Crear un cliente del SDK DynamoDB de AWS.
5. Cree la clase modelada `DynamoDBEnhancedClient` y cree una tabla con ella.

   Para obtener más información sobre el cliente mejorado de DynamoDB, consulte crear un cliente mejorado.

```
public class MigrationExampleStep2 {

    public static void MigrationStep2(String kmsKeyId, String ddbTableName, int
 sortReadValue) {
        // 1. Continue to configure your keyring, table schema, legacy
        //    attribute actions, allowedUnsignedAttributes, and
        //    DynamoDBEncryptor as you did in Step 1.
        final MaterialProviders matProv = MaterialProviders.builder()
                .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
                .build();
```

```
        final CreateAwsKmsMrkMultiKeyringInput keyringInput =
CreateAwsKmsMrkMultiKeyringInput.builder()
                .generator(kmsKeyId)
                .build();
        final IKeyring kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);

        final TableSchema<SimpleClass> schemaOnEncrypt =
TableSchema.fromBean(SimpleClass.class);

        final List<String> allowedUnsignedAttributes = Arrays.asList("attribute3");

        final Map<String, CryptoAction> legacyActions = new HashMap<>();
        legacyActions.put("partition_key", CryptoAction.SIGN_ONLY);
        legacyActions.put("sort_key", CryptoAction.SIGN_ONLY);
        legacyActions.put("attribute1", CryptoAction.ENCRYPT_AND_SIGN);
        legacyActions.put("attribute2", CryptoAction.SIGN_ONLY);
        legacyActions.put("attribute3", CryptoAction.DO_NOTHING);

        final AWSKMS kmsClient = AWSKMSClientBuilder.defaultClient();
        final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kmsClient,
kmsKeyId);
        final DynamoDBEncryptor oldEncryptor = DynamoDBEncryptor.getInstance(cmp);

        // 2. Update your legacy behavior to only write new items using the new
        //    format.
        //    For Legacy Policy, use 'FORBID_LEGACY_ENCRYPT_ALLOW_LEGACY_DECRYPT'. This
policy
        //    continues to read items in both formats, but will only write items
        //    using the new format.
        final LegacyOverride legacyOverride = LegacyOverride
                .builder()
                .encryptor(oldEncryptor)
                .policy(LegacyPolicy.FORBID_LEGACY_ENCRYPT_ALLOW_LEGACY_DECRYPT)
                .attributeActionsOnEncrypt(legacyActions)
                .build();

        // 3. Create a DynamoDbEncryptionInterceptor with the above configuration.
        final Map<String, DynamoDbEnhancedTableEncryptionConfig> tableConfigs = new
HashMap<>();
        tableConfigs.put(ddbTableName,
                DynamoDbEnhancedTableEncryptionConfig.builder()
                        .logicalTableName(ddbTableName)
                        .keyring(kmsKeyring)
                        .allowedUnsignedAttributes(allowedUnsignedAttributes)
```

```
                        .schemaOnEncrypt(tableSchema)
                        .legacyOverride(legacyOverride)
                        .build());
        final DynamoDbEncryptionInterceptor interceptor =
                DynamoDbEnhancedClientEncryption.CreateDynamoDbEncryptionInterceptor(
                        CreateDynamoDbEncryptionInterceptorInput.builder()
                                .tableEncryptionConfigs(tableConfigs)
                                .build()
                );

        // 4. Create a new AWS SDK DynamoDb client using the
        //    interceptor from Step 3.
        final DynamoDbClient ddb = DynamoDbClient.builder()
                .overrideConfiguration(
                        ClientOverrideConfiguration.builder()
                                .addExecutionInterceptor(interceptor)
                                .build())
                .build();

        // 5. Create the DynamoDbEnhancedClient using the AWS SDK DynamoDb Client
 created
        //    in Step 4, and create a table with your modeled class.
        final DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();
        final DynamoDbTable<SimpleClass> table = enhancedClient.table(ddbTableName,
 tableSchema);
    }
}
```

Tras implementar los cambios del paso 2, debe volver a cifrar todos los elementos antiguos de la tabla con el nuevo formato para poder continuar con el paso 3. No hay una única métrica o consulta que puedas ejecutar para cifrar rápidamente los elementos existentes. Utilice el proceso que mejor se adapte a su sistema. Por ejemplo, podría utilizar un proceso asíncrono que escanee lentamente la tabla y reescriba los elementos utilizando las nuevas acciones de atributo y la nueva configuración de cifrado que haya definido.

Paso 3. Lea y escriba únicamente los elementos en el nuevo formato

Tras volver a cifrar todos los elementos de la tabla con el nuevo formato, puede eliminar el comportamiento anterior de la configuración. Siga los pasos que se indican a continuación para configurar el cliente para que solo lea y escriba los elementos en el nuevo formato.

Los siguientes procedimientos proporcionan una descripción general de los pasos que se muestran en el siguiente ejemplo de código.

1. Continúe configurando el conjunto de claves, el esquema de la tabla y `allowedUnsignedAttributes` tal como lo hizo en el paso 1. Elimine las acciones y acciones de los atributos heredados `DynamoDBEncryptor` de su configuración.
2. Crear una `DynamoDbEncryptionInterceptor`.
3. Crear un cliente del SDK DynamoDB de AWS.
4. Cree la clase modelada `DynamoDBEnhancedClient` y cree una tabla con ella.

   Para obtener más información sobre el cliente mejorado de DynamoDB, consulte crear un cliente mejorado.

```
public class MigrationExampleStep3 {

    public static void MigrationStep3(String kmsKeyId, String ddbTableName, int
 sortReadValue) {
        // 1. Continue to configure your keyring, table schema,
        //    and allowedUnsignedAttributes as you did in Step 1.
        //    Do not include the configurations for the DynamoDBEncryptor or
        //    the legacy attribute actions.
        final MaterialProviders matProv = MaterialProviders.builder()
                .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
                .build();
        final CreateAwsKmsMrkMultiKeyringInput keyringInput =
 CreateAwsKmsMrkMultiKeyringInput.builder()
                .generator(kmsKeyId)
                .build();
        final IKeyring kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);

        final TableSchema<SimpleClass> schemaOnEncrypt =
 TableSchema.fromBean(SimpleClass.class);

        final List<String> allowedUnsignedAttributes = Arrays.asList("attribute3");


        // 3. Create a DynamoDbEncryptionInterceptor with the above configuration.
        //    Do not configure any legacy behavior.
        final Map<String, DynamoDbEnhancedTableEncryptionConfig> tableConfigs = new
 HashMap<>();
```

```
        tableConfigs.put(ddbTableName,
                DynamoDbEnhancedTableEncryptionConfig.builder()
                        .logicalTableName(ddbTableName)
                        .keyring(kmsKeyring)
                        .allowedUnsignedAttributes(allowedUnsignedAttributes)
                        .schemaOnEncrypt(tableSchema)
                        .build());
        final DynamoDbEncryptionInterceptor interceptor =
                DynamoDbEnhancedClientEncryption.CreateDynamoDbEncryptionInterceptor(
                        CreateDynamoDbEncryptionInterceptorInput.builder()
                                .tableEncryptionConfigs(tableConfigs)
                                .build()
                );

        // 4. Create a new AWS SDK DynamoDb client using the
        //    interceptor from Step 3.
        final DynamoDbClient ddb = DynamoDbClient.builder()
                .overrideConfiguration(
                        ClientOverrideConfiguration.builder()
                                .addExecutionInterceptor(interceptor)
                                .build())
                .build();

        // 5. Create the DynamoDbEnhancedClient using the AWS SDK Client
        //    created in Step 4, and create a table with your modeled class.
        final DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();
        final DynamoDbTable<SimpleClass> table = enhancedClient.table(ddbTableName,
 tableSchema);
    }
}
```

Cliente de cifrado de DynamoDB antiguo

El 9 de junio de 2023, nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de base de datos de AWS. El SDK de cifrado de bases de datos de AWS sigue siendo compatible con las versiones antiguas de DynamoDB Encryption Client. Para obtener más información sobre las distintas partes de la biblioteca de cifrado del cliente que cambiaron con el cambio de nombre, consulte Cambio de nombre del Cliente de encriptación de Amazon DynamoDB.

Para migrar a la versión más reciente de la biblioteca de cifrado del cliente de Java para DynamoDB, consulte Migrar a la versión 3.x.

Temas

Compatibilidad con la versión SDK de cifrado de bases de datos de AWS para DynamoDB

Los temas del capítulo Legacy proporcionan información sobre las versiones 1. x —2. x del cliente de cifrado de DynamoDB para Java y versiones 1. x —3. x del cliente de cifrado de DynamoDB para Python.

En la siguiente tabla se enumeran los idiomas y las versiones que admiten el cifrado del cliente en Amazon DynamoDB.

Lenguaje de programación	Versión	Fase del ciclo de vida de la versión principal del SDK
Java	Versiones 1. x	Fase de fin de soporte, efectiva a partir de julio de 2022
Java	Versiones 2. x	Disponibilidad general (GA)
Java	Versión 3.x	Disponibilidad general (GA)
Python	Versiones 1. x	Fase de fin de soporte, efectiva a partir de julio de 2022

Lenguaje de programación	Versión	Fase del ciclo de vida de la versión principal del SDK
Python	Versiones 2. x	Fase de fin de soporte, efectiva a partir de julio de 2022
Python	Versiones 3. x	Disponibilidad general (GA)

Cómo funciona el cliente de cifrado de DynamoDB

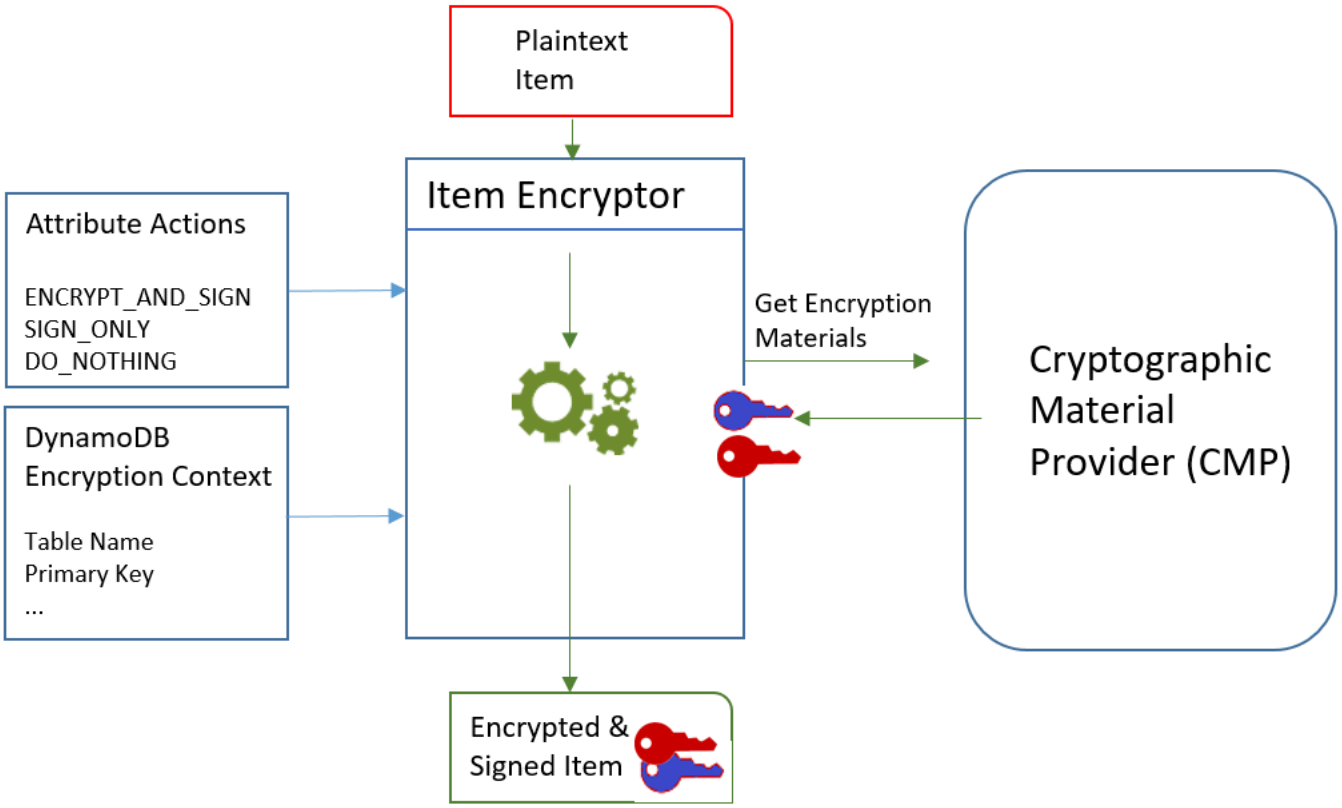
> **Note**
>
> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En el siguiente tema, se presenta información sobre las versiones 1.x—2.x del cliente de cifrado de DynamoDB para Java y versiones 1.x—3.x del cliente de cifrado de DynamoDB para Python. Para obtener más información, consulte el SDK de cifrado de bases de datos de AWS para la compatibilidad de la versión de DynamoDB.

El cliente de cifrado de DynamoDB está diseñado específicamente para proteger los datos que almacena en DynamoDB. Las bibliotecas incluyen implementaciones seguras que puede ampliar o utilizar sin hacer ningún cambio. La mayoría de los elementos se representan mediante elementos abstractos para que pueda crear y utilizar componentes personalizados compatibles.

Cifrado y firma de elementos de tabla

La esencia del cliente de cifrado de DynamoDB es un encriptador de elementos que cifra, firma, verifica y descifra los elementos de la tabla. Recibe información acerca de los elementos de tabla e instrucciones acerca de qué elementos hay que cifrar y firmar. Obtiene los materiales de cifrado, y las instrucciones sobre su uso, de un proveedor de materiales criptográficos que usted selecciona y configura.

En el siguiente diagrama, se muestra una vista general de este proceso.



Para cifrar y firmar un elemento de la tabla, el cliente de cifrado de DynamoDB necesita:

- Información acerca de la tabla. Obtiene información acerca de la tabla de un contexto de cifrado de DynamoDB que usted suministra. Algunos elementos auxiliares obtienen la información necesaria de DynamoDB y crean automáticamente el contexto de cifrado de DynamoDB para usted.

  > **Note**
  >
  > El contexto de cifrado de DynamoDB en el cliente de cifrado de DynamoDB no está relacionado con el contexto de cifrado en AWS Key Management Service (AWS KMS) y el AWS Encryption SDK.

- Los atributos que hay que cifrar y firmar. Obtiene esta información de las acciones de atributo que usted suministra.
- Materiales de cifrado, incluidas las claves de cifrado y firma. Los obtiene de un proveedor de materiales criptográficos (CMP) que usted selecciona y configura.

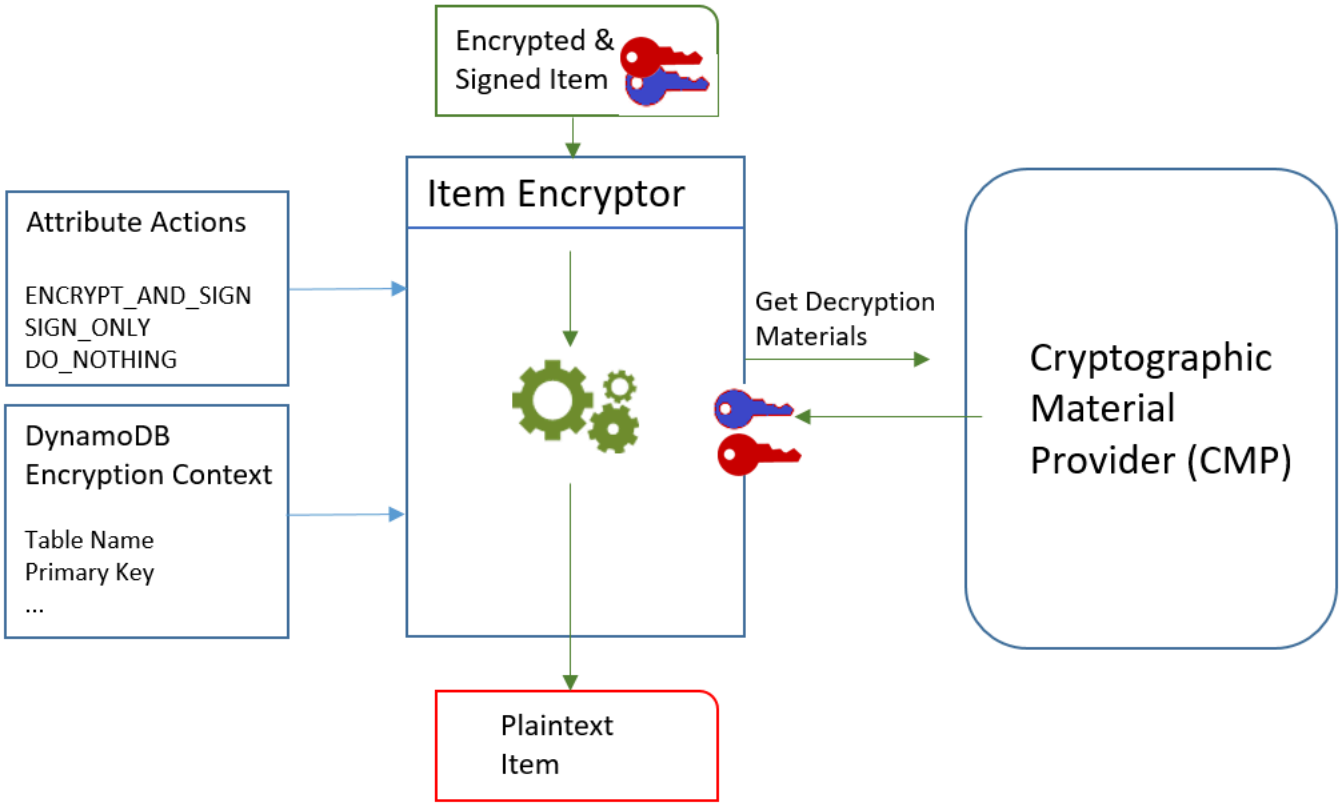
- Instrucciones para cifrar y firmar el elemento. El CMP añade instrucciones de uso de los materiales de cifrado, incluidos los algoritmos de cifrado y firma, a la descripción de material real.

El encriptador de elementos utiliza todos estos elementos para cifrar y firmar el elemento. El encriptador de elementos también añade dos atributos al elemento: un atributo de descripción de material que contiene las instrucciones de cifrado y firma (la descripción de material real) y un atributo que contiene la firma. Puede interactuar directamente con el encriptador de elementos, o puede utilizar características auxiliares que interactúan con el encriptador de elementos para implementar un comportamiento predeterminado seguro.

El resultado es un elemento de DynamoDB que contiene datos cifrados y firmados.

Verificación y descifrado de elementos de tabla

Estos componentes también funcionan juntos para verificar y descifrar el elemento, como se muestra en el siguiente diagrama.



Para verificar y descifrar un elemento, el cliente de cifrado de DynamoDB necesita los mismos componentes, componentes con la misma configuración o componentes diseñados especialmente para descifrar los elementos, de la siguiente manera:

- Información acerca de la tabla del contexto de cifrado de DynamoDB.
- Qué atributos verificar y descifrar. Los obtiene de las acciones de atributo.
- Materiales de descifrado, incluidas las claves de verificación y descifrado, del proveedor de materiales criptográficos (CMP) que usted selecciona y configura.

  El elemento cifrado no incluye ningún registro del CMP que se utilizó para cifrarlo. Debe proporcionar el mismo CMP, un CMP con la misma configuración o un CMP que esté diseñado para descifrar elementos.
- Información acerca de cómo el elemento se cifró y firmó, incluidos los algoritmos de cifrado y firma. El cliente los obtiene del atributo de descripción de material del elemento.

El encriptador de elementos utiliza todos estos elementos para verificar y descifrar el elemento. También elimina los atributos de descripción de material y firma. El resultado es un elemento de DynamoDB como texto no cifrado.

Conceptos del Cliente de encriptación de Amazon DynamoDB

> **Note**
>
> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En el siguiente tema, se presenta información sobre las versiones 1.x—2.x del cliente de cifrado de DynamoDB para Java y versiones 1.x—3.x del cliente de cifrado de DynamoDB para Python. Para obtener más información, consulte el SDK de cifrado de bases de datos de AWS para la compatibilidad de la versión de DynamoDB.

En este tema se explican la terminología y los conceptos empleados en el Cliente de encriptación de Amazon DynamoDB.

Para obtener información acerca de cómo interactúan los componentes del cliente de cifrado de DynamoDB, consulte Cómo funciona el cliente de cifrado de DynamoDB.

Temas

- Proveedor de materiales criptográficos (CMP)
- Encriptadores de elementos
- Acciones de atributo
- Descripción de material
- Contexto de cifrado de DynamoDB
- Almacén de proveedores

Proveedor de materiales criptográficos (CMP)

Al implementar el cliente de cifrado de DynamoDB, una de sus primeras tareas consiste en seleccionar un proveedor de materiales criptográficos (CMP) (también conocido como proveedor de materiales de cifrado). Esta elección determina gran parte del resto de la implementación.

Un proveedor de materiales criptográficos (CMP) recopila, reúne y devuelve los materiales criptográficos que el encriptador de elementos utiliza para cifrar y firmar los elementos de tabla. El CMP determina los algoritmos de cifrado que se utilizarán y cómo generar y proteger las claves de cifrado y firma.

El CMP interactúa con el encriptador de elementos. El encriptador de elementos solicita materiales de cifrado o descifrado al CMP, y el CMP los devuelve al encriptador de elementos. A continuación, el encriptador de elementos utiliza los materiales criptográficos para cifrar y firmar, o para verificar y descifrar, el elemento.

Debe especificar el CMP al configurar el cliente. Puede crear un CMP personalizado compatible o bien puede utilizar uno de los numerosos CMP de la biblioteca. La mayoría de los CMP están disponibles para varios lenguajes de programación.

Encriptadores de elementos

El encriptador de elementos es un componente de nivel inferior que realiza operaciones criptográficas para el cliente de cifrado de DynamoDB. Solicita materiales criptográficos a un proveedor de materiales criptográficos (CMP) y después utiliza los materiales que el CMP devuelve para cifrar y firmar, o para verificar y descifrar, el elemento de tabla.

Puede interactuar directamente con el encriptador de elementos o puede utilizar los elementos auxiliares proporcionados en la biblioteca. Por ejemplo, el cliente de cifrado de DynamoDB para Java incluye una clase auxiliar `AttributeEncryptorDynamoDBMapper` que puede utilizar con ,

en lugar de interactuar directamente con el encriptador de elementos `DynamoDBEncryptor`. La biblioteca Python incluye las clases auxiliares `EncryptedTable`, `EncryptedClient` y `EncryptedResource` que interactúan con el encriptador de elementos por usted.

Acciones de atributo

Las acciones de atributo indican al encriptador de elementos qué acciones hay que realizar en cada atributo del elemento.

Los valores de las acciones de atributo pueden ser uno de los siguientes:

- Encrypt and sign: cifra el valor del atributo. Incluir el atributo (nombre y valor) en la firma del elemento.
- Sign only: incluye el atributo en la firma del artículo.
- Do nothing: no cifre ni firme el atributo.

Para cualquier atributo que pueda almacenar datos confidenciales, use Encrypt and sign. Para los atributos de clave principal (clave de partición y clave de clasificación), utilice Sign only. El atributo de descripción de material y el atributo de firma no se firman ni se cifran. No es necesario especificar acciones para estos atributos.

Elija cuidadosamente sus acciones de atributo. En caso de duda, use Encrypt and sign. Una vez que haya utilizado la para proteger los elementos de la tabla, no puede cambiar la acción de un atributo sin arriesgarse a que se produzca un error de validación de firma. Para obtener más información, consulte Cambiar el modelo de datos.

> **Warning**
>
> No cifre los atributos de clave principal. Deben permanecer en texto no cifrado para que DynamoDB pueda encontrar el elemento sin realizar un examen completo de la tabla.

Si el contexto de cifrado de identifica sus atributos de clave principal, el cliente generará un error si intenta cifrarlos.

La técnica empleada para especificar las acciones de atributo es diferente para cada lenguaje de programación. También puede ser específica de las clases auxiliares que utilice.

Para obtener más información, consulte la documentación de su lenguaje de programación.

- Python
- Java

Descripción de material

La descripción de material de un elemento de tabla cifrado consta de información, como los algoritmos de cifrado, acerca del cifrado y la firma del elemento de tabla. El proveedor de materiales criptográficos (CMP) registra la descripción de material mientras reúne los materiales criptográficos para el cifrado y la firma. Posteriormente, cuando necesita reunir los materiales criptográficos para verificar y descifrar el elemento, utiliza la descripción de material como guía.

En el DynamoDB Encryption Client, la descripción de material se refiere a tres elementos relacionados:

Descripción de material solicitado

Algunos proveedores de materiales criptográficos (CMP) permiten especificar opciones avanzadas, como un algoritmo de cifrado. Para indicar sus opciones, añada pares de nombre y valor a la propiedad de descripción de material del contexto de cifrado de en la solicitud para cifrar un elemento de tabla. Este elemento se conoce como la descripción de material solicitado. Los valores válidos de la descripción de material solicitado los define el CMP elegido.

> **Note**
>
> Puesto que la descripción de material puede anular valores predeterminados seguros, le recomendamos que omita la descripción de material solicitado a menos que tenga una razón de peso para utilizarla.

Descripción de material real

La descripción de material que los proveedores de materiales criptográficos (CMP) devuelven se conoce como la descripción de material real. Describe los valores reales que el CMP utilizó cuando reunió los materiales criptográficos. Por lo general, consiste en la descripción de material solicitado, si se utiliza, con algunos cambios y adiciones.

Atributo de descripción de material

El cliente guarda la descripción de material real en el atributo de descripción de material del elemento cifrado. El nombre del atributo de descripción de material es `amzn-ddb-map-desc` y

su valor es la descripción de material real. El cliente utiliza los valores del atributo de descripción de material para verificar y descifrar el elemento.

Contexto de cifrado de DynamoDB

El contexto de cifrado de proporciona información acerca de la tabla y el elemento al proveedor de materiales criptográficos (CMP). En las implementaciones avanzadas, el contexto de cifrado de DynamoDB puede incluir una descripción del material solicitado.

Al cifrar elementos de tabla, el contexto de cifrado de DynamoDB está enlazado criptográficamente a los valores de atributo cifrados. Al descifrar, si el contexto de cifrado de no es una coincidencia exacta que distingue mayúsculas de minúsculas para el contexto de cifrado de utilizado para cifrar, se produce un error en la operación de descifrado. Si interactúa directamente con el cifrado de elementos debe proporcionar un contexto de cifrado de DynamoDB cuando llame a un método de cifrado o descifrado. La mayoría de los elementos auxiliares crean automáticamente el contexto de cifrado de .

> **Note**
>
> El contexto de cifrado de DynamoDB en el cliente de cifrado de DynamoDB no está relacionado con el contexto de cifrado en AWS Key Management Service (AWS KMS) y el AWS Encryption SDK.

El contexto de cifrado de puede incluir los siguientes campos. Todos los campos y valores son opcionales.

- Nombre de la tabla
- Nombre de la clave de partición
- Nombre de la clave de clasificación
- Pares de nombre y valor de los atributos
- Descripción de material solicitado

Almacén de proveedores

Un almacén de proveedores es un componente que devuelve proveedores de materiales criptográficos (CMP). El almacén de proveedores puede crear los CMP u obtenerlos de otra fuente,

como otro almacén de proveedores. El almacén de proveedores guarda en el almacenamiento persistente las versiones de los CMP que crea; cada CMP almacenado se identifica mediante el nombre de material del solicitante y el número de versión.

El proveedor más reciente ??? del cliente de cifrado de DynamoDB obtiene sus CMP de otro almacén de proveedores, aunque usted puede utilizar el almacén de proveedores para suministrar CMP a cualquier componente. Cada proveedor más reciente está asociado a un almacén de proveedores, pero un almacén de proveedores puede suministrar CMP a varios solicitantes en varios hosts.

El almacén de proveedores crea nuevas versiones de los CMP bajo demanda, y devuelve versiones nuevas y existentes. También devuelve el número de versión más reciente de un nombre de material determinado. De esta forma, el solicitante puede saber cuándo el almacén de proveedores tiene una nueva versión de su CMP que puede solicitar.

El cliente de cifrado de DynamoDB incluye un MetaStore, que es un almacén de proveedores que crea CMP encapsulados con claves almacenadas en DynamoDB y cifradas mediante un interno.

Más información:

- Almacén de proveedores: Java, Python
- MetaStore: Java, Python

Proveedor de materiales criptográficos

> **Note**
>
> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En el siguiente tema, se presenta información sobre las versiones 1.x—2.x del cliente de cifrado de DynamoDB para Java y versiones 1.x—3.x del cliente de cifrado de DynamoDB para Python. Para obtener más información, consulte el SDK de cifrado de bases de datos de AWS para la compatibilidad de la versión de DynamoDB.

Una de las decisiones más importantes que debe tomar al utilizar el cliente de cifrado de DynamoDB es seleccionar un proveedor de materiales criptográficos (CMP). El CMP combina y devuelve materiales criptográficos al encriptador de elementos. También determina cómo se generan las

claves de cifrado y de firma, si se generan nuevos materiales de clave para cada elemento o si se reutilizan, así como los algoritmos de cifrado y de firma que se utilizan.

Puede elegir un CMP para las implementaciones proporcionadas en las bibliotecas del cliente de cifrado de DynamoDB o crear un CMP personalizado compatible. Su opción de CMP también podría depender del lenguaje de programación que utilice.

En este tema se describen los CMP más comunes y se ofrecen algunos consejos para ayudarle a elegir el mejor para su aplicación.

Proveedor de materiales de KMS directo

El proveedor de materiales de KMS directo protege los elementos de la tabla con un AWS KMS keyque nunca se deja AWS Key Management Service(AWS KMS) sin cifrar. Su aplicación no tiene que generar ni gestionar ningún material criptográfico. Dado que utiliza la AWS KMS key para generar claves de cifrado y de firma únicas para cada elemento, este proveedor llama a AWS KMS cada vez que cifra o descifra un elemento.

Si utiliza AWS KMS y una llamada de AWS KMS por transacción es práctica para su aplicación, este proveedor es una buena opción.

Para obtener más información, consulte Proveedor de materiales de KMS directo.

Proveedor de materiales encapsulado (CMP encapsulado)

El proveedor de materiales encapsulado (CMP encapsulado) permite generar y administrar las claves de encapsulación y de firma desde fuera del cliente de cifrado de DynamoDB.

El CMP encapsulado genera una clave de cifrado única para cada elemento. A continuación utiliza las claves de encapsulación (o desencapsulación) y de firma que suministre. Por tanto, determina cómo se generan las claves de firma y encapsulación y si son únicas para cada elemento o si se reutilizan. El CMP encapsulado es una alternativa segura al proveedor de KMS directo para aplicaciones que no utilizan AWS KMS y puede administrar materiales criptográficos de forma segura.

Para obtener más información, consulte Proveedor de materiales encapsulado.

Proveedor más reciente

El proveedor más reciente es un proveedor de materiales criptográficos (CMP) que se ha diseñado para funcionar con un almacén de proveedores. Obtiene CMP del almacén de

proveedores y obtiene los materiales criptográficos que devuelve desde los CMP. El proveedor más reciente normalmente utiliza cada CMP para satisfacer varias solicitudes para materiales criptográficos, pero puede usar las características del almacén de proveedores para controlar hasta qué punto se reutilizan los materiales, determinar la frecuencia con la que se rota su CMP e incluso cambiar el tipo de CMP que se utiliza sin cambiar el proveedor más reciente.

Puede usar el proveedor más reciente con cualquier almacén de proveedores compatible. El cliente de cifrado de DynamoDB incluye un MetaStore, que es un almacén de proveedores que devuelve CMP encapsulados.

El proveedor más reciente es una buena opción para aplicaciones que necesitan minimizar las llamadas a su origen criptográfico y para aplicaciones que pueden reutilizar algunos materiales criptográficos sin infringir sus requisitos de seguridad. Por ejemplo, le permite proteger sus materiales criptográficos con un AWS KMS key en AWS Key Management Service(AWS KMS) sin tener que llamar AWS KMS cada vez que cifra o descifra un elemento.

Para obtener más información, consulte Proveedor más reciente.

Proveedor de materiales estático

El proveedor de materiales estático se ha diseñado para pruebas, demostraciones de prueba de concepto y compatibilidad heredada. No genera materiales criptográficos únicos para cada elemento. Devuelve las mismas claves de cifrado y firma que suministra y dichas claves se utilizan directamente para cifrar, descifrar y firmar los elementos de tabla.

> **Note**
>
> El Proveedor estático asimétrico de la biblioteca de Java no es un proveedor estático. Solo suministra constructores alternativos para el CMP encapsulado. Es seguro para uso de producción, pero se debe utilizar directamente el CMP encapsulado siempre que sea posible.

Temas

- Proveedor de materiales de KMS directo
- Proveedor de materiales encapsulado
- Proveedor más reciente
- Proveedor de materiales estático

Proveedor de materiales de KMS directo

> **Note**
>
> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En el siguiente tema, se presenta información sobre las versiones 1.x—2.x del cliente de cifrado de DynamoDB para Java y versiones 1.x—3.x del cliente de cifrado de DynamoDB para Python. Para obtener más información, consulte el SDK de cifrado de bases de datos de AWS para la compatibilidad de la versión de DynamoDB.

El proveedor de materiales de Direct KMS (proveedor de Direct KMS) protege los elementos de la tabla con un AWS KMS keyque nunca deja AWS Key Management Service (AWS KMS) sin cifrar. Este proveedor de materiales criptográficos devuelve una clave de cifrado y una clave de firma únicas para cada elemento de la tabla. Para ello, llama a AWS KMS cada vez que cifra o descifra un elemento.

Si procesa elementos de DynamoDB con una frecuencia alta y a gran escala, podría superar los AWS KMS límites de solicitudes por segundo y provocar demoras en el procesamiento. Si necesita superar estos límites, cree un caso en el AWS SupportCentro. También podría considerar la posibilidad de utilizar un proveedor de materiales criptográficos con una reutilización de claves limitada, como el proveedor más reciente.

Para utilizar el proveedor de KMS directo, el proveedor de llamadas debe tener una Cuenta de AWS, al menos una AWS KMS key y permiso para llamar a las operaciones GenerateDataKey y Decrypt en la AWS KMS key. La AWS KMS key debe ser una clave de cifrado simétrica; el cliente de cifrado de DynamoDB no admite el cifrado asimétrico. Si utiliza una tabla global de DynamoDB, posiblemente desee especificar una AWS KMS clave de múltiples regiones. Para obtener más información, consulte Modo de uso.

> **Note**
>
> Cuando se utiliza el proveedor de KMS directo, los nombres y los valores de los atributos de clave principal aparecen en texto no cifrado en el AWS KMScontexto de cifrado y en los AWS CloudTrail en los registros de las operaciones AWS KMS relacionadas. Sin embargo, el cliente de cifrado de DynamoDB nunca expone el texto no cifrado de ningún valor de atributo cifrado.

El proveedor de KMS directo es uno de los diversos proveedores de materiales criptográficos (CMP) que admite el cliente de cifrado de DynamoDB. Para obtener información acerca de otros CMP, consulte Proveedor de materiales criptográficos.

Para ver código de ejemplo, consulte:

- Java: AwsKmsEncryptedItem
- Python: aws-kms-encrypted-table, aws-kms-encrypted-item

Temas

- Modo de uso
- Cómo funciona

Modo de uso

Para crear un proveedor de KMS directo, utilice el parámetro de ID de clave para especificar una clave KMS de cifrado simétrico en su cuenta. El valor del parámetro ID de clave puede ser el ID de clave, el ARN de clave, el nombre de alias o el ARN de alias de AWS KMS key. Para obtener más información sobre los identificadores clave en la Guía para desarrolladores de AWS Key Management Service.

El proveedor de KMS directo necesita una clave KMS de cifrado simétrica. No puede utilizar una clave KMS asimétrica. Sin embargo, puede utilizar una clave KMS de múltiples regiones, una clave KMS con material de claves importado o una clave KMS en un almacén de claves personalizado. Debe tener los permisos kms:GenerateDataKey y kms:Decrypt en la clave KMS. Por lo tanto, debe utilizar una clave administrada por el cliente, no una clave KMS administrada por AWS o de la propiedad de AWS.

El cliente de cifrado de DynamoDB para Python determina la región a la que se debe llamar AWS KMS desde la región en el valor del parámetro de ID de clave, si incluye alguna. De lo contrario, utiliza la región en el cliente de AWS KMS, si se especifica una, o la región que se configura en el AWS SDK for Python (Boto3). Para obtener información acerca de la selección de regiones en Python, consulte Configuración en la Referencia de la API del SDK de AWS para Python (Boto3).

El cliente de cifrado de DynamoDB para Java determina la región a la que se debe llamar AWS KMS desde la región del cliente de AWS KMS, si el cliente que especifique incluye una región. De lo contrario, utiliza la región que usted configure en la AWS SDK for Java. Para obtener información

sobre la selección de la región en el AWS SDK for Java, consulte Selección de Región de AWS en la Guía para desarrolladores de AWS SDK for Java.

Java

```
// Replace the example key ARN and Region with valid values for your application
final String keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
final String region = 'us-west-2'

final AWSKMS kms = AWSKMSClientBuilder.standard().withRegion(region).build();
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, keyArn);
```

Python

En el siguiente ejemplo, se utiliza la clave ARN para especificar el AWS KMS key. Si su identificador de clave no incluye una Región de AWS, el cliente de cifrado de DynamoDB obtiene la región de la sesión de Botocore configurada, si la hay, o de los valores predeterminados de Boto.

```
# Replace the example key ID with a valid value
kms_key = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key)
```

Si utiliza tablas globales de Amazon DynamoDB, le recomendamos que cifre los datos con una clave de múltiples regiones AWS KMS. Las claves de múltiples regiones son AWS KMS keys en diferentes Regiones de AWS que se pueden utilizar indistintamente porque tienen el mismo ID de clave y el mismo material de claves. Para obtener más detalles, consulte Uso de claves de múltiples regiones en la Guía para desarrolladores de AWS Key Management Service.

> **Note**
>
> Si utiliza las tablas globales de la versión 2017.11.29, debe configurar las acciones de los atributos para que los campos de replicación reservados no estén cifrados ni firmados. Para obtener más información, consulte Problemas con las tablas globales de versiones anteriores.

Para usar una clave de múltiples regiones con el cliente de cifrado de DynamoDB, cree una clave de múltiples regiones y replíquela en las regiones en las que se ejecuta la aplicación. A continuación, configure el proveedor de Direct KMS para que utilice la clave de múltiples regiones en la región a la que llama el cliente de cifrado de DynamoDB AWS KMS.

En el siguiente ejemplo, se configura el cliente de cifrado de DynamoDB para que cifre datos en la región Este de EE. UU. (Norte de Virginia) (us-east-1) y los descifra en la región Oeste de EE. UU. (Oregón) (us-west-2) mediante una clave de múltiples regiones.

Java

En este ejemplo, el cliente de cifrado de DynamoDB obtiene la región para llamar a AWS KMS desde la región del AWS KMS cliente. El valor `keyArn` identifica una clave de múltiples regiones en la misma región.

```
// Encrypt in us-east-1

// Replace the example key ARN and Region with valid values for your application
final String usEastKey = 'arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab'
final String region = 'us-east-1'

final AWSKMS kms = AWSKMSClientBuilder.standard().withRegion(region).build();
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, usEastKey);
```

```
// Decrypt in us-west-2

// Replace the example key ARN and Region with valid values for your application
final String usWestKey = 'arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab'
final String region = 'us-west-2'

final AWSKMS kms = AWSKMSClientBuilder.standard().withRegion(region).build();
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, usWestKey);
```

Python

En este ejemplo, el cliente de cifrado de DynamoDB obtiene la región para llamar a AWS KMS desde la región en la ARN clave.

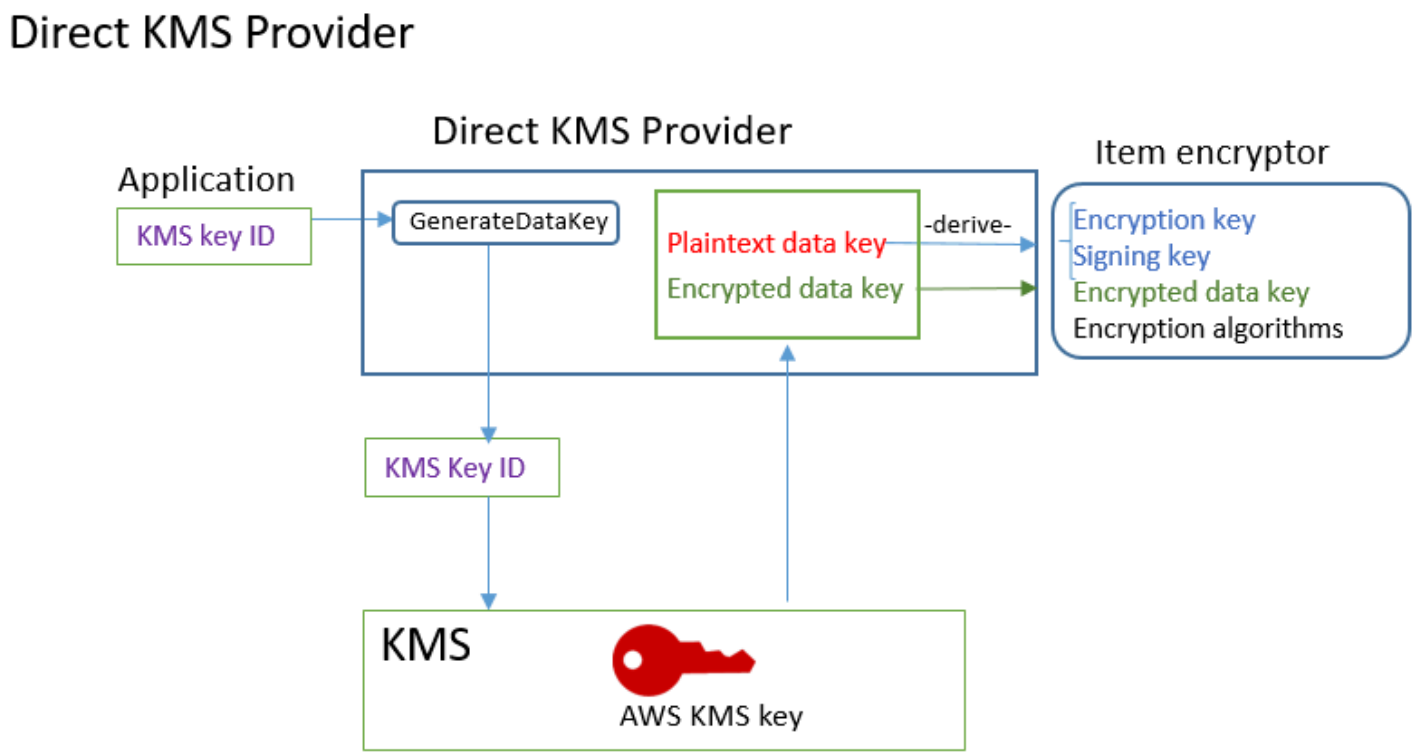
```
# Encrypt in us-east-1
```

```
# Replace the example key ID with a valid value
us_east_key = 'arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab'
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=us_east_key)
```

```
# Decrypt in us-west-2

# Replace the example key ID with a valid value
us_west_key = 'arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab'
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=us_west_key)
```

Cómo funciona

El proveedor de KMS directo devuelve las claves de cifrado y firma protegidas por una AWS KMS key que especifique, tal como se muestra en el diagrama siguiente.



- Para generar materiales de cifrado, el proveedor de KMS directo solicita al AWS KMS que genere una clave de datos única para cada elemento utilizando una AWS KMS key que especifique. Deriva las claves de cifrado y de firma para el elemento desde la copia de texto no cifrado de la

clave de datos y, a continuación, devuelve las claves de cifrado y de firma, junto con la clave de datos cifrada, que está almacenada en el atributo de descripción de material del elemento.

El encriptador de elementos utiliza las claves de cifrado y de firma y las elimina de la memoria lo antes posible. En el sistema cifrado solo se guarda la copia cifrada de la clave de datos desde la que se derivaron.

- Para generar materiales de descifrado, el proveedor de KMS directo pide a AWS KMS que descifre la clave de datos cifrada. A continuación, deriva las claves de verificación y firma desde la clave de datos de texto no cifrado y los devuelve al encriptador de elementos.

  El encriptador de elementos verifica el elemento y, si la verificación se realiza correctamente, descifra los valores cifrados. A continuación, elimina las claves de la memoria lo antes posible.

Obtener los materiales de cifrado

En esta sección se describen en detalle las entradas, las salidas y el procesamiento del proveedor de KMS directo cuando recibe una solicitud para materiales de cifrado desde el encriptador de elementos.

Entrada (desde la aplicación)

- El ID de clave de una AWS KMS key.

Entrada (desde el encriptador de elementos)

- Contexto de cifrado de DynamoDB

Salida (al encriptador de elementos)

- Clave de cifrado (texto no cifrado)
- Clave de firma
- En la descripción de material real: estos valores se guardan en el atributo de descripción de material que el cliente agrega al elemento.
  - amzn-ddb-env-key: clave de datos cifrada en Base64 cifrada por la AWS KMS key
  - amzn-ddb-env-alg: algoritmo de cifrado, de forma predeterminada AES/256
  - amzn-ddb-sig-alg: algoritmo de firma, de forma predeterminada, HmacSHA256/256
  - amzn-ddb-wrap-alg: kms

Processing

1. El proveedor de KMS directo envía a AWS KMS una solicitud para utilizar la AWS KMS key especificada para generar una clave de datos única para el elemento. La operación devuelve una clave de texto no cifrado y una copia que está cifrada con la AWS KMS key. Esto se conoce como el material de claves inicial.

   La solicitud incluye los siguientes valores en texto no cifrado en contexto de cifrado de AWS KMS. Estos valores no secretos están vinculados criptográficamente al objeto cifrado, de modo que se requiere el mismo contexto de cifrado para el descifrado. Puede utilizar estos valores para identificar la llamada a AWS KMS en registros de AWS CloudTrail.

   - amzn-ddb-env-alg: algoritmo de cifrado, de forma predeterminada AES/256
   - amzn-ddb-sig-alg: algoritmo de firma, de forma predeterminada, HmacSHA256/256
   - (Opcional) aws-kms-table: *`nombre de la tabla`*
   - (Opcional) *`nombre de la clave de partición`*: *`valor de la clave de partición`* (los valores binarios están codificados en Base64)
   - (Opcional) *`nombre de la clave de clasificación`*: *`valor de la clave de clasificación`* (los valores binarios están codificados en Base64)

   El proveedor de KMS directo obtiene los valores para el AWS KMScontexto de cifrado desde el contexto de cifrado de DynamoDB para el elemento. Si el contexto de cifrado de DynamoDB no incluye un valor como el nombre de tabla, el par nombre-valor se omite desde el AWS KMS contexto de cifrado.

2. El proveedor de KMS directo deriva una clave de cifrado simétrica y una clave de firma de la clave de datos. De forma predeterminada, utiliza algoritmo hash seguro (SHA) 256 y función de derivación de clave basada en HMAC RFC5869 para derivar una clave de cifrado simétrica AES de 256 bits y una clave de firma HMAC-SHA-256 de 256 bits.
3. El proveedor de KMS directo devuelve la salida al encriptador de elementos.
4. El encriptador de elementos utiliza la clave de cifrado para cifrar los atributos especificados y la clave de firma para firmarlos, utilizando los algoritmos especificados en la descripción de material real. Elimina las claves de texto no cifrado de la memoria lo antes posible.

Obtener los materiales de descifrado

En esta sección se describen en detalle las entradas, las salidas y el procesamiento del proveedor de KMS directo cuando recibe una solicitud para materiales de descifrado desde el encriptador de elementos.

Entrada (desde la aplicación)

- El ID de clave de una AWS KMS key.

  El valor del ID de clave puede ser el ID de clave, el ARN de clave, el nombre de alias o el ARN de alias del AWS KMS key. Los valores que no se especifiquen en el ID de clave, como la región, deben estar disponibles en el perfil nombrado AWS. El ARN de clave proporciona todos los valores que necesita AWS KMS.

Entrada (desde el encriptador de elementos)

- Una copia del contexto de cifrado de DynamoDB que contiene el contenido del atributo de descripción de material.

Salida (al encriptador de elementos)

- Clave de cifrado (texto no cifrado)
- Clave de firma

Processing

1. El proveedor de KMS directo obtiene la clave de datos cifrados desde el atributo de descripción del material en el elemento cifrado.
2. Solicita AWS KMS utilizar la AWS KMS key especificada para descifrar la clave de datos cifrados. La operación devuelve una clave de texto no cifrado.

   Esta solicitud debe usar el mismo contexto de cifrado de AWS KMS que se utilizó para generar y cifrar la clave de datos.

   - aws-kms-table: *`nombre de la tabla`*
   - *`nombre de la clave de partición`*: *`valor de la clave de partición`* (los valores binarios están codificados en Base64)

- (Opcional) *nombre de la clave de clasificación*: *valor de la clave de clasificación* (los valores binarios están codificados en Base64)
- amzn-ddb-env-alg: algoritmo de cifrado, de forma predeterminada AES/256
- amzn-ddb-sig-alg: algoritmo de firma, de forma predeterminada, HmacSHA256/256

3. El proveedor de KMS directo utiliza algoritmo hash seguro (SHA) 256 y función de derivación de clave basada en HMAC RFC5869 para derivar una clave de cifrado simétrica AES de 256 bits y una clave de firma HMAC-SHA-256 de 256 bits.
4. El proveedor de KMS directo devuelve la salida al encriptador de elementos.
5. El encriptador de elementos utiliza la clave de firma para verificar el elemento. Si se realiza correctamente, utiliza la clave de cifrado simétrica para descifrar los valores de atributo cifrados. Estas operaciones utilizan los algoritmos de cifrado y firma especificados en la descripción de material real. El encriptador de elementos elimina las claves de texto no cifrado de la memoria lo antes posible.

Proveedor de materiales encapsulado

> **Note**
>
> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En el siguiente tema, se presenta información sobre las versiones 1.x—2.x del cliente de cifrado de DynamoDB para Java y versiones 1.x—3.x del cliente de cifrado de DynamoDB para Python. Para obtener más información, consulte el SDK de cifrado de bases de datos de AWS para la compatibilidad de la versión de DynamoDB.

El proveedor de materiales encapsulado (CMP encapsulado) le permite utilizar las claves de encapsulación y de firma desde cualquier fuente con el cliente de cifrado de DynamoDB. El CMP encapsulado no depende de ningún servicio de AWS. Sin embargo, debe generar y administrar las claves de encapsulación y de firma fuera del cliente, incluida la entrega de las claves correctas para verificar y descifrar el elemento.

El CMP encapsulado genera una clave de cifrado de elemento única para cada elemento. Encapsula la clave de cifrado del elemento con la clave de encapsulación que proporcione y guarda la clave de cifrado de elemento encapsulado en el atributo de descripción de materiales del elemento. Dado que suministra las claves de encapsulación y de firma, determina cómo se generan las claves de firma y encapsulación y si son únicas para cada elemento o si se reutilizan.

El CMP encapsulado es una implementación segura y supone una buena opción para aplicaciones que puedan administrar materiales criptográficos.

El CMP encapsulado es uno de los diversos proveedores de materiales criptográficos (CMP) que admite el cliente de cifrado de DynamoDB. Para obtener información acerca de otros CMP, consulte Proveedor de materiales criptográficos.

Para ver código de ejemplo, consulte:

- Java: AsymmetricEncryptedItem
- Python: wrapped-rsa-encrypted-table, wrapped-symmetric-encrypted-table

Temas

- Modo de uso
- Cómo funciona

Modo de uso

Para crear un CMP encapsulado, especifique una clave de encapsulación (requerida durante el cifrado), una clave de desencapsulación (requerida durante el descifrado) y una clave de firma. Debe proporcionar las claves al cifrar y descifrar elementos.

Las claves de encapsulación, desencapsulación y firma pueden ser claves simétricas o pares de claves asimétricas.

Java

```
// This example uses asymmetric wrapping and signing key pairs
final KeyPair wrappingKeys = ...
final KeyPair signingKeys = ...

final WrappedMaterialsProvider cmp =
    new WrappedMaterialsProvider(wrappingKeys.getPublic(),
                                 wrappingKeys.getPrivate(),
                                 signingKeys);
```

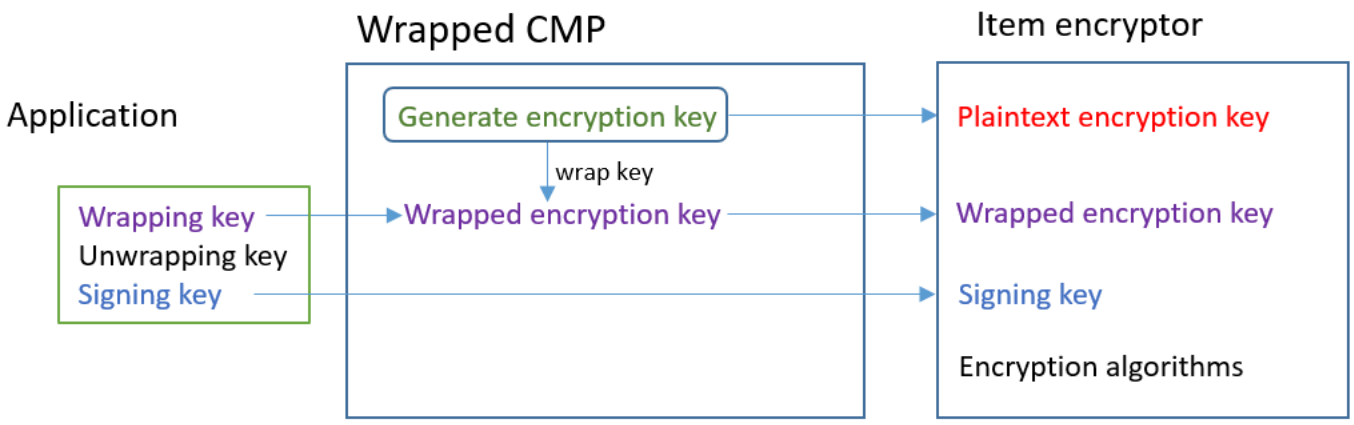
Python

```
# This example uses symmetric wrapping and signing keys
wrapping_key = ...
```

```
signing_key  = ...

wrapped_cmp = WrappedCryptographicMaterialsProvider(
    wrapping_key=wrapping_key,
    unwrapping_key=wrapping_key,
    signing_key=signing_key
)
```

Cómo funciona

El CMP encapsulado genera una clave de cifrado de elemento nueva para cada elemento. Utiliza las claves de encapsulación, desencapsulación y firma que proporcione, tal y como se muestra en el siguiente diagrama.



Obtener los materiales de cifrado

En esta sección se describen en detalle las entradas, las salidas y el procesamiento del proveedor de materiales encapsulado (CMP encapsulado) cuando recibe una solicitud para materiales de cifrado.

Entrada (desde la aplicación)

- Clave de encapsulación: una clave simétrica Advanced Encryption Standard (AES) o una clave pública RSA. Obligatorio si los valores de atributos están cifrados. De lo contrario, es opcional y se pasa por alto.
- Clave de desencapsulación: opcional y se pasa por alto.
- Clave de firma

Entrada (desde el encriptador de elementos)

- Contexto de cifrado de DynamoDB

Salida (al encriptador de elementos):

- Clave de cifrado de elemento de texto no cifrado
- Clave de firma (sin cambios)
- Descripción de material real: estos valores se guardan en el atributo de descripción de material que el cliente añade al elemento.
  - `amzn-ddb-env-key`: clave de cifrado de elemento encapsulado cifrado en Base64
  - `amzn-ddb-env-alg`: algoritmo de cifrado utilizado para cifrar el elemento. El valor predeterminado es AES-256-CBC.
  - `amzn-ddb-wrap-alg`: el algoritmo de encapsulación que utilizó el CMP encapsulado para encapsular la clave de cifrado del elemento. Si la clave de encapsulación es una clave AES, la clave se encapsula utilizando `AES-Keywrap` no rellenado, tal como se define en RFC 3394. Si la clave de encapsulación es una clave RSA, la clave se cifra utilizando RSA OAEP con relleno MGF1.

Processing

Cuando se cifra un elemento, transfiere una clave de encapsulación y una clave de firma. Una clave de desencapsulación es opcional y se pasa por alto.

1. El CMP encapsulado genera una clave de cifrado de elemento simétrica única para el elemento de tabla.
2. Utiliza la clave de cifrado que especifica para encapsular la clave de cifrado del elemento. A continuación, lo elimina de la memoria lo antes posible.
3. Devuelve la clave de cifrado del elemento con texto no cifrado, la clave de firma que suministró y una descripción de material real que incluye la clave de cifrado del elemento encapsulado y los algoritmos de cifrado y encapsulación.
4. El encriptador de elementos utiliza la clave de cifrado de texto no cifrado para cifrar el elemento. Utiliza la clave de firma que suministró para firmar el elemento. A continuación, elimina las claves de texto no cifrado de la memoria lo antes posible. Copia los campos en la descripción de material real, incluida la clave de cifrado encapsulada (`amzn-ddb-env-key`), en el atributo de descripción de material del elemento.

Obtener los materiales de descifrado

En esta sección se describen en detalle las entradas, las salidas y el procesamiento del proveedor de materiales encapsulado (CMP encapsulado) cuando recibe una solicitud para materiales de descifrado.

Entrada (desde la aplicación)

- Clave de encapsulación: opcional y se pasa por alto.
- Clave de encapsulación: la misma clave simétrica Advanced Encryption Standard (AES) o clave privada RSA que corresponde a la clave pública RSA utilizada para cifrar. Obligatorio si los valores de atributos están cifrados. De lo contrario, es opcional y se pasa por alto.
- Clave de firma

Entrada (desde el encriptador de elementos)

- Una copia del contexto de cifrado de DynamoDB que contiene el contenido del atributo de descripción de material.

Salida (al encriptador de elementos)

- Clave de cifrado de elemento de texto no cifrado
- Clave de firma (sin cambios)

Processing

Cuando se descifra un elemento, transfiere una clave de desencapsulación y una clave de firma. Una clave de encapsulación es opcional y se pasa por alto.

1. El CMP encapsulado obtiene la clave de cifrado del elemento encapsulado desde el atributo de descripción de material del elemento.
2. Utiliza la clave de desencapsulación y el algoritmo para desencapsular la clave de cifrado del elemento.
3. Devuelve la clave de cifrado del elemento con texto no cifrado, la clave de firma y los algoritmos de cifrado y de firma al encriptador de elementos.

4. El encriptador de elementos utiliza la clave de firma para verificar el elemento. Si se realiza correctamente, utiliza la clave de cifrado de elementos para descifrar el elemento. A continuación, elimina las claves de texto no cifrado de la memoria lo antes posible.

Proveedor más reciente

> **Note**
>
> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En el siguiente tema, se presenta información sobre las versiones 1.x—2.x del cliente de cifrado de DynamoDB para Java y versiones 1.x—3.x del cliente de cifrado de DynamoDB para Python. Para obtener más información, consulte el SDK de cifrado de bases de datos de AWS para la compatibilidad de la versión de DynamoDB.

El proveedor más reciente es un proveedor de materiales criptográficos (CMP) que se ha diseñado para funcionar con un almacén de proveedores. Obtiene CMP del almacén de proveedores y obtiene los materiales criptográficos que devuelve desde los CMP. Normalmente utiliza cada CMP para satisfacer varias solicitudes para materiales criptográficos. Pero puede utilizar las características de su almacén de proveedores para controlar hasta qué punto se reutilizan los materiales, determinar la frecuencia con la que se rota su CMP e, incluso, cambiar el tipo de CMP que utiliza sin cambiar el proveedor más reciente.

> **Note**
>
> El código asociado al símbolo `MostRecentProvider` del proveedor más reciente puede almacenar materiales criptográficos en la memoria durante todo el proceso. Podría permitir a la persona que llama usar claves que ya no está autorizada a usar.
> El símbolo `MostRecentProvider` está obsoleto en las versiones anteriores compatibles del cliente de cifrado de DynamoDB y se eliminó de la versión 2.0.0. Se sustituye por el símbolo `CachingMostRecentProvider`. Para obtener más información, consulte Actualizaciones del proveedor más reciente.

El proveedor más reciente es una buena opción para aplicaciones que necesitan minimizar las llamadas al almacén de proveedores y su origen criptográfico y para aplicaciones que pueden reutilizar algunos materiales criptográficos sin infringir sus requisitos de seguridad. Por ejemplo, le

permite proteger sus materiales criptográficos con un AWS KMS key en AWS Key Management Service (AWS KMS) sin tener que llamar a AWS KMS cada vez que cifra o descifra un elemento.

El almacén de proveedores que elija determina el tipo de CMP que utiliza el proveedor más reciente y la frecuencia con la que obtiene un CMP nuevo. Puede utilizar cualquier almacén de proveedores compatible con el proveedor más reciente, incluidos los almacenes de proveedor personalizados que diseñe.

El cliente de cifrado de DynamoDB incluye un MetaStore que crea y devuelve proveedores de materiales encapsulados (CMP encapsulados). El MetaStore guarda varias versiones de los CMP encapsulados que genera en una tabla de DynamoDB interna y los protege con un cifrado del cliente mediante una instancia interna del cliente de cifrado de DynamoDB.

Puede configurar el MetaStore para utilizar cualquier tipo de CMP interno para proteger los materiales de la tabla, incluido un proveedor de KMS directo que genera materiales criptográficos protegidos por su AWS KMS key, un CMP encapsulado que usa las claves de encapsulación y de firma que proporcione o un CMP personalizado compatible que diseñe.

Para ver código de ejemplo, consulte:

- Java: MostRecentEncryptedItem
- Python: most_recent_provider_encrypted_table

Temas

- Modo de uso
- Cómo funciona
- Actualizaciones del proveedor más reciente

Modo de uso

Para crear un proveedor más reciente, tiene que crear y configurar un almacén de proveedores y, a continuación, crear un proveedor más reciente que utiliza el almacén de proveedores.

En estos ejemplos, se muestra cómo crear un proveedor más reciente que utiliza un MetaStore y protege las versiones en su tabla interna de DynamoDB con materiales criptográficos de un proveedor directo de KMS. Estos ejemplos utilizan el símbolo `CachingMostRecentProvider`.

Cada proveedor más reciente tiene un nombre que identifica sus CMP en la tabla Metastore, una configuración de tiempo de vida (TTL) y una configuración de tamaño de caché que determina

cuántas entradas puede contener la caché. En estos ejemplos, se establece el tamaño de la caché en 1000 entradas y un TTL de 60 segundos.

Java

```
// Set the name for MetaStore's internal table
final String keyTableName = 'metaStoreTable'

// Set the Region and AWS KMS key
final String region = 'us-west-2'
final String keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

// Set the TTL and cache size
final long ttlInMillis = 60000;
final long cacheSize = 1000;

// Name that identifies the MetaStore's CMPs in the provider store
final String materialName = 'testMRP'

// Create an internal DynamoDB client for the MetaStore
final AmazonDynamoDB ddb =
 AmazonDynamoDBClientBuilder.standard().withRegion(region).build();

// Create an internal Direct KMS Provider for the MetaStore
final AWSKMS kms = AWSKMSClientBuilder.standard().withRegion(region).build();
final DirectKmsMaterialProvider kmsProv = new DirectKmsMaterialProvider(kms,
 keyArn);

// Create an item encryptor for the MetaStore,
// including the Direct KMS Provider
final DynamoDBEncryptor keyEncryptor = DynamoDBEncryptor.getInstance(kmsProv);

// Create the MetaStore
final MetaStore metaStore = new MetaStore(ddb, keyTableName, keyEncryptor);

//Create the Most Recent Provider
final CachingMostRecentProvider cmp = new CachingMostRecentProvider(metaStore,
 materialName, ttlInMillis, cacheSize);
```

Python

```
# Designate an AWS KMS key
```

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

# Set the name for MetaStore's internal table
meta_table_name = 'metaStoreTable'

# Name that identifies the MetaStore's CMPs in the provider store
material_name = 'testMRP'

# Create an internal DynamoDB table resource for the MetaStore
meta_table = boto3.resource('dynamodb').Table(meta_table_name)

# Create an internal Direct KMS Provider for the MetaStore
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key_id)

# Create the MetaStore with the Direct KMS Provider
meta_store = MetaStore(
    table=meta_table,
    materials_provider=kms_cmp
)

# Create a Most Recent Provider using the MetaStore
#    Sets the TTL (in seconds) and cache size (# entries)
most_recent_cmp = MostRecentProvider(
    provider_store=meta_store,
    material_name=material_name,
    version_ttl=60.0,
    cache_size=1000
)
```

Cómo funciona

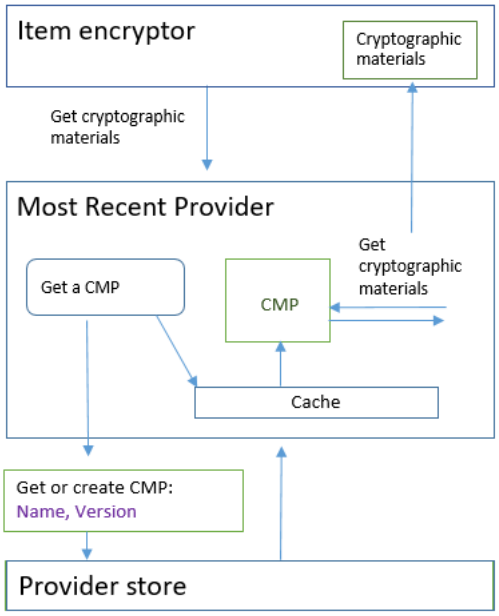
El proveedor más reciente obtiene los CMP desde un almacén de proveedores. A continuación, utiliza el CMP para generar los materiales criptográficos que devuelve al encriptador de elementos.

Acerca del proveedor más reciente

El proveedor más reciente obtiene un proveedor de materiales criptográficos (CMP) desde un almacén de proveedores. A continuación, utiliza el CMP para generar los materiales criptográficos que devuelve. Cada proveedor más reciente está asociado a un almacén de proveedores, pero un almacén de proveedores puede suministrar CMP a múltiples proveedores en varios hosts.

El proveedor más reciente puede funcionar con cualquier CMP compatible desde cualquier almacén de proveedores. El encriptador de elementos solicita materiales de cifrado o descifrado al CMP y los devuelve al encriptador de elementos. No realiza ninguna operación criptográfica.

Para solicitar un CMP desde su almacén de proveedores, el proveedor más reciente proporciona su nombre de material y la versión de un CMP existente que desea utilizar. Para los materiales de cifrado, el proveedor más reciente solicita siempre la versión máxima ("más reciente"). Para los materiales de descifrado, solicita la versión del CMP que se utilizó para crear los materiales de cifrado, tal como se muestra en el diagrama siguiente.



El proveedor más reciente guarda versiones de los CMP que el almacén de proveedores devuelve en una caché de elementos menos usados recientemente (LRU) local en memoria. La caché permite que el proveedor más reciente reciba los CMP que necesita sin llamar al almacén de proveedores para cada elemento. Puede borrar la caché bajo demanda.

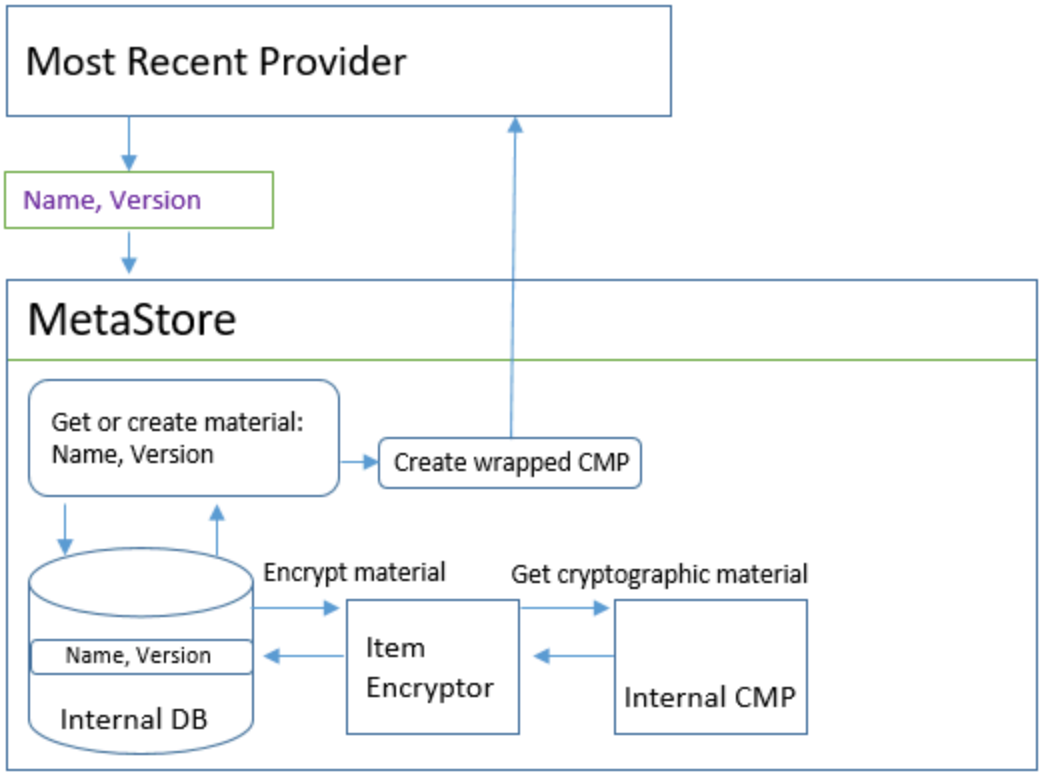
El proveedor más reciente utiliza un valor de tiempo de vida configurable que se puede ajustar en función de las características de la aplicación.

Acerca del MetaStore

Puede utilizar un proveedor más reciente con cualquier almacén de proveedores, incluido un almacén de proveedores personalizado compatible. El cliente de cifrado de DynamoDB incluye un MetaStore, una implementación segura que puede configurar y personalizar.

Un MetaStore es un almacén de proveedores que crea y devuelve CMP encapsulados que se configuran con la clave de encapsulación, la clave de desencapsulación y la clave de firma que requieren los CMP encapsulados. Un MetaStore es una opción segura para un proveedor más reciente, ya que los CMP encapsulados siempre generan claves de cifrado de elemento únicas para cada elemento. Solo se reutilizan la clave de encapsulación que protege la clave de cifrado del elemento y las claves de firma.

En el siguiente diagrama se muestran los componentes del MetaStore y cómo interactúa con el proveedor más reciente.



MetaStore genera los CMP empaquetados y, a continuación, los almacena (en forma cifrada) en una tabla interna de DynamoDB. La clave de partición es el nombre del material del proveedor más reciente; la clave de clasificación es el número de versión. Los materiales de la tabla están protegidos mediante un cliente de cifrado interno de DynamoDB, incluido un encriptador de elementos y un proveedor de materiales criptográficos (CMP) interno.

Puede utilizar cualquier tipo de CMP interno en su MetaStore, incluido un proveedor de KMS directo, un CMP encapsulado con materiales criptográficos que proporcione o un CMP personalizado compatible. Si el CMP interno de su MetaStore es un proveedor de KMS directo, sus claves de empaquetado y firma reutilizables están protegidas con un AWS KMS key en AWS Key Management

Service (AWS KMS). El MetaStore llama a AWS KMS cada vez que añade una nueva versión de CMP a su tabla interna u obtiene una versión de CMP desde su tabla interna.

Establecer un valor de tiempo de vida

Puede establecer un valor de tiempo de vida (TTL) para cada proveedor más reciente que cree. En general, utilice el valor TTL más bajo que resulte práctico para su aplicación.

El uso del valor de TTL se cambia en el símbolo `CachingMostRecentProvider` del proveedor más reciente.

> **Note**
>
> El `MostRecentProvider` símbolo del proveedor más reciente quedó obsoleto en las versiones anteriores compatibles del cliente de cifrado de DynamoDB y se eliminó de la versión 2.0.0. Se sustituye por el símbolo `CachingMostRecentProvider`. Se recomienda que actualice el código lo antes posible. Para obtener más información, consulte Actualizaciones del proveedor más reciente.

`CachingMostRecentProvider`

El `CachingMostRecentProvider` utiliza el valor de TTL de dos maneras diferentes.

- El TTL determina la frecuencia con la que el proveedor más reciente busca en la tienda del proveedor una nueva versión del CMP. Si hay una nueva versión disponible, el proveedor más reciente reemplaza su CMP y actualiza sus materiales criptográficos. De lo contrario, seguirá utilizando su CMP y sus materiales criptográficos actuales.
- El TTL determina durante cuánto tiempo se pueden usar los CMP de la memoria caché. Antes de utilizar un CMP almacenado en caché para el cifrado, el proveedor más reciente evalúa el tiempo que permanece en la memoria caché. Si el tiempo de caché de un CMP supera el TTL, el CMP se expulsa de la memoria caché y el proveedor más reciente obtiene una nueva versión del CMP de la última versión de la tienda de su proveedor.

`MostRecentProvider`

En el `MostRecentProvider`, el TTL determina la frecuencia con la que el proveedor más reciente busca en la tienda del proveedor una nueva versión del CMP. Si hay una nueva versión disponible, el proveedor más reciente reemplaza su CMP y actualiza sus materiales criptográficos. De lo contrario, seguirá utilizando su CMP y sus materiales criptográficos actuales.

El TTL no determina la frecuencia con la que se crea una nueva versión del CMP. Las nuevas versiones de CMP se crean rotando los materiales criptográficos.

Un valor de TTL ideal varía según la aplicación y sus objetivos de latencia y disponibilidad. Un TTL menor mejora el perfil de seguridad al reducir el tiempo que los materiales criptográficos se almacenan en la memoria. Además, un TTL menor actualiza la información crítica con más frecuencia. Por ejemplo, si su CMP interno es un proveedor de KMS directo, verificará con más frecuencia que la persona que llama siga estando autorizada a utilizar un AWS KMS key.

Sin embargo, si el TTL es demasiado breve, las llamadas frecuentes a la tienda del proveedor pueden aumentar los costos y hacer que la tienda del proveedor limite las solicitudes de su aplicación y de otras aplicaciones que comparten su cuenta de servicio. También podría resultarle útil coordinar el TTL con la velocidad de rotación de los materiales criptográficos.

Durante las pruebas, varíe el TTL y el tamaño de la caché según las distintas cargas de trabajo hasta que encuentre una configuración que se adapte a su aplicación y a sus estándares de seguridad y rendimiento.

Rotación de materiales criptográficos

Cuando un proveedor más reciente necesita materiales de cifrado, siempre utiliza la versión más reciente que conozca de su CMP. La frecuencia con la que comprueba si hay una versión más reciente viene determinada por el valor de tiempo de vida (TTL) que se establece al configurar el proveedor más reciente.

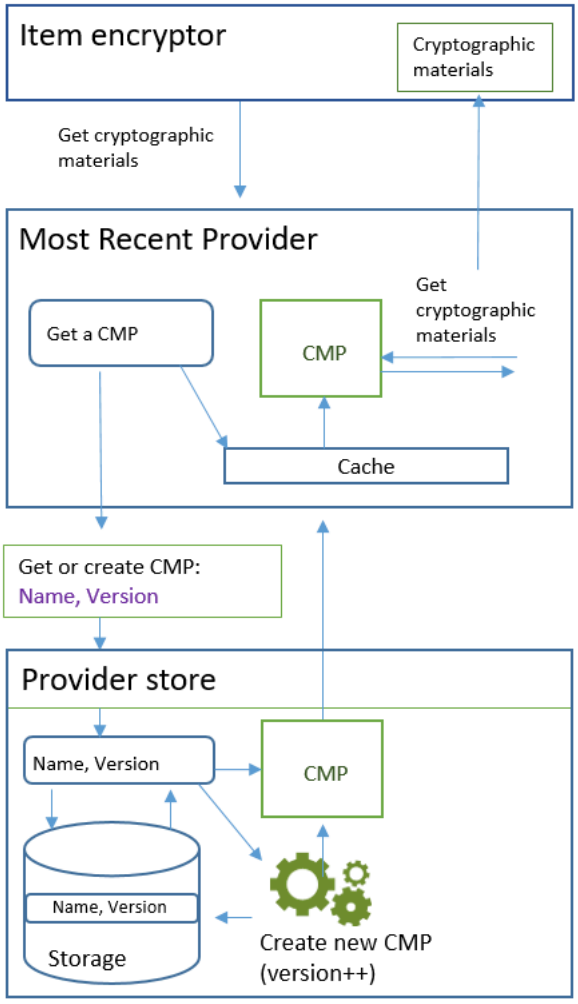
Cuando el TTL caduca, el proveedor más reciente busca en la tienda del proveedor una versión más reciente del CMP. Si hay alguna disponible, el proveedor más reciente la obtiene y reemplaza el CMP en su caché. Utiliza este CMP y sus materiales criptográficos hasta que descubre que la tienda del proveedor tiene una versión más reciente.

Para indicarle al almacén de proveedores que cree una nueva versión de un CMP para un proveedor más reciente, llame a la operación Crear nuevo proveedor del almacén de proveedores con el nombre del material del proveedor más reciente. El almacén de proveedores crea un nuevo CMP y guarda una copia cifrada en su almacén interno con un número de versión mayor. (También devuelve un CMP, pero puede descartarlo). En consecuencia, la próxima vez que el proveedor más reciente consulta al almacén de proveedores para conocer el número de versión máximo de sus CMP, obtiene el nuevo número de la versión superior y lo utiliza en posteriores solicitudes al almacén para ver si se creó una nueva versión del CMP.

Puede programar sus llamadas a Crear nuevo proveedor en función de la hora, del número de elementos o de los atributos procesados o de cualquier otra métrica que tenga sentido para su aplicación.

Obtener los materiales de cifrado

El proveedor más reciente utiliza el siguiente proceso, mostrado en este diagrama, para obtener los materiales de cifrado que devuelve al encriptador de elementos. La salida depende del tipo de CMP que el almacén de proveedores devuelve. El proveedor más reciente puede utilizar cualquier almacén de proveedores compatible, incluso el MetaStore que se incluye en el cliente de cifrado de DynamoDB.



Al crear un proveedor más reciente con el `CachingMostRecentProvider`símbolo, se especifica un almacén de proveedores, un nombre para el proveedor más reciente y un valor de tiempo de vida (TTL). Si lo desea, también puede especificar un tamaño de caché, que determina la cantidad máxima de materiales criptográficos que pueden existir en la caché.

Cuando el encriptador de elementos solicita al proveedor más reciente materiales de cifrado, el proveedor más reciente empieza buscando en su caché la versión más reciente de su CMP.

- Si encuentra el CMP con la versión más reciente en su caché y el CMP no ha excedido el valor de TTL, el proveedor más reciente utiliza el CMP para generar materiales de cifrado. A continuación, devuelve los materiales de cifrado al encriptador de elementos. Esta operación no requiere una llamada al almacén de proveedores.
- Si la última versión del CMP no está en su caché, o si está en la caché, pero excedió su valor de TTL, el proveedor más reciente solicita un CMP desde su almacén de proveedores. La solicitud incluye el nombre del material del proveedor más reciente y el número de versión máximo que conoce.

  1. El almacén de proveedores devuelve un CMP desde su almacenamiento persistente. Si el almacén de proveedores es un MetaStore, obtiene un CMP encapsulado cifrado desde su tabla de DynamoDB interna utilizando el nombre del material del proveedor más reciente como clave de partición y el número de la versión como la clave de clasificación. El MetaStore utiliza su encriptador de elementos interno y el CMP interno para descifrar el CMP encapsulado. A continuación, devuelve el CMP de texto no cifrado al proveedor más reciente. Si el CMP interno es un proveedor de KMS directo, este paso incluye una llamada al AWS Key Management Service (AWS KMS).
  2. El CMP agrega el campo `amzn-ddb-meta-id` a la descripción de material real. Su valor es el nombre de material y la versión del CMP en su tabla interna. El almacén del proveedor devuelve el CMP al proveedor más reciente.
  3. El proveedor más reciente almacena en la memoria caché el CMP.
  4. El proveedor más reciente utiliza el CMP para generar materiales de cifrado. A continuación, devuelve los materiales de cifrado al encriptador de elementos.

Obtener los materiales de descifrado

Cuando el encriptador de elementos solicita al proveedor más reciente los materiales de descifrado, el proveedor más reciente utiliza el proceso siguiente para obtenerlos y devolverlos.

1. El proveedor más reciente solicita al almacén de proveedores el número de la versión de los materiales criptográficos que se utilizaron para cifrar el elemento. Transfiere la descripción de material real desde el atributo de descripción de material del elemento.
2. El almacén de proveedores obtiene el número de versión de CMP de cifrado desde el campo `amzn-ddb-meta-id` en la descripción de material real y lo devuelve al proveedor más reciente.

3. El proveedor más reciente busca en la caché la versión del CMP que se utilizó para cifrar y firmar el elemento.

- Si encuentra que la versión coincidente del CMP está en su caché y el CMP no excedió el valor del tiempo de vida (TTL), el proveedor más reciente utiliza el CMP para generar materiales de descifrado. A continuación, devuelve los materiales de descifrado al encriptador de elementos. Esta operación no requiere una llamada al almacén de proveedores o cualquier otro CMP.
- Si la versión coincidente del CMP no está en su caché, o si la caché AWS KMS key excedió su valor de TTL, el proveedor más reciente solicita un CMP desde su almacén de proveedores. Envía el nombre del material y el número de versión de CMP de cifrado en la solicitud.

  1. El almacén de proveedores busca su almacenamiento persistente para el CMP utilizando el nombre del proveedor más reciente como clave de partición y el número de la versión como la clave de clasificación.
     - Si el nombre y el número de versión no están en su almacenamiento persistente, el almacén de proveedores genera una excepción. Si el almacén de proveedores se utilizó para generar el CMP, el CMP se debería almacenar en su almacenamiento persistente, a menos que se haya eliminado de forma intencionada.
     - Si el CMP con el nombre y número de versión coincidentes están en el almacenamiento persistente del almacén de proveedores, este devuelve el CMP especificado al proveedor más reciente.

       Si el almacén de proveedores es una MetaStore, obtiene el CMP cifrado desde su tabla de DynamoDB. A continuación, utiliza materiales criptográficos desde su CMP interno para descifrar el CMP cifrado antes de devolver el CMP al proveedor más reciente. Si el CMP interno es un proveedor de KMS directo, este paso incluye una llamada al AWS Key Management Service (AWS KMS).
  2. El proveedor más reciente almacena en la memoria caché el CMP.
  3. El proveedor más reciente utiliza el CMP para generar materiales de descifrado. A continuación, devuelve los materiales de descifrado al encriptador de elementos.

Actualizaciones del proveedor más reciente

El símbolo del proveedor más reciente cambia de `MostRecentProvider` a `CachingMostRecentProvider`.

> **Note**
>
> El símbolo `MostRecentProvider`, que representa al proveedor más reciente, está obsoleto en la versión 1.15 del cliente de cifrado de DynamoDB para Java y en la versión 1.3 del cliente de cifrado de DynamoDB para Python, y se eliminó de las versiones 2.0.0 del cliente de cifrado de DynamoDB en las implementaciones de ambos lenguajes. En su lugar, utilice el `CachingMostRecentProvider`.

El `CachingMostRecentProvider` implementa los siguientes cambios:

- El `CachingMostRecentProvider` elimina periódicamente los materiales criptográficos de la memoria cuando su tiempo en la memoria supera el valor del tiempo de vida (TTL) configurado.

  Es posible que `MostRecentProvider` almacene materiales criptográficos en la memoria durante el tiempo de vida del proceso. Como resultado, es posible que el proveedor más reciente no esté al tanto de los cambios de autorización. Es posible que utilice claves de cifrado una vez revocados los permisos de uso de la persona que llama.

  Si no puede actualizar a esta nueva versión, puede obtener un efecto similar si llama periódicamente al `clear()` método de la memoria caché. Este método vacía manualmente el contenido de la caché y requiere que el proveedor más reciente solicite un nuevo CMP y nuevos materiales criptográficos.
- El `CachingMostRecentProvider` también incluye una configuración de tamaño de la caché que le da más control sobre la caché.

Para actualizar el `CachingMostRecentProvider`, debe cambiar el nombre del símbolo en el código. En todos los demás aspectos, el `CachingMostRecentProvider` es totalmente compatible con versiones anteriores del `MostRecentProvider`. No es necesario volver a cifrar ningún elemento de la mesa.

Sin embargo, `CachingMostRecentProvider` genera más llamadas a la infraestructura clave subyacente. Llama a la tienda del proveedor al menos una vez en cada intervalo de vida (TTL, tiempo de vida). Lo más probable es que las aplicaciones con numerosos CMP activos (debido a la rotación frecuente) o las aplicaciones con grandes flotas sean más sensibles a este cambio.

Antes de publicar el código actualizado, pruébelo minuciosamente para asegurarse de que las llamadas más frecuentes no perjudiquen la aplicación ni provoquen una limitación por parte de los

servicios de los que depende su proveedor, como AWS Key Management Service (AWS KMS) o Amazon DynamoDB. Para mitigar cualquier problema de rendimiento, ajuste el tamaño de la caché y el tiempo de vida de `CachingMostRecentProvider` en función de las características de rendimiento que observe. Para obtener instrucciones, consulte Establecer un valor de tiempo de vida.

Proveedor de materiales estático

> **Note**
>
> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En el siguiente tema, se presenta información sobre las versiones 1.x—2.x del cliente de cifrado de DynamoDB para Java y versiones 1.x—3.x del cliente de cifrado de DynamoDB para Python. Para obtener más información, consulte el SDK de cifrado de bases de datos de AWS para la compatibilidad de la versión de DynamoDB.

El proveedor de materiales estático (CMP estático) es un proveedor de materiales criptográfico (CMP) muy sencillo destinado para pruebas, demostraciones de prueba de concepto y compatibilidad heredada.

Para utilizar el CMP estático para cifrar un elemento de tabla, proporcione una clave de cifrado simétrica con Advanced Encryption Standard (AES) y una clave o un par de claves de firma. Debe proporcionar las mismas claves para descifrar el elemento cifrado. El CMP estático no realiza ninguna operación criptográfica. En lugar de ello, transfiere las claves de cifrado que suministra al encriptador de elementos sin cambios. El encriptador de elementos cifra los elementos directamente bajo la clave de cifrado. A continuación, utiliza la clave de cifrado directamente para firmarlos.

Dado que el CMP estático no genera ningún material criptográfico único, todos los elementos de tabla que procesa están cifrados con la misma clave de cifrado y están firmados por la misma clave de firma. Cuando se utiliza la misma clave para cifrar los valores de atributos en muchos elementos o se utiliza la misma clave o par de claves para firmar todos los elementos, se arriesga a exceder los límites criptográficos de las claves.

> **Note**
>
> El Proveedor estático asimétrico de la biblioteca de Java no es un proveedor estático. Solo suministra constructores alternativos para el CMP encapsulado. Es seguro para uso de producción, pero se debe utilizar directamente el CMP encapsulado siempre que sea posible.

El CMP estático es uno de varios proveedores de materiales criptográficos (CMP) que admite el cliente de cifrado de DynamoDB. Para obtener información acerca de otros CMP, consulte Proveedor de materiales criptográficos.

Para ver código de ejemplo, consulte:

- Java: SymmetricEncryptedItem

Temas

- Modo de uso
- Cómo funciona

Modo de uso

Para crear un proveedor estático, suministre una clave un par de claves de cifrado o y una clave o par de claves de firma. Tiene que proporcionar material de claves para cifrar y descifrar elementos de tabla.

Java

```
// To encrypt
SecretKey cek = ...;        // Encryption key
SecretKey macKey =  ...;    // Signing key
EncryptionMaterialsProvider provider = new SymmetricStaticProvider(cek, macKey);

// To decrypt
SecretKey cek = ...;        // Encryption key
SecretKey macKey =  ...;    // Verification key
EncryptionMaterialsProvider provider = new SymmetricStaticProvider(cek, macKey);
```

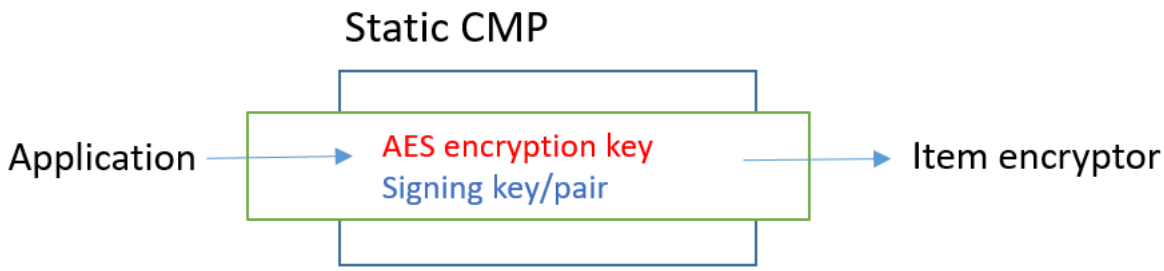
Python

```
# You can provide encryption materials, decryption materials, or both
encrypt_keys = EncryptionMaterials(
    encryption_key = ...,
    signing_key = ...
)

decrypt_keys = DecryptionMaterials(
    decryption_key = ...,
```

```
    verification_key = ...
)

static_cmp = StaticCryptographicMaterialsProvider(
    encryption_materials=encrypt_keys
    decryption_materials=decrypt_keys
)
```

Cómo funciona

El proveedor estático transfiere las claves de cifrado y de firma que suministre al encriptador de elementos, donde se utilizan directamente para cifrar y firmar los elementos de tabla. A menos que suministre distintas claves para cada elemento, se utilizan las mismas claves para todos ellos.



Obtener los materiales de cifrado

En esta sección se describen en detalle las entradas, las salidas y el procesamiento del proveedor de materiales estático (CMP estático) cuando recibe una solicitud para materiales de cifrado.

Entrada (desde la aplicación)

- Una clave de cifrado: esta debe ser una clave simétrica, como una clave del estándar de cifrado avanzado (AES).
- Una clave de firma: esta puede ser una clave simétrica o un par de claves asimétricas.

Entrada (desde el encriptador de elementos)

- Contexto de cifrado de DynamoDB

Salida (al encriptador de elementos)

- La clave de cifrado transferida como entrada.
- La clave de firma transferida como entrada.
- Descripción de material real: la descripción de material solicitada, si la hay, sin cambios.

Obtener los materiales de descifrado

En esta sección se describen en detalle las entradas, las salidas y el procesamiento del proveedor de materiales estático (CMP estático) cuando recibe una solicitud para materiales de descifrado.

Aunque incluye métodos independientes para obtener materiales de cifrado y obtener materiales de descifrado, el comportamiento es el mismo.

Entrada (desde la aplicación)

- Una clave de cifrado: esta debe ser una clave simétrica, como una clave del estándar de cifrado avanzado (AES).
- Una clave de firma: esta puede ser una clave simétrica o un par de claves asimétricas.

Entrada (desde el encriptador de elementos)

- Contexto de cifrado de DynamoDB (no utilizado)

Salida (al encriptador de elementos)

- La clave de cifrado transferida como entrada.
- La clave de firma transferida como entrada.

Lenguajes de programación disponibles para el Cliente de encriptación de Amazon DynamoDB

> **Note**
>
> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En el siguiente tema, se presenta información sobre las versiones 1.x—2.x del cliente de cifrado de DynamoDB para Java y versiones 1.x—3.x del cliente de cifrado de

DynamoDB para Python. Para obtener más información, consulte el SDK de cifrado de bases de datos de AWS para la compatibilidad de la versión de DynamoDB.

El Cliente de encriptación de Amazon DynamoDB está disponible para los siguientes lenguajes de programación. Las bibliotecas específicas de lenguaje varían, pero las implementaciones resultantes son interoperables. Por ejemplo, puede cifrar (y firmar) un elemento con el cliente Java y descifrar el elemento con el cliente Python.

Para obtener más información, consulte el tema correspondiente.

Temas

- Cliente de encriptación de Amazon DynamoDB para Java
- Cliente de cifrado de DynamoDB para Python

Cliente de encriptación de Amazon DynamoDB para Java

Note

Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En el siguiente tema, se presenta información sobre las versiones 1.x—2.x del cliente de cifrado de DynamoDB para Java y versiones 1.x—3.x del cliente de cifrado de DynamoDB para Python. Para obtener más información, consulte el SDK de cifrado de bases de datos de AWS para la compatibilidad de la versión de DynamoDB.

En este tema, se explica cómo instalar y utilizar el Cliente de encriptación de Amazon DynamoDB para Java. Para obtener información acerca de la programación con el cliente de cifrado de DynamoDB, consulte los ejemplos de Java, los ejemplos en el repositorio aws-dynamodb-encryption-java en GitHub y el Javadoc para el cliente de cifrado de DynamoDB.

Note

Las versiones 1.x.x del cliente de cifrado de DynamoDB para Java se encuentran en fase de fin de soporte a partir de julio de 2022. Actualice a una versión más reciente tan pronto como sea posible.

Temas

Requisitos previos

Antes de instalar el Cliente de encriptación de Amazon DynamoDB para Java, asegúrese de que cumple 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).

AWS SDK for Java

El cliente de cifrado de DynamoDB requiere el módulo DynamoDB del AWS SDK for Java aunque su aplicación no interactúe con DynamoDB. Puede instalar todo el SDK o solo este módulo. Si utiliza Maven, añada `aws-java-sdk-dynamodb` al archivo `pom.xml`.

Para obtener más información acerca de cómo instalar y configurar AWS SDK for Java, consulte AWS SDK for Java.

Instalación

Puede instalar el Cliente de encriptación de Amazon DynamoDB para Java de las siguientes maneras.

Manualmente

Para instalar el Cliente de encriptación de Amazon DynamoDB para Java, clone o descargue el repositorio de GitHub aws-dynamodb-encryption-java.

Con Apache Maven

El Cliente de encriptación de Amazon DynamoDB para Java está disponible en Apache Maven con la siguiente definición de dependencias.

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-dynamodb-encryption-java</artifactId>
  <version>version-number</version>
</dependency>
```

Después de instalar el SDK, comience por consultar el código de ejemplo de esta guía y el Javadoc del cliente de cifrado de DynamoDB en GitHub.

Uso del cliente de cifrado de DynamoDB para Java

> **Note**
>
> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En el siguiente tema, se presenta información sobre las versiones 1.x—2.x del cliente de cifrado de DynamoDB para Java y versiones 1.x—3.x del cliente de cifrado de DynamoDB para Python. Para obtener más información, consulte el SDK de cifrado de bases de datos de AWS para la compatibilidad de la versión de DynamoDB.

En este tema, se explican algunas de las características del cliente de cifrado de DynamoDB en Java que podrían no encontrarse en otras implementaciones de lenguaje de programación.

Para obtener más información sobre la programación con el cliente de cifrado de DynamoDB, consulte los ejemplos de Java, los ejemplos de `aws-dynamodb-encryption-java repository` en GitHub y Javadoc para el cliente de cifrado de DynamoDB.

Temas

- Encriptadores de elementos: AttributeEncryptor y DynamoDBEncryptor
- Configuración del comportamiento de almacenamiento
- Acciones de atributo en Java
- Reemplazar nombres de tabla

Encriptadores de elementos: AttributeEncryptor y DynamoDBEncryptor

El cliente de cifrado de DynamoDB de Java tiene dos encriptadores de elementos: el DynamoDBEncryptor de nivel inferior y el AttributeEncryptor.

El `AttributeEncryptor` es una clase auxiliar que le ayuda a utilizar DynamoDBMapper en el AWS SDK for Java con el `DynamoDB Encryptor` en el cliente de cifrado de DynamoDB. Cuando utiliza el `AttributeEncryptor` con el `DynamoDBMapper`, cifra y firma de forma transparente los elementos cuando los guarda. También verifica y descifra de forma transparente los elementos al cargarlos.

Configuración del comportamiento de almacenamiento

Puede usar el `AttributeEncryptor` y `DynamoDBMapper` para agregar o editar elementos de la tabla con atributos que solo están firmados o que están cifrados y firmados. Para estas tareas, recomendamos que lo configure para que use el comportamiento de guardado `PUT`, tal como se muestra en el siguiente ejemplo. De lo contrario, es posible que no pueda descifrar los datos.

```
DynamoDBMapperConfig mapperConfig =
 DynamoDBMapperConfig.builder().withSaveBehavior(SaveBehavior.PUT).build();
DynamoDBMapper mapper = new DynamoDBMapper(ddb, mapperConfig, new
 AttributeEncryptor(encryptor));
```

Si utiliza el comportamiento de almacenamiento predeterminado, que actualiza los atributos en el elemento de la tabla, solo se incluyen en la firma atributos que se hayan cambiado. Como resultado, en las lecturas posteriores de todos los atributos, la firma no se validará, porque no incluye los atributos no modelados.

También puede utilizar el comportamiento de guardado `CLOBBER`. Este comportamiento es idéntico al comportamiento de guardado `PUT`, pero deshabilita el bloqueo positivo y sobrescribe el elemento en la tabla.

Para evitar errores de firma, el cliente de cifrado de DynamoDB lanza una excepción de tiempo de ejecución si se utiliza un `AttributeEncryptor` con un `DynamoDBMapper` que no está configurado con un comportamiento seguro de `CLOBBER` o `PUT`.

Para ver un ejemplo de uso de este código, consulte Uso de DynamoDBMapper y el ejemplo de AwsKmsEncryptedObject.java en el repositorio `aws-dynamodb-encryption-java` en GitHub.

Acciones de atributo en Java

Las acciones de atributo determinan qué valores de atributo se cifran y se firman, cuáles solo se firman y cuáles se omiten. El método que utilice para especificar acciones de atributo depende de si usa `DynamoDBMapper` y `AttributeEncryptor` o el DynamoDBEncryptor de nivel inferior.

> **Important**
>
> Después de utilizar las acciones de atributo para cifrar los elementos de la tabla, agregar o quitar atributos del modelo de datos puede provocar un error de validación de firma que le impide descifrar los datos. Para ver una explicación detallada, consulte Cambiar el modelo de datos.

Acciones de atributo para el DynamoDBMapper

Cuando utilice `DynamoDBMapper` y `AttributeEncryptor`, utilice anotaciones para especificar las acciones de atributos. El cliente de cifrado de DynamoDB utiliza las anotaciones de atributo estándar que definen el tipo de atributo para determinar cómo proteger un atributo. De forma predeterminada, todos los atributos están cifrados y firmados, excepto las claves principales, que están firmadas, pero no cifradas.

> **Note**
>
> No cifre el valor de los atributos con la anotación @DynamoDBVersionAttribute, aunque puede (y debería) firmarlos. De lo contrario, las condiciones que utilizan su valor tendrán efectos no previstos.

```
// Attributes are encrypted and signed
@DynamoDBAttribute(attributeName="Description")

// Partition keys are signed but not encrypted
@DynamoDBHashKey(attributeName="Title")

// Sort keys are signed but not encrypted
@DynamoDBRangeKey(attributeName="Author")
```

Para especificar las excepciones, utilice las anotaciones de cifrado que se definen en el cliente de cifrado de DynamoDB para Java. Si las especifica en el nivel de clase, se convierten en el valor predeterminado para la clase.

```
// Sign only
@DoNotEncrypt

// Do nothing; not encrypted or signed
@DoNotTouch
```

Por ejemplo, estas anotaciones firman pero no cifran el atributo `PublicationYear` y no cifran o firman el valor del atributo `ISBN`.

```
// Sign only (override the default)
@DoNotEncrypt
@DynamoDBAttribute(attributeName="PublicationYear")

// Do nothing (override the default)
@DoNotTouch
@DynamoDBAttribute(attributeName="ISBN")
```

Acciones de atributo para el DynamoDBEncryptor

Para especificar las acciones de atributo cuando se utiliza el DynamoDBEncryptor directamente, cree un objeto `HashMap` en el que las parejas de nombre-valor representen nombres de atributo y las acciones especificadas.

Los valores válidos para las acciones de atributo que se definen en el tipo enumerado `EncryptionFlags`. Puede utilizar `ENCRYPT` y `SIGN` juntos, usar `SIGN` solo u omitir ambos. Sin embargo, si usa `ENCRYPT` solo, el cliente de cifrado de DynamoDB generará un error. No puede cifrar un atributo que no firme.

```
ENCRYPT
SIGN
```

> **Warning**
>
> No cifre los atributos de clave principal. Deben permanecer en texto no cifrado para que DynamoDB pueda encontrar el elemento sin realizar un examen completo de la tabla.

Si especifica una clave principal en el contexto de cifrado y, a continuación, especifica ENCRYPT en la acción de atributo para alguno de los atributos de clave principal, el cliente de cifrado de DynamoDB genera una excepción.

Por ejemplo, el siguiente código Java crea un HashMap de `actions` que cifra y firma todos los atributos del elemento `record`. Las excepciones son la clave de partición y los atributos de clave de clasificación, que se firman pero no se cifran, y el atributo `test`, que ni se signa ni se cifra.

```
final EnumSet<EncryptionFlags> signOnly = EnumSet.of(EncryptionFlags.SIGN);
final EnumSet<EncryptionFlags> encryptAndSign = EnumSet.of(EncryptionFlags.ENCRYPT,
 EncryptionFlags.SIGN);
final Map<String, Set<EncryptionFlags>> actions = new HashMap<>();

for (final String attributeName : record.keySet()) {
  switch (attributeName) {
    case partitionKeyName: // no break; falls through to next case
    case sortKeyName:
      // Partition and sort keys must not be encrypted, but should be signed
      actions.put(attributeName, signOnly);
      break;
    case "test":
      // Don't encrypt or sign
      break;
    default:
      // Encrypt and sign everything else
      actions.put(attributeName, encryptAndSign);
      break;
  }
}
```

A continuación, cuando llama al método encryptRecord del `DynamoDBEncryptor`, especifique el mapa como el valor del parámetro `attributeFlags`. Por ejemplo, esta llamada a `encryptRecord` utiliza el mapa `actions`.

```
// Encrypt the plaintext record
final Map<String, AttributeValue> encrypted_record = encryptor.encryptRecord(record,
 actions, encryptionContext);
```

Reemplazar nombres de tabla

En el cliente de cifrado de DynamoDB, el nombre de la tabla de DynamoDB es un elemento del contexto de cifrado de DynamoDB que se pasa a los métodos de cifrado y descifrado. Al cifrar o

firmar elementos de la tabla, el contexto de cifrado de DynamoDB, incluido el nombre de la tabla, está enlazado criptográficamente al texto cifrado. Si el contexto de cifrado de DynamoDB que se pasa al método de descifrado no coincide con el contexto de cifrado de DynamoDB que se pasó al método de cifrado, se produce un error en la operación de descifrado.

Ocasionalmente, el nombre de una tabla cambia, como cuando se realiza una copia de seguridad de una tabla o se realiza una recuperación puntual. Al descifrar o verificar la firma de estos elementos, debe pasar el mismo contexto de cifrado de DynamoDB que se utilizó para cifrar y firmar los elementos, incluido el nombre de la tabla original. El nombre de la tabla actual no es necesario.

Cuando se utiliza `DynamoDBEncryptor`, se ensambla el contexto de cifrado de DynamoDB manualmente. Sin embargo, si está utilizando `DynamoDBMapper`, el `AttributeEncryptor` crea el contexto de cifrado de DynamoDB para usted, incluido el nombre de la tabla actual. Para indicar a `AttributeEncryptor` que cree un contexto de cifrado con un nombre de tabla diferente, utilice el `EncryptionContextOverrideOperator`.

Por ejemplo, el código siguiente crea instancias del proveedor de materiales criptográficos (CMP) y el `DynamoDBEncryptor`. Luego llama al método `setEncryptionContextOverrideOperator` de `DynamoDBEncryptor`. Utiliza el operador `overrideEncryptionContextTableName`, que anula un nombre de tabla. Cuando se configura de esta manera, el `AttributeEncryptor` crea un contexto de cifrado de DynamoDB que incluye `newTableName` en lugar de `oldTableName`. Para obtener un ejemplo completo, vea EncryptionContextOverridesWithDynamoDBMapper.java.

```
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, keyArn);
final DynamoDBEncryptor encryptor = DynamoDBEncryptor.getInstance(cmp);

encryptor.setEncryptionContextOverrideOperator(EncryptionContextOperators.overrideEncryptionCon
                oldTableName, newTableName));
```

Cuando se llama al método de carga de `DynamoDBMapper`, que descifra y verifica el elemento, se especifica el nombre de la tabla original.

```
mapper.load(itemClass, DynamoDBMapperConfig.builder()

 .withTableNameOverride(DynamoDBMapperConfig.TableNameOverride.withTableNameReplacement(oldTabl
                .build());
```

También puede utilizar el operador `overrideEncryptionContextTableNameUsingMap`, que anula varios nombres de tabla.

Normalmente los operadores de reemplazo de nombres de tabla se utilizan al descifrar datos y verificar firmas. Sin embargo, puede usarlos para establecer el nombre de la tabla en el contexto de cifrado de DynamoDB en un valor diferente al cifrar y firmar.

No utilice los operadores de anulación de nombre de tabla si está utilizando `DynamoDBEncryptor`. En su lugar, cree un contexto de cifrado con el nombre de la tabla original y envíelo al método de descifrado.

Ejemplo de código para el cliente de cifrado de DynamoDB para Java

> **Note**
>
> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En el siguiente tema, se presenta información sobre las versiones 1.x—2.x del cliente de cifrado de DynamoDB para Java y versiones 1.x—3.x del cliente de cifrado de DynamoDB para Python. Para obtener más información, consulte el SDK de cifrado de bases de datos de AWS para la compatibilidad de la versión de DynamoDB.

Los siguientes ejemplos muestran cómo utilizar el cliente de cifrado de DynamoDB para Java para proteger los elementos de tabla de DynamoDB en su aplicación. Puede encontrar más ejemplos (y aportar los suyos) en el directorio de ejemploshttps://github.com/aws/aws-dynamodb-encryption-java/tree/master/examples del repositorio aws-dynamodb-encryption-java en GitHub.

Temas

- Uso de DynamoDBEncryptor
- Uso de DynamoDBMapper

Uso de DynamoDBEncryptor

Este ejemplo muestra cómo utilizar el DynamoDBEncryptor de nivel inferior con el proveedor de KMS directo. El proveedor de KMS directo genera y protege sus materiales criptográficos con un AWS KMS key en AWS Key Management Service (AWS KMS) que usted especifique.

Puede utilizar cualquier proveedor de materiales criptográficos (CMP) compatible con el `DynamoDBEncryptor` y puede utilizar el proveedor de KMS directo con el `DynamoDBMapper` y AttributeEncryptor.

Vea la muestra de código completa: AwsKmsEncryptedItem.java

Paso 1: crear el proveedor de KMS directo

Cree una instancia de cliente de AWS KMS con la región especificada. A continuación, utilice la instancia de cliente para crear una instancia del proveedor de KMS directo con su AWS KMS key preferido.

Este ejemplo utiliza el Nombre de recurso de Amazon (ARN) para identificar la AWS KMS key, pero puede utilizar cualquier identificador de clave válido.

```
final String keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
final String region = 'us-west-2'

final AWSKMS kms = AWSKMSClientBuilder.standard().withRegion(region).build();
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, keyArn);
```

Paso 2: crear un elemento

Este ejemplo define un HashMap `record` que representa un elemento de tabla de ejemplo.

```
final String partitionKeyName = "partition_attribute";
final String sortKeyName = "sort_attribute";

final Map<String, AttributeValue> record = new HashMap<>();
record.put(partitionKeyName, new AttributeValue().withS("value1"));
record.put(sortKeyName, new AttributeValue().withN("55"));
record.put("example", new AttributeValue().withS("data"));
record.put("numbers", new AttributeValue().withN("99"));
record.put("binary", new AttributeValue().withB(ByteBuffer.wrap(new byte[]{0x00,
 0x01, 0x02})));
record.put("test", new AttributeValue().withS("test-value"));
```

Paso 3: crear un DynamoDBEncryptor

Cree una instancia del `DynamoDBEncryptor` con el proveedor de KMS directo.

```
final DynamoDBEncryptor encryptor = DynamoDBEncryptor.getInstance(cmp);
```

Paso 4: crear un contexto de cifrado de DynamoDB

El contexto de cifrado de DynamoDB contiene información acerca de la estructura de la tabla y cómo se cifra y se firma. Si utiliza el `DynamoDBMapper`, el `AttributeEncryptor` crea el contexto de cifrado automáticamente.

```
final String tableName = "testTable";

final EncryptionContext encryptionContext = new EncryptionContext.Builder()
    .withTableName(tableName)
    .withHashKeyName(partitionKeyName)
    .withRangeKeyName(sortKeyName)
    .build();
```

Paso 5: crear el objeto de acciones de atributo

Las acciones de atributo determinan qué atributos del elemento se cifran y se firman, cuáles solo se firman y cuáles no se cifran o firman.

En Java, para especificar las acciones de atributo, puede crear un HashMap de nombre de atributo y pares de valor `EncryptionFlags`.

Por ejemplo, el siguiente código Java crea un HashMap `actions` que cifra y firma todos los atributos en el elemento `record`, excepto para los atributos de clave de partición y de clave de clasificación, que están firmados pero no cifrados y el atributo `test`, que no está firmado o cifrado.

```
final EnumSet<EncryptionFlags> signOnly = EnumSet.of(EncryptionFlags.SIGN);
final EnumSet<EncryptionFlags> encryptAndSign = EnumSet.of(EncryptionFlags.ENCRYPT,
 EncryptionFlags.SIGN);
final Map<String, Set<EncryptionFlags>> actions = new HashMap<>();

for (final String attributeName : record.keySet()) {
  switch (attributeName) {
    case partitionKeyName: // fall through to the next case
    case sortKeyName:
      // Partition and sort keys must not be encrypted, but should be signed
      actions.put(attributeName, signOnly);
      break;
    case "test":
      // Neither encrypted nor signed
      break;
```

```
    default:
      // Encrypt and sign all other attributes
      actions.put(attributeName, encryptAndSign);
      break;
  }
}
```

Paso 6: cifrar y firmar el elemento

Para cifrar y firmar el elemento de tabla, llame al método `encryptRecord` en la instancia del `DynamoDBEncryptor`. Especifique el elemento de tabla (`record`), las acciones de atributo (`actions`) y el contexto de cifrado (`encryptionContext`).

```
final Map<String, AttributeValue> encrypted_record = encryptor.encryptRecord(record,
 actions, encryptionContext);
```

Paso 7: colocar el elemento en la tabla de DynamoDB

Finalmente, coloque el elemento cifrado y firmado en la tabla de DynamoDB.

```
final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();
ddb.putItem(tableName, encrypted_record);
```

Uso de DynamoDBMapper

El ejemplo siguiente le muestra cómo utilizar la clase auxiliar del DynamoDB Mapper con el Proveedor de KMS directo. El proveedor de KMS directo genera y protege sus materiales criptográficos con un AWS KMS key en AWS Key Management Service (AWS KMS) que usted especifique.

Puede utilizar cualquier proveedor de materiales criptográficos (CMP) compatible con el `DynamoDBMapper` y puede utilizar el proveedor de KMS directo con el `DynamoDBEncryptor` de nivel inferior.

Vea la muestra de código completa: AwsKmsEncryptedObject.java

Paso 1: crear el proveedor de KMS directo

Cree una instancia de cliente de AWS KMS con la región especificada. A continuación, utilice la instancia de cliente para crear una instancia del proveedor de KMS directo con su AWS KMS key preferido.

Este ejemplo utiliza el Nombre de recurso de Amazon (ARN) para identificar la AWS KMS key, pero puede utilizar cualquier identificador de clave válido.

```
final String keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
final String region = 'us-west-2'

final AWSKMS kms = AWSKMSClientBuilder.standard().withRegion(region).build();
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, keyArn);
```

Paso 2: Crear el Encriptador de DynamoDB y DynamoDBMapper

Utilice el Proveedor de KMS directo que creó en el paso anterior para crear una instancia del Encriptador de DynamoDB. Debe crear instancias en el Encriptador de DynamoDB de nivel inferior para utilizar DynamoDB Mapper.

A continuación, cree una instancia de base de datos de DynamoDB y una configuración de mapeador, y úselas para crear una instancia de DynamoDB Mapper.

> **Important**
>
> Al utilizar el `DynamoDBMapper` para añadir o editar elementos firmados (o cifrados y firmados), configúrelo para usar un comportamiento de almacenamiento, como PUT, que incluye todos los atributos, como se muestra en el ejemplo siguiente. De lo contrario, es posible que no pueda descifrar los datos.

```
final DynamoDBEncryptor encryptor = DynamoDBEncryptor.getInstance(cmp)
final AmazonDynamoDB ddb =
 AmazonDynamoDBClientBuilder.standard().withRegion(region).build();

DynamoDBMapperConfig mapperConfig =
 DynamoDBMapperConfig.builder().withSaveBehavior(SaveBehavior.PUT).build();
DynamoDBMapper mapper = new DynamoDBMapper(ddb, mapperConfig, new
 AttributeEncryptor(encryptor));
```

Paso 3: Definir la tabla de DynamoDB

A continuación, defina la tabla de DynamoDB. Utilice anotaciones para especificar las acciones del atributo. En este ejemplo, se crea una tabla de DynamoDB, `ExampleTable`, y una clase `DataPoJo` que representa elementos de la tabla.

En este ejemplo de tabla, los atributos de clave principal se firmarán, pero no se cifrarán. Esto se aplica al `partition_attribute`, que se ha anotado con `@DynamoDBHashKey`, y el `sort_attribute`, que se ha anotado con `@DynamoDBRangeKey`.

Los atributos que son anotados con `@DynamoDBAttribute`, como `some numbers`, se cifrarán y firmarán. Las excepciones son atributos que utilizan las anotaciones de cifrado `@DoNotEncrypt` (solo firmar) o `@DoNotTouch` (no cifrar ni firmar) definidas por el cliente de cifrado de DynamoDB. Por ejemplo, ya que el atributo `leave me` tiene una anotación `@DoNotTouch`, no se cifrará ni se firmará.

```
@DynamoDBTable(tableName = "ExampleTable")
public static final class DataPoJo {
  private String partitionAttribute;
  private int sortAttribute;
  private String example;
  private long someNumbers;
  private byte[] someBinary;
  private String leaveMe;

  @DynamoDBHashKey(attributeName = "partition_attribute")
  public String getPartitionAttribute() {
    return partitionAttribute;
  }

  public void setPartitionAttribute(String partitionAttribute) {
    this.partitionAttribute = partitionAttribute;
  }

  @DynamoDBRangeKey(attributeName = "sort_attribute")
  public int getSortAttribute() {
    return sortAttribute;
  }

  public void setSortAttribute(int sortAttribute) {
    this.sortAttribute = sortAttribute;
  }

  @DynamoDBAttribute(attributeName = "example")
  public String getExample() {
    return example;
  }
```

```
  public void setExample(String example) {
    this.example = example;
  }

  @DynamoDBAttribute(attributeName = "some numbers")
  public long getSomeNumbers() {
    return someNumbers;
  }

  public void setSomeNumbers(long someNumbers) {
    this.someNumbers = someNumbers;
  }

  @DynamoDBAttribute(attributeName = "and some binary")
  public byte[] getSomeBinary() {
    return someBinary;
  }

  public void setSomeBinary(byte[] someBinary) {
    this.someBinary = someBinary;
  }

  @DynamoDBAttribute(attributeName = "leave me")
  @DoNotTouch
  public String getLeaveMe() {
    return leaveMe;
  }

  public void setLeaveMe(String leaveMe) {
    this.leaveMe = leaveMe;
  }

  @Override
  public String toString() {
    return "DataPoJo [partitionAttribute=" + partitionAttribute + ", sortAttribute="
        + sortAttribute + ", example=" + example + ", someNumbers=" + someNumbers
        + ", someBinary=" + Arrays.toString(someBinary) + ", leaveMe=" + leaveMe +
 "]";
  }
}
```

Paso 4: Cifrar y guardar un elemento de la tabla

Ahora, al crear un elemento de tabla y utilizar DynamoDB Mapper para guardarlo, el elemento se cifra automáticamente y firma antes de que se agregue a la tabla.

Este ejemplo define un elemento de tabla llamado `record`. Antes de que se guarde en la tabla, sus atributos se cifran y firman en función de las anotaciones de la clase `DataPoJo`. En este caso, todos los atributos salvo `PartitionAttribute`, `SortAttribute` y `LeaveMe` se cifran y se firman. `PartitionAttribute` y `SortAttributes` solo se firman. El atributo `LeaveMe` no está cifrado o firmado.

Para cifrar y firmar el elemento `recordy`, a continuación, añadirlo a `ExampleTable`, llame al método `save` de la clase `DynamoDBMapper`. Dado que el DynamoDB mapper está configurado para utilizar el PUT comportamiento de almacenamiento, el elemento sustituye a cualquier elemento con las mismas claves principales, en lugar de actualizarla. De este modo, se garantiza que las firmas coincidan y puede descifrar el elemento cuando se obtiene de la tabla.

```
DataPoJo record = new DataPoJo();
record.setPartitionAttribute("is this");
record.setSortAttribute(55);
record.setExample("data");
record.setSomeNumbers(99);
record.setSomeBinary(new byte[]{0x00, 0x01, 0x02});
record.setLeaveMe("alone");

mapper.save(record);
```

Cliente de cifrado de DynamoDB para Python

> **Note**
>
> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En el siguiente tema, se presenta información sobre las versiones 1.x—2.x del cliente de cifrado de DynamoDB para Java y versiones 1.x—3.x del cliente de cifrado de DynamoDB para Python. Para obtener más información, consulte el SDK de cifrado de bases de datos de AWS para la compatibilidad de la versión de DynamoDB.

En este tema, se explica cómo instalar y utilizar el cliente de cifrado de DynamoDB para en Python. Puede encontrar el código en el repositorio aws-dynamodb-encryption-python en GitHub, incluido código de muestra completo y probado como ayuda para comenzar.

> **Note**
>
> Versiones 1.x.x y 2.x.x del cliente de cifrado de DynamoDB para Python se encuentran en la fase de fin de soporte a partir de julio de 2022. Actualice a una versión más reciente tan pronto como sea posible.

Temas

Requisitos previos

Antes de instalar el Cliente de encriptación de Amazon DynamoDB para Python, asegúrese de que cumple los siguientes requisitos previos.

Una versión compatible de Python

El Cliente de encriptación de Amazon DynamoDB para las versiones 3.1.0 y posteriores de Python requiere Python 3.6 o posterior. Para descargar Python, visite el sitio de descargas de Python.

Las versiones anteriores del Cliente de encriptación de Amazon DynamoDB para Python admiten Python 2.7 y Python 3.4 y versiones posteriores, pero le recomendamos que utilice la versión más reciente del cliente de cifrado de DynamoDB.

La herramienta de instalación pip para Python

Python 3.6 y versiones posteriores incluyen pip, 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

Utilice pip para instalar el Cliente de encriptación de Amazon DynamoDB para Python, como se muestra en los siguientes ejemplos.

Para instalar la versión más reciente

```
pip install dynamodb-encryption-sdk
```

Para obtener más información acerca de cómo utilizar pip para instalar y actualizar paquetes, consulte la página sobre la instalación de paquetes.

El cliente de cifrado de DynamoDB requiere la biblioteca de criptografía en todas las plataformas. Todas las versiones de pip instalan y compilan la biblioteca cryptography en Windows. pip 8.1 y las versiones posteriores instalan y compilan la biblioteca cryptography en Linux. Si utiliza una versión anterior de pip y su entorno Linux no dispone de las herramientas necesarias para compilar la biblioteca cryptography, tiene que instalarlas. Para obtener más información, consulte Building cryptography on Linux.

Puede obtener la versión de desarrollo más reciente del cliente de cifrado de DynamoDB del repositorio aws-dynamodb-encryption-python en GitHub.

Después de instalar el cliente de cifrado de DynamoDB, comience examinando el código de Python de ejemplo de esta guía.

Uso del cliente de cifrado de DynamoDB para Python

> **Note**
>
> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En el siguiente tema, se presenta información sobre las versiones 1.x—2.x del cliente de cifrado de DynamoDB para Java y versiones 1.x—3.x del cliente de cifrado de DynamoDB para Python. Para obtener más información, consulte el SDK de cifrado de bases de datos de AWS para la compatibilidad de la versión de DynamoDB.

En este tema, se explican algunas de las características del cliente de cifrado de DynamoDB para Python que podrían no encontrarse en otras implementaciones de lenguaje de programación. Estas

características se han diseñado para facilitar el uso del cliente de cifrado de DynamoDB de la forma más segura. A menos que tenga un caso de uso inusual, le recomendamos que las utilice.

Para obtener información acerca de la programación con el cliente de cifrado de DynamoDB, consulte los ejemplos de Python en esta guía, los ejemplos del repositorio aws-dynamodb-encryption-python en GitHub y la documentación de Python para el cliente de cifrado de DynamoDB.

Temas

- Clases auxiliares de cliente
- Clase TableInfo
- Acciones de atributo en Python

Clases auxiliares de cliente

El cliente de cifrado de DynamoDB para Python incluye varias clases auxiliares de cliente, que interactúan con las clases de Boto 3 para DynamoDB. Estas clases auxiliares se han diseñado para facilitar agregar cifrado y firma a su aplicación de DynamoDB existente y evitar los problemas más habituales del siguiente modo:

- Impide que cifre la clave principal en su elemento, bien añadiendo una acción de anulación para la clave principal al objeto AttributeActions o generando una excepción si su objeto `AttributeActions` indica explícitamente al cliente que cifre la clave principal. Si la acción predeterminada en su objeto `AttributeActions` es `DO_NOTHING`, las clases auxiliares de cliente utilizan dicha acción para la clave principal. De lo contrario, utilizan `SIGN_ONLY`.
- Cree un objeto TableInfo y rellene el contexto de cifrado de DynamoDB en función de una llamada a DynamoDB. Esto ayuda a garantizar que el contexto de cifrado de DynamoDB sea exacto y el cliente pueda identificar la clave principal.
- Admite métodos, tales como `put_item` y `get_item`, que cifran y descifran de modo transparente los elementos de tabla al escribir o leer desde una tabla de DynamoDB. Solo el método `update_item` no se admite.

Puede utilizar las clases auxiliares de cliente en lugar de interactuar directamente con el encriptador de elementos de nivel inferior. Utilice estas clases a menos que tenga que establecer opciones avanzadas en el encriptador de elementos.

Las clases auxiliares de cliente incluyen:

- EncryptedTable para aplicaciones que utilizan el recurso Table en DynamoDB para procesar una tabla a la vez.
- EncryptedResource para aplicaciones que utilizan la clase Recurso de servicios en DynamoDB para el procesamiento por lotes.
- EncryptedClient para aplicaciones que utilizan el cliente de nivel inferior en DynamoDB.

Para usar las clases auxiliares del cliente, la persona que llama debe tener permiso para llamar a la operación DescribeTable de DynamoDB en la tabla de destino.

Clase TableInfo

La clase TableInfo es una clase auxiliar que representa una tabla de DynamoDB, completa con campos para su clave principal e índices secundarios. Le ayuda a obtener información precisa y en tiempo real sobre la tabla.

Si utiliza una clase auxiliar de cliente, crea y utiliza un objeto `TableInfo` automáticamente. De lo contrario, puede crear una explícitamente. Para ver un ejemplo, consulte Utilice el encriptador de elementos.

Cuando se llama al método `refresh_indexed_attributes` en un objeto `TableInfo`, se rellenan los valores de las propiedades del objeto mediante una llamada a la operación DescribeTable de DynamoDB. Consultar la tabla ofrece mucha más confianza que codificar de forma rígida los nombres de índice. La clase `TableInfo` incluye además una `encryption_context_values` propiedad que proporciona los valores requeridos para el contexto de cifrado de DynamoDB.

Para utilizar el método `refresh_indexed_attributes`, la persona que llama debe tener permiso para llamar a la operación DescribeTable de DynamoDB en la tabla de destino.

Acciones de atributo en Python

Las acciones de atributo indican al encriptador de elementos qué acciones hay que realizar en cada atributo del elemento. Para especificar acciones de atributo en Python, cree un objeto `AttributeActions` con una acción predeterminada y cualquier excepción para atributos particulares. Los valores válidos se definen en el tipo enumerado `CryptoAction`.

> **Important**
>
> Después de utilizar las acciones de atributo para cifrar los elementos de la tabla, agregar o quitar atributos del modelo de datos puede provocar un error de validación de firma que le

impide descifrar los datos. Para ver una explicación detallada, consulte Cambiar el modelo de datos.

```
DO_NOTHING = 0
SIGN_ONLY = 1
ENCRYPT_AND_SIGN = 2
```

Por ejemplo, este objeto `AttributeActions` establece `ENCRYPT_AND_SIGN` como predeterminado para todos los atributos y especifica excepciones para los atributos `ISBN` y `PublicationYear`.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={
        'ISBN': CryptoAction.DO_NOTHING,
        'PublicationYear': CryptoAction.SIGN_ONLY
    }
)
```

Si utiliza una clase auxiliar de cliente, no tiene que especificar una acción de atributo para los atributos de clave principal. Las clases auxiliares de cliente impiden que cifre su clave principal.

Si no utiliza una clase auxiliar de cliente y la acción predeterminada es `ENCRYPT_AND_SIGN`, debe especificar una acción para la clave principal. La acción recomendada para las claves principales es `SIGN_ONLY`. Para facilitarlo, utilice el método `set_index_keys`, que utiliza SIGN_ONLY para claves principales o DO_NOTHING, cuando esa es la acción predeterminada.

Warning

No cifre los atributos de clave principal. Deben permanecer en texto no cifrado para que DynamoDB pueda encontrar el elemento sin realizar un examen completo de la tabla.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
)
actions.set_index_keys(*table_info.protected_index_keys())
```

Código de ejemplo para el cliente de cifrado de DynamoDB para Python

> **Note**
>
> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En el siguiente tema, se presenta información sobre las versiones 1.x—2.x del cliente de cifrado de DynamoDB para Java y versiones 1.x—3.x del cliente de cifrado de DynamoDB para Python. Para obtener más información, consulte el SDK de cifrado de bases de datos de AWS para la compatibilidad de la versión de DynamoDB.

En los siguientes ejemplos, se muestra cómo utilizar el cliente de cifrado de DynamoDB para Python para proteger los datos de DynamoDB en su aplicación. Puede encontrar más ejemplos (y aportar los suyos propios) en el directorio de ejemplos del repositorio aws-dynamodb-encryption-python en GitHub.

Temas

- Utilizar la clase auxiliar de cliente EncryptedTable
- Utilice el encriptador de elementos

Utilizar la clase auxiliar de cliente EncryptedTable

El ejemplo siguiente le muestra cómo utilizar el proveedor de KMS directo con la `EncryptedTable` clase auxiliar cliente. Este ejemplo utiliza el mismo proveedor de materiales criptográficos que el ejemplo Utilice el encriptador de elementos siguiente. Sin embargo, utiliza la clase `EncryptedTable` en lugar de interactuar directamente con el encriptador de elementos de nivel inferior.

Comparando estos ejemplos, puede ver el trabajo que realiza la clase auxiliar cliente automáticamente. Esto incluye la creación del contexto de cifrado de DynamoDB y asegurarse de que los atributos de clave principal estén siempre firmados, pero nunca cifrados. Para crear el contexto de cifrado y descubrir la clave principal, las clases auxiliares del cliente llaman a la operación DescribeTable de DynamoDB. Para ejecutar este código, debe tener permiso para llamar a esta operación.

Vea la muestra de código completa: aws_kms_encrypted_table.py

Paso 1: crear la tabla

Empiece creando una instancia de una tabla de DynamoDB estándar con el nombre de la tabla.

```
table_name='test-table'
table = boto3.resource('dynamodb').Table(table_name)
```

Paso 2: crear un proveedor de materiales criptográficos

Cree una instancia del proveedor de materiales criptográficos (CMP) que ha seleccionado.

Este ejemplo utiliza el proveedor de KMS directo, pero puede utilizar cualquier CMP compatible. Para crear un proveedor de KMS directo, especifique un AWS KMS key. En este ejemplo, se utiliza el Nombre de recurso de Amazon (ARN) de la AWS KMS key, pero puede utilizar cualquier identificador de clave válido.

```
kms_key_id='arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key_id)
```

Paso 3: crear el objeto de acciones de atributo

Las acciones de atributo indican al encriptador de elementos qué acciones hay que realizar en cada atributo del elemento. El objeto `AttributeActions` de este ejemplo cifra y firma todos los elementos, excepto el atributo `test`, que se pasa por alto.

No especifique acciones de atributo para los atributos de clave principal cuando utilice una clase auxiliar cliente. La clase `EncryptedTable` firma, pero no cifra nunca, los atributos de clave principal.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={'test': CryptoAction.DO_NOTHING}
)
```

Paso 4: crear la tabla cifrada

Cree la tabla cifrada utilizando la tabla estándar, el proveedor de KMS directo y las acciones de atributo. Este paso completa la configuración.

```
encrypted_table = EncryptedTable(
```

```
    table=table,
    materials_provider=kms_cmp,
    attribute_actions=actions
)
```

Paso 5: colocar el elemento de texto no cifrado en la tabla

Cuando se llama al método `put_item` en la `encrypted_table`, los elementos de la tabla se cifran de modo transparente, se firman y se agrega a su tabla de DynamoDB.

Primero, defina el elemento de tabla.

```
plaintext_item = {
    'partition_attribute': 'value1',
    'sort_attribute': 55
    'example': 'data',
    'numbers': 99,
    'binary': Binary(b'\x00\x01\x02'),
    'test': 'test-value'
}
```

A continuación, colóquelo en la tabla.

```
encrypted_table.put_item(Item=plaintext_item)
```

Para obtener el elemento desde la tabla de DynamoDB en su forma cifrada, llame al método `get_item` en el objeto `table`. Para obtener el objeto descifrado, llame al método `get_item` en el objeto `encrypted_table`.

Utilice el encriptador de elementos

En este ejemplo, se muestra cómo interactuar directamente con el encriptador de elementos en la al cifrar elementos de tabla, en lugar de utilizar las clases auxiliares de cliente que interactúan con el encriptador de elementos.

Cuando se utiliza esta técnica, crea el contexto de cifrado de DynamoDB y el objeto de configuración (`CryptoConfig`) manualmente. Además, cifra los elementos en una llamada y los coloca en su tabla de DynamoDB en una llamada independiente. Esto le permite personalizar sus `put_item` llamadas y utilizar el cliente de cifrado de DynamoDB para cifrar y firmar datos estructurados que nunca se envían a DynamoDB.

Este ejemplo utiliza el proveedor de KMS directo, pero puede utilizar cualquier CMP compatible.

Vea la muestra de código completa: aws_kms_encrypted_item.py

Paso 1: crear la tabla

Empiece creando una instancia de un recurso de tabla de DynamoDB estándar con el nombre de la tabla.

```
table_name='test-table'
table = boto3.resource('dynamodb').Table(table_name)
```

Paso 2: crear un proveedor de materiales criptográficos

Cree una instancia del proveedor de materiales criptográficos (CMP) que ha seleccionado.

Este ejemplo utiliza el proveedor de KMS directo, pero puede utilizar cualquier CMP compatible. Para crear un proveedor de KMS directo, especifique un AWS KMS key. En este ejemplo, se utiliza el Nombre de recurso de Amazon (ARN) de la AWS KMS key, pero puede utilizar cualquier identificador de clave válido.

```
kms_key_id='arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key_id)
```

Paso 3: utilizar la clase auxiliar TableInfo

Para obtener información acerca de la tabla de DynamoDB, cree una instancia de la clase auxiliar TableInfo. Cuando trabaja directamente con el encriptador de elementos, tiene que crear una instancia `TableInfo` y llamar a sus métodos. Las clases auxiliares de cliente lo hacen automáticamente.

El `refresh_indexed_attributes` método de `TableInfo` utiliza la operación DescribeTable de DynamoDB para obtener información precisa en tiempo real sobre la tabla. Incluye su clave principal y sus índices secundarios locales y globales. El intermediario tiene que tener permiso para llamar a `DescribeTable`.

```
table_info = TableInfo(name=table_name)
table_info.refresh_indexed_attributes(table.meta.client)
```

Paso 4: crear el contexto de cifrado de DynamoDB

El contexto de cifrado de DynamoDB contiene información acerca de la estructura de la tabla y cómo se cifra y se firma. En este ejemplo, se crea un contexto de cifrado de DynamoDB explícitamente, porque interactúa con el encriptador de elementos. Las clases auxiliares de cliente crean el contexto de cifrado de DynamoDB para usted.

Para obtener la clave de partición y la clave de clasificación, puede utilizar las propiedades de la clase auxiliar TableInfo.

```
index_key = {
    'partition_attribute': 'value1',
    'sort_attribute': 55
}

encryption_context = EncryptionContext(
    table_name=table_name,
    partition_key_name=table_info.primary_index.partition,
    sort_key_name=table_info.primary_index.sort,
    attributes=dict_to_ddb(index_key)
)
```

Paso 5: crear el objeto de acciones de atributo

Las acciones de atributo indican al encriptador de elementos qué acciones hay que realizar en cada atributo del elemento. El objeto `AttributeActions` en este ejemplo cifra y firma todos los elementos, excepto los atributos de clave principal, que se firman, pero no se cifran y el atributo `test`, que se pasa por alto.

Cuando se interactúa directamente con el encriptador de elementos y la acción predeterminada es `ENCRYPT_AND_SIGN`, debe especificar una acción alternativa para la clave principal. Puede utilizar el método `set_index_keys`, que usa `SIGN_ONLY` para la clave principal o utiliza `DO_NOTHING` si es la acción predeterminada.

Para especificar la clave principal, en este ejemplo, se utilizan las claves de índice en el objeto TableInfo, que se rellena mediante una llamada a DynamoDB. Esta técnica es más segura que los nombres de clave principal de codificación rígida.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
```

```
    attribute_actions={'test': CryptoAction.DO_NOTHING}
)
actions.set_index_keys(*table_info.protected_index_keys())
```

Paso 6: crear la configuración para el elemento

Para configurar el cliente de cifrado de DynamoDB, utilice los objetos que acaba de crear en una configuración CryptoConfig para el elemento de tabla. Las clases auxiliares de cliente crean la CryptoConfig automáticamente.

```
crypto_config = CryptoConfig(
    materials_provider=kms_cmp,
    encryption_context=encryption_context,
    attribute_actions=actions
)
```

Paso 7: cifrar el elemento

En este paso, se cifra y firma el elemento, pero no lo coloca en la tabla de DynamoDB.

Cuando utiliza una clase auxiliar de cliente, sus elementos se cifran y se firman de modo transparente y, a continuación, se agregan a su tabla de DynamoDB cuando llama al `put_item` método de la clase auxiliar. Cuando utiliza el encriptador de elementos directamente, las acciones de cifrado y colocación son independientes.

En primer lugar, cree un elemento de texto no cifrado.

```
plaintext_item = {
    'partition_attribute': 'value1',
    'sort_key': 55,
    'example': 'data',
    'numbers': 99,
    'binary': Binary(b'\x00\x01\x02'),
    'test': 'test-value'
}
```

A continuación, cífrelo y fírmelo. El método `encrypt_python_item` requiere el objeto de configuración `CryptoConfig`.

```
encrypted_item = encrypt_python_item(plaintext_item, crypto_config)
```

Paso 8: colocar el elemento en la tabla

En este paso, se coloca el elemento cifrado y firmado en la tabla de DynamoDB.

```
table.put_item(Item=encrypted_item)
```

Para ver el elemento cifrado, llame al método `get_item` en el objeto `table` original, en lugar del objeto `encrypted_table`. Obtiene el elemento de la tabla DynamoDB sin verificarlo y descifrarlo.

```
encrypted_item = table.get_item(Key=partition_key)['Item']
```

En la imagen siguiente se muestra una parte de un elemento de tabla cifrado y firmado de ejemplo.

Los valores de atributo cifrados son datos binarios. Los nombres y los valores de los atributos de clave principal (`partition_attribute` y `sort_attribute`) y el atributo `test` permanecen en texto no cifrado. La salida muestra además el atributo que contiene la firma (`*amzn-ddb-map-sig*`) y el atributo de descripción de materiales (`*amzn-ddb-map-desc*`).

```
{
    '*amzn-ddb-map-desc*': Binary(b'\x00\x00\x00\x00\x00\x00\x00\x10amzn-ddb-env-alg\
\x00\x00\x00\xe0AQEBAHhA84wnXjEJdBbBBylRUFcZZK2j7xwh6UyLoL28nQ
+0FAAAAH4wfAYJKoZIhvcNAQcGoG8wbQIBADBoBgkqhkiG9w0BBwEwHgYJYIZIAWUDBAEuMBEEDPeFBydmoJD
izYl0R0C4M7wAK6E1/N/bgTmHI=\x00\x00\x00\x17amzn-ddb-map-signingAlg\x00\x00\x00\nHmacS
\x00\x00\x00\x11/CBC/PKCS5Padding\x00\x00\x00\x10amzn-ddb-sig-alg\x00\x00\x00\x0eHmac
\x00\x00\x00\x0faws-kms-ec-attr\x00\x00\x00\x06*keys*'),
    '*amzn-ddb-map-sig*': Binary(b"\xd3\xc6\xc7\n\xb7#\x13\xd1Y\xea\xe4.|^\xbd\xdf\xe
    'binary': Binary(b'!"\xc5\x92\xd7\x13\x1d\xe8Bs\x9b\x7f\xa8\x8e\x9c\xcf\x10\x1e\x
    'example': Binary(b"'b\x933\x9a+s\xf1\xd6a\xc5\xd5\x1aZ\xed\xd6\xce\xe9X\xf0T\xcb
    'numbers': Binary(b'\xd5\xa0\\d\xcc\x85\xf5\x1e\xb9-f!\xb9\xb8\x8a\x1aT\xbaq\xf7\
    'partition_attribute': 'value1',
    'sort_attribute': 55,
    'test': 'test-value'
}
```

Cambiar el modelo de datos

> **Note**
>
> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En el siguiente tema, se presenta información sobre las versiones 1.x—2.x del cliente de cifrado de DynamoDB para Java y versiones 1.x—3.x del cliente de cifrado de

DynamoDB para Python. Para obtener más información, consulte el SDK de cifrado de bases de datos de AWS para la compatibilidad de la versión de DynamoDB.

Cada vez que cifra o descifra un elemento, tiene que proporcionar acciones de atributo ??? que indiquen al DynamoDB qué atributos cifrar y firmar, qué atributos firmar (pero no cifrar) y cuáles omitir. Las acciones de atributo no se guardan en el elemento cifrado y no actualiza automáticamente las acciones de atributo.

Important

El cliente de cifrado de DynamoDB no admite el cifrado de datos de tablas de DynamoDB existentes y no cifrados.

Cada vez que cambie el modelo de datos, es decir, cuando agregue o quite atributos de los elementos de la tabla, corre el riesgo de que se produzca un error. Si las acciones de atributo que especifique no cuentan para todos los atributos del elemento, el elemento podría no cifrarse y firmarse del modo previsto. Lo que es más importante, si las acciones de atributo que proporciona al descifrar un elemento difieren de las acciones de atributo que proporcione al cifrar el elemento, la verificación de la firma podría fallar.

Por ejemplo, si las acciones de atributo utilizadas para cifrar el elemento indican que se firme el atributo `test`, la firma en el elemento incluirá el atributo `test`. Pero, si las acciones de atributo utilizadas para descifrar el elemento no se tienen en cuenta para el atributo `test`, la verificación devolverá un error, ya que el cliente intentará verificar una firma que no incluye el atributo `test`.

Este es un problema particular cuando varias aplicaciones leen y escriben los mismos elementos de porque debe calcular la misma firma para los elementos en todas las aplicaciones. También es un problema para cualquier aplicación distribuida porque los cambios en las acciones de atributos deben propagarse a todos los hosts. Incluso si un host accede a sus tablas de en un proceso, establecer un proceso de prácticas recomendadas ayudará a evitar errores si el proyecto se vuelve más complejo.

Para evitar errores de validación de firmas que le impidan leer los elementos de la tabla, siga las instrucciones siguientes.

- Añadir un atributo: si el nuevo atributo cambia sus acciones de atributo, implemente completamente el cambio de acción de atributo antes de incluir el nuevo atributo en un elemento.

- Eliminar un atributo: si dejas de usar un atributo en tus artículos, no cambies las acciones de los atributos.
- Cambiar la acción: después de haber utilizado una configuración de acciones de atributos para cifrar los elementos de la tabla, no podrá cambiar de forma segura la acción predeterminada o la acción de un atributo existente sin volver a cifrar todos los elementos de la tabla.

Los errores de validación de firmas pueden ser extremadamente difíciles de resolver, por lo que el mejor enfoque es prevenirlos.

Temas

- Adición de un atributo
- Eliminación de un atributo

Adición de un atributo

Al agregar un nuevo atributo a los elementos de tabla, es posible que tenga que cambiar las acciones de atributo. Para evitar errores de validación de firmas, se recomienda implementar este cambio en un proceso de dos etapas. Verifique que la primera etapa esté completa antes de comenzar la segunda etapa.

1. Cambie las acciones de atributo en todas las aplicaciones que leen o escriben en la tabla. Implemente estos cambios y confirme que la actualización se ha propagado a todos los hosts de destino.
2. Escriba valores en el nuevo atributo de los elementos de la tabla.

Este enfoque de dos etapas garantiza que todas las aplicaciones y hosts tengan las mismas acciones de atributo y calculará la misma firma antes de que cualquier otro encuentre el nuevo atributo. Esto es importante incluso cuando la acción del atributo es Do nothing (no cifrar ni firmar), porque el valor predeterminado de algunos encriptadores es cifrar y firmar.

Los siguientes ejemplos muestran el código de la primera etapa de este proceso. Agregan un nuevo atributo de elemento, `link`, que almacena un vínculo a otro elemento de tabla. Dado que este vínculo debe permanecer como texto sin formato, el ejemplo le asigna la acción de solo firma. Después de implementar completamente este cambio y comprobar que todas las aplicaciones y hosts tienen las nuevas acciones de atributo, puede comenzar a usar el atributo `link` en los elementos de tabla.

Java DynamoDB Mapper

Cuando usa `DynamoDB Mapper` y `AttributeEncryptor`, todos los atributos están cifrados y firmados por defecto, excepto las claves principales, que están firmadas pero no cifradas. Para especificar una acción de solo firma, utilice la anotación `@DoNotEncrypt`.

En este ejemplo se utiliza la anotación `@DoNotEncrypt` para el nuevo atributo `link`.

```
@DynamoDBTable(tableName = "ExampleTable")
public static final class DataPoJo {
  private String partitionAttribute;
  private int sortAttribute;
  private String link;

  @DynamoDBHashKey(attributeName = "partition_attribute")
  public String getPartitionAttribute() {
    return partitionAttribute;
  }

  public void setPartitionAttribute(String partitionAttribute) {
    this.partitionAttribute = partitionAttribute;
  }

  @DynamoDBRangeKey(attributeName = "sort_attribute")
  public int getSortAttribute() {
    return sortAttribute;
  }

  public void setSortAttribute(int sortAttribute) {
    this.sortAttribute = sortAttribute;
  }

  @DynamoDBAttribute(attributeName = "link")
  @DoNotEncrypt
  public String getLink() {
    return link;
  }

  public void setLink(String link) {
    this.link = link;
  }

  @Override
```

```
  public String toString() {
    return "DataPoJo [partitionAttribute=" + partitionAttribute + ",
        sortAttribute=" + sortAttribute + ",
        link=" + link + "]";
  }
}
```

Java DynamoDB encryptor

En el encriptador de nivel inferior, debe establecer acciones para cada atributo. En este ejemplo se utiliza una instrucción switch donde el valor predeterminado es `encryptAndSign` y se especifican excepciones para la clave de partición, la clave de clasificación y el nuevo atributo `link`. En este ejemplo, si el código de atributo de vínculo no se implementó completamente antes de utilizarlo, algunas aplicaciones cifrarían y firmarían el atributo de vínculo, pero solo lo firmarían otras.

```
for (final String attributeName : record.keySet()) {
    switch (attributeName) {
        case partitionKeyName:
            // fall through to the next case
        case sortKeyName:
            // partition and sort keys must be signed, but not encrypted
            actions.put(attributeName, signOnly);
            break;
        case "link":
            // only signed
            actions.put(attributeName, signOnly);
            break;
        default:
            // Encrypt and sign all other attributes
            actions.put(attributeName, encryptAndSign);
            break;
    }
}
```

Python

En DynamoDB para Python, puede especificar una acción predeterminada para todos los atributos y, a continuación, especificar excepciones.

Si utiliza una clase auxiliar de cliente de Python, no tiene que especificar una acción de atributo para los atributos de clave principal. Las clases auxiliares de cliente impiden que cifre su clave

principal. Sin embargo, si no está utilizando una clase auxiliar de cliente, debe establecer la acción SIGN_ONLY en la clave de partición y la clave de clasificación. Si accidentalmente cifra la partición o la clave de clasificación, no podrá recuperar los datos sin un análisis completo de la tabla.

En este ejemplo se especifica una excepción para el nuevo atributo `link`, que obtiene la acción `SIGN_ONLY`.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={
      'example': CryptoAction.DO_NOTHING,
      'link': CryptoAction.SIGN_ONLY
    }
)
```

Eliminación de un atributo

Si ya no necesita un atributo en los elementos que se han cifrado con , puede dejar de usar el atributo. Sin embargo, no elimine ni cambie la acción de ese atributo. Si lo hace y, a continuación, encuentra un elemento con ese atributo, la firma calculada para el artículo no coincidirá con la firma original y la validación de la firma fallará.

Aunque podría tener la tentación de eliminar todos los rastros del atributo del código, agregue un comentario de que el elemento ya no se usa en lugar de eliminarlo. Incluso si realiza un análisis de tabla completo para eliminar todas las instancias del atributo, un elemento cifrado con ese atributo podría almacenarse en caché o estar en proceso en algún lugar de la configuración.

Solución de problemas en la aplicación DynamoDB Encryption Client

> **Note**
>
> Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En el siguiente tema, se presenta información sobre las versiones 1.x—2.x del cliente de cifrado de DynamoDB para Java y versiones 1.x—3.x del cliente de cifrado de DynamoDB para Python. Para obtener más información, consulte el SDK de cifrado de bases de datos de AWS para la compatibilidad de la versión de DynamoDB.

En esta sección se describen los problemas que podría encontrar al utilizar el y se ofrecen sugerencias para resolverlos.

Para enviar comentarios sobre el cliente de cifrado DynamoDB, registre un problema en el repositorio aws-dynamodb-encryption-java o aws-dynamodb-encryption-python de GitHub.

Para enviar comentarios sobre esta documentación, utilice el enlace de comentarios de cualquier página. También puede registrar un problema o contribuir a aws-dynamodb-encryption-docs, el repositorio de código abierto para esta documentación de GitHub.

Temas

Acceso denegado

Problema: su aplicación ha denegado el acceso a un recurso que la necesita.

Sugerencia: obtenga información acerca de los permisos requeridos y agréguelos al contexto de seguridad en el que se ejecuta su aplicación.

Detalles

Para ejecutar una aplicación que usa una biblioteca de , el intermediario debe tener permiso para utilizar sus componentes. De lo contrario, se les denegará el acceso a los elementos requeridos.

- El cliente de cifrado de DynamoDB no requiere una cuenta de Amazon Web Services (AWS) ni depende de ningún servicio. AWS No obstante, si su aplicación utiliza AWS, necesita un Cuenta de AWS y usuarios que tengan permiso para utilizar la cuenta.
- El cliente de cifrado de DynamoDB no requiere Amazon DynamoDB. Sin embargo, si la aplicación que utiliza el cliente crea tablas de DynamoDB, coloca elementos en una tabla u obtiene elementos de una tabla, el intermediario debe tener permiso para utilizar las operaciones de DynamoDB requeridas en su Cuenta de AWS. Para obtener más información, consulte los temas de control de acceso en la Guía para desarrolladores de Amazon DynamoDB.

- Si la aplicación utiliza una clase auxiliar de cliente en el cliente de cifrado de DynamoDB para Python, la persona que llama debe tener permiso para llamar a la operación DescribeTable de DynamoDB.
- El cliente de cifrado de DynamoDB no AWS Key Management Service requiere (). AWS KMS Sin embargo, si la aplicación utiliza un proveedor de materiales de Direct KMS o utiliza un proveedormás reciente con un almacén de proveedores que lo utiliceAWS KMS, la persona que llama debe tener permiso para utilizar las operaciones AWS KMSGenerateDataKey y Decrypt.

Errores de verificación de firma

Problema: un elemento no se puede descifrar porque la verificación de firma devuelve un error. El elemento podría no estar cifrado y firmado del modo previsto.

Sugerencia: asegúrese de que las acciones de atributos que proporcione cuenten para todos los atributos del elemento. Al descifrar un elemento, asegúrese de proporcionar acciones de atributo que coincidan con las acciones utilizadas para cifrar el elemento.

Detalles

Las acciones de atributo que proporciona indican qué atributos cifrar y firmar, qué atributos firmar (pero no cifrar) y cuáles ignorar.

Si las acciones de atributo que especifique no cuentan para todos los atributos del elemento, el elemento podría no cifrarse y firmarse del modo previsto. Si las acciones de atributo que proporciona al descifrar un elemento difieren de las acciones de atributo que proporcione al cifrar el elemento, la verificación de la firma podría fallar. Se trata de un problema particular para aplicaciones distribuidas en las que las nuevas acciones de atributos podrían no haberse propagado a todos los hosts.

Los errores de validación de firmas son difíciles de resolver. Para ayudar a prevenirlos, tome precauciones adicionales al cambiar el modelo de datos. Para obtener más información, consulte Cambiar el modelo de datos.

Problemas con las tablas globales de versiones anteriores

Problema: los elementos de una tabla global de Amazon DynamoDB de una versión anterior no se pueden descifrar porque no se puede comprobar la firma.

Sugerencia: defina las acciones de los atributos de forma que los campos de replicación reservados no estén cifrados ni firmados.

Detalles

Puede utilizar el cliente de cifrado de DynamoDB con las tablas globales de DynamoDB. Se recomienda utilizar tablas globales con una clave KMS de múltiples regiones y replicar la clave KMS en todos los Regiones de AWS lugares donde esté replicada la tabla global.

A partir de la versión 2019.11.21 de tablas globales, puede utilizarlas con el cliente de cifrado de DynamoDB sin ninguna configuración especial. Sin embargo, si utiliza tablas globales de la versión 2017.11.29, debe asegurarse de que los campos de replicación reservados no estén cifrados ni firmados.

Si utiliza las tablas globales de la versión 2017.11.29, debe configurar las acciones de atributo para los siguientes atributos `DO_NOTHING` en `@DoNotTouch`Java o Python.

- `aws:rep:deleting`
- `aws:rep:updatetime`
- `aws:rep:updateregion`

Si utiliza cualquier otra versión de las tablas globales, no es necesario realizar ninguna acción.

Rendimiento deficiente del proveedor más reciente

Problema: la aplicación responde menos, especialmente después de actualizarse a una versión más reciente del cliente de cifrado de DynamoDB.

Sugerencia: ajuste el tiempo de vida útil y el tamaño de la memoria caché.

Detalles

El proveedor más reciente está diseñado para mejorar el rendimiento de las aplicaciones que utilizan el cliente de cifrado de DynamoDB al permitir una reutilización limitada del material criptográfico. Al configurar el proveedor más reciente para su aplicación, debe equilibrar la mejora del rendimiento con los problemas de seguridad que se derivan del almacenamiento en caché y la reutilización.

En las versiones más recientes del cliente de cifrado de DynamoDB, el valor del tiempo de vida (TTL) determina durante cuánto tiempo se pueden utilizar los proveedores de material criptográfico (CMP) en caché. El TTL también determina la frecuencia con la que el proveedor más reciente comprueba si hay una nueva versión del CMP.

Si su TTL es demasiado largo, su aplicación podría infringir sus normas empresariales o normas de seguridad. Si tu TTL es demasiado breve, las llamadas frecuentes a la tienda del proveedor pueden provocar que la tienda del proveedor limite las solicitudes de tu aplicación y de otras aplicaciones que comparten tu cuenta de servicio. Para resolver este problema, ajusta el TTL y el tamaño de la caché a un valor que cumpla tus objetivos de latencia y disponibilidad y que se ajuste a tus estándares de seguridad. Para obtener más información, consulte Establecer un valor de tiempo de vida.

Cambio de nombre del Cliente de encriptación de Amazon DynamoDB

Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

El 9 de junio de 2023, nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de base de datos de AWS. El SDK de cifrado de bases de datos de AWS es compatible con Amazon DynamoDB. Puede descifrar y leer elementos cifrados por el cliente de cifrado de DynamoDB heredado. Para obtener más información sobre las versiones heredadas de DynamoDB Encryption Client, consulte Compatibilidad con la versión SDK de cifrado de bases de datos de AWS para DynamoDB.

El SDK de cifrado de bases de datos de AWS incluye la versión 3.x de la biblioteca de cifrado del cliente de Java para DynamoDB, que es una reescritura importante del cliente de cifrado de DynamoDB para Java. Incluye numerosas actualizaciones, como un nuevo formato de datos estructurados, una compatibilidad mejorada de multitenencia, cambios de esquema fluidos y compatibilidad con el cifrado para búsquedas.

Para obtener más información sobre las nuevas características introducidas con el SDK para el cifrado de bases de datos de AWS, consulte los siguientes temas.

Cifrado para búsquedas

Puede diseñar bases de datos que puedan buscar registros cifrados sin tener que descifrar toda la base de datos. Según el modelo de amenazas y los requisitos de consulta, puede utilizar el cifrado para búsquedas para realizar búsquedas de coincidencias exactas o consultas complejas más personalizadas en sus registros cifrados.

Llaveros

El SDK de cifrado de bases de datos de AWS utiliza llaveros para realizar el cifrado de sobre. Los llaveros generan, cifran y descifran las claves de datos que protegen sus registros. El SDK de cifrado de bases de datos de AWS admite llaveros AWS KMS que utilizan cifrado simétrico o RSA asimétrico AWS KMS keyspara proteger las claves de datos y llaveros jerárquicos AWS

KMS que le permiten proteger sus materiales criptográficos mediante una clave KMS de cifrado simétrico sin tener que llamar AWS KMS cada vez que cifra o descifra un registro. También puede especificar su propio material de claves con los llaveros Raw AES y Raw RSA.

Cambios de esquema sin complicaciones

Al configurar el SDK de cifrado de bases de datos de AWS, se proporcionan acciones criptográficas que indican al cliente qué campos debe cifrar y firmar, qué campos debe firmar (pero no cifrar) y cuáles debe ignorar. Una vez que haya utilizado el SDK de cifrado de bases de datos de AWS para proteger sus registros, podrá seguir realizando cambios en el modelo de datos. Puede actualizar sus acciones criptográficas, como agregar o eliminar campos cifrados, en una sola implementación.

Configurar las tablas de DynamoDB existentes para el cifrado del cliente

Las versiones heredadas del cliente de cifrado de DynamoDB se diseñaron para implementarse en tablas nuevas y despobladas. Con el SDK de cifrado de bases de datos de AWS para DynamoDB, puede migrar las tablas de Amazon DynamoDB existentes a la versión 3.x de la biblioteca de cifrado del cliente de Java para DynamoDB.

Referencia

Nuestra biblioteca de cifrado del cliente pasó a llamarse SDK de cifrado de bases de datos de AWS. En esta guía para desarrolladores, se sigue proporcionando información sobre el cliente de cifrado de DynamoDB.

En los temas siguientes, se proporcionan detalles técnicos del SDK de cifrado de bases de datos de AWS.

Formato de descripción del material

La descripción del material sirve como encabezado de un registro cifrado. Al cifrar y firmar campos con el SDK de cifrado de bases de datos de AWS, el encriptador registra la descripción del material a medida que reúne los materiales criptográficos y almacena la descripción del material en un nuevo campo (`aws_dbe_head`) que el encriptador agrega al registro. La descripción del material es una estructura de datos con formato portátil que contiene la clave de datos cifrados e información sobre cómo se cifró y firmó el registro. En la siguiente tabla, se describen los valores que forman la descripción del material. Los bytes se anexan en el orden mostrado

Valor	Longitud en bytes
Versión	1
Firmas habilitadas	1
ID de registro	32
Cifra la leyenda	Variable
Longitud del contexto de cifrado	2
Contexto de cifrado	Variable
Encrypted Data Key Count	1
Claves de datos cifrados	Variable

Valor	Longitud en bytes
Compromiso de registro	1

Versión

La versión de este `aws_dbe_head` formato de campo.

Firmas habilitadas

Codifica si las firmas están habilitadas para este registro.

Valor de byte	Significado
`0x01`	Firmas habilitadas (predeterminado)
`0x00`	Firmas deshabilitadas

ID de registro

Valor de 256-bits generado de manera aleatoria que identifica el registro. El ID del registro:

- Identifica de forma única el registro cifrado.
- Vincula la descripción del material al registro cifrado.

Cifra la leyenda

Una descripción serializada de los campos autenticados que se cifraron. La leyenda de cifrado se utiliza para determinar qué campos debe intentar descifrar el método de descifrado.

Valor del byte	Significado
`0x65`	`ENCRYPT_AND_SIGN`
`0x73`	`SIGN_ONLY`

La leyenda de cifrado se serializa de la siguiente manera:

1. Lexicográficamente mediante la secuencia de bytes que representa su ruta canónica.

2. Para cada campo, en orden, agregue uno de los valores de bytes especificados anteriormente para indicar si ese campo debe cifrarse.

Longitud del contexto de cifrado

La longitud del contenido cifrado. Se trata de un valor de 2 bytes interpretado como un entero sin signo de 16 bits. La longitud máxima es de 65.535 bytes.

Contexto de cifrado

Un conjunto de pares de nombre-valor que contienen datos autenticados adicionales no secretos y arbitrarios.

Cuando las firmas digitales están habilitadas, el contexto de cifrado contiene el par clave-valor `{"aws-crypto-footer-ecdsa-key": Qtxt}`. `Qtxt`representa el punto de la curva elíptica `Q` comprimido según la versión 2.0 de la SEC 1 y, a continuación, codificado en base64.

Encrypted Data Key Count

El número de claves de datos cifradas. Se trata de un valor de 1-byte interpretado como un entero sin signo de 8-bits que especifica el número de claves de datos cifradas. El número máximo de claves de datos cifrados en cada registro es 255.

Claves 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 en 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

Campo	Longitud en bytes
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).

Key Provider ID Length

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.

Key Provider ID

El identificador del proveedor de claves. Se utiliza para indicar el proveedor de la clave de datos cifrada y está previsto que sea extensible.

Key Provider Information Length

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.

Key Provider Information

La información del proveedor de claves. Viene determinada por el proveedor de claves.

Cuando está utilizando un AWS KMSllavero, este valor contiene el Nombre de recurso de Amazon (ARN) de la AWS KMS key.

Encrypted Data Key Length

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.

Encrypted Data Key

La clave de datos cifrada. Se trata de la clave de datos cifrada por el proveedor de claves.

Compromiso de registro

Un hash distinto del código de autenticación de mensajes basado en hash (HMAC) de 256-bits que se calcula sobre todos los bytes de descripción del material anteriores mediante la clave de compromiso.

AWS KMS Detalles técnicos del llavero jerárquico

El AWS KMS llavero jerárquico utiliza una clave de datos única para cifrar cada campo 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 empaquetado 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 llavero jerárquico utiliza la clave de empaquetado 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 empaquetado 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
El identificador de la clave de la rama	Variable	Codificado con UTF-8

Valor	Longitud en bytes	Interpretado como
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 desarrolladores del SDK de cifrado de bases de datos de AWS

En la siguiente tabla se describen cambios significativos de esta documentación. Además de estos cambios importantes, también actualizamos la documentación con frecuencia para mejorar las descripciones y los ejemplos y para dar cuenta de los comentarios que nos envía. Para recibir notificaciones sobre cambios significativos, suscríbase al canal RSS.

Cambio	Descripción	Fecha
Versión de disponibilidad general (GA)	Documentación actualizada para la versión GA de la versión 3.x de la biblioteca de cifrado del cliente de Java para DynamoDB. **Warning** Las claves de rama creadas durante la versión preliminar para desarrolladores ya no son compatibles.	24 de julio de 2023
Cambio de marca del cliente de cifrado de DynamoDB	La biblioteca de cifrado del cliente pasa a llamarse SDK de cifrado de bases de datos de AWS.	9 de junio de 2023
Versión de prueba	Se agregó y actualizó la documentación para la versión 3.x de la biblioteca de cifrado del cliente de Java para DynamoDB, que incluye un nuevo formato de datos estructurados, compatibilidad	9 de junio de 2023

	mejorada de multitenencia, cambios de esquema sin problemas y compatibilidad con cifrado para búsquedas.	
Cambio de documentación	Sustituir el AWS Key Management Service término clave maestra del cliente (CMK) con AWS KMS key y clave KMS.	30 de agosto de 2021
Nueva característica	Se agregó compatibilidad con claves AWS Key Managemen t Service (AWS KMS) de múltiples regiones. Las claves multirregiones son AWS KMS claves en diferentes Regiones de AWS que se pueden usar indistintamente porque tienen el mismo ID de clave y el mismo material de claves.	8 de junio de 2021
Nuevo ejemplo	Se ha añadido un ejemplo de cómo usar DynamoDBMapper en Java.	6 de septiembre de 2018
Soporte de Python	Se ha agregado soporte para Python, además de Java.	2 de mayo de 2018
Versión inicial	Versión inicial de esta documentación.	2 de mayo de 2018

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.