aws

Entwicklerhandbuch

AWS SDK für Datenbankverschlüsselung

AWS SDK für Datenbankverschlüsselung: Entwicklerhandbuch

Table of Contents

Was ist der AWS SDK für Datenbankverschlüsselung?

Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt . AWS Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Das AWS Database Encryption SDK besteht aus einer Reihe von Softwarebibliotheken, mit denen Sie die clientseitige Verschlüsselung in Ihr Datenbankdesign integrieren können. Das AWS Database Encryption SDK bietet Verschlüsselungslösungen auf Datensatzebene. Sie geben an, welche Felder verschlüsselt sind und welche Felder in den Signaturen enthalten sind, die die Authentizität Ihrer Daten sicherstellen. Durch die Verschlüsselung Ihrer sensiblen Daten während der Übertragung und im Speicher wird sichergestellt, dass Ihre Klartextdaten nicht für Dritte verfügbar sind, einschließlich. AWS Das AWS Database Encryption SDK wird kostenlos unter der Apache 2.0-Lizenz bereitgestellt.

Dieses Entwicklerhandbuch bietet einen konzeptionellen Überblick über das AWS Database Encryption SDK, einschließlich einer Einführung in seine Architektur, Einzelheiten darüber, wie es Ihre Daten schützt, wie es sich von serverseitiger Verschlüsselung unterscheidet, und Anleitungen zur Auswahl kritischer Komponenten für Ihre Anwendung, um Ihnen den Einstieg zu erleichtern.

Das AWS Database Encryption SDK unterstützt Amazon DynamoDB mit Verschlüsselung auf Attributebene.

Das AWS Database Encryption SDK bietet die folgenden Vorteile:

Speziell für Datenbankanwendungen entwickelt

Sie müssen kein Kryptografie-Experte sein, um das AWS Database Encryption SDK verwenden zu können. Die Implementierungen beinhalten Hilfsmethoden, die so konzipiert sind, dass sie mit Ihren vorhandenen Anwendungen funktionieren.

Nachdem Sie die erforderlichen Komponenten erstellt und konfiguriert haben, verschlüsselt und signiert der Verschlüsselungsclient Ihre Datensätze transparent, wenn Sie sie einer Datenbank hinzufügen, und verifiziert und entschlüsselt sie, wenn Sie sie abrufen.

Beinhaltet sichere Verschlüsselung und Signierung

Das AWS Database Encryption SDK umfasst sichere Implementierungen, die die Feldwerte in jedem Datensatz mit einem eindeutigen Datenverschlüsselungsschlüssel verschlüsseln und den

Datensatz anschließend signieren, um ihn vor unbefugten Änderungen wie dem Hinzufügen oder Löschen von Feldern oder dem Austauschen verschlüsselter Werte zu schützen.

Verwendet kryptographisches Material aus beliebigen Quellen

Das AWS Database Encryption SDK verwendet Schlüsselringe, um den eindeutigen Datenverschlüsselungsschlüssel zu generieren, zu verschlüsseln und zu entschlüsseln, der Ihren Datensatz schützt. Schlüsselringe bestimmen die Umschließungsschlüssel, mit denen dieser Datenschlüssel verschlüsselt wird.

Sie können Schlüssel aus jeder beliebigen Quelle einschließen, einschließlich Kryptografiediensten wie AWS Key Management Service()AWS KMS oder. AWS CloudHSM Das AWS Database Encryption SDK benötigt keinen AWS-Konto oder keinen AWS Dienst.

Support für das Zwischenspeichern kryptografischer Materialien

Der AWS KMS hierarchische Schlüsselbund ist eine Caching-Lösung für kryptografisches Material, die die Anzahl der AWS KMS Aufrufe reduziert, indem AWS KMS geschützte Branch-Schlüssel verwendet werden, die in einer Amazon DynamoDB-Tabelle gespeichert sind, und anschließend das für Ver- und Entschlüsselungsvorgänge verwendete Zweigschlüsselmaterial lokal zwischenspeichert. Damit können Sie Ihre kryptografischen Materialien mit einem KMS-Schlüssel mit symmetrischer Verschlüsselung schützen, ohne jedes Mal, wenn Sie einen Datensatz ver- oder entschlüsseln, erneut aufrufen zu müssen. AWS KMS Der AWS KMS hierarchische Schlüsselbund ist eine gute Wahl für Anwendungen, bei denen die Anzahl der Aufrufe minimiert werden muss. AWS KMS

Durchsuchbare Verschlüsselung

Sie können Datenbanken entwerfen, die verschlüsselte Datensätze durchsuchen können, ohne die gesamte Datenbank zu entschlüsseln. Abhängig von Ihrem Bedrohungsmodell und Ihren Abfrageanforderungen können Sie eine durchsuchbare Verschlüsselung verwenden, um Suchen nach exakten Treffern oder individuellere komplexe Abfragen in Ihrer verschlüsselten Datenbank durchzuführen.

Support für mehrinstanzenfähige Datenbankschemas

Mit dem AWS Database Encryption SDK können Sie Daten schützen, die in Datenbanken mit einem gemeinsamen Schema gespeichert sind, indem Sie jeden Mandanten mit unterschiedlichen Verschlüsselungsmaterialien isolieren. Wenn mehrere Benutzer Verschlüsselungsvorgänge in Ihrer Datenbank durchführen, verwenden Sie einen der AWS KMS Schlüsselbunde, um jedem Benutzer einen eigenen Schlüssel zur Verfügung zu stellen, den er für seine

kryptografischen Operationen verwenden kann. Weitere Informationen finden Sie unter Arbeiten mit Mehrmandantendatenbanken.

Support für nahtlose Schemaaktualisierungen

Wenn Sie das AWS Database Encryption SDK konfigurieren, stellen Sie kryptografische Aktionen bereit, die dem Client mitteilen, welche Felder verschlüsselt und signiert, welche Felder signiert (aber nicht verschlüsselt) und welche ignoriert werden sollen. Nachdem Sie das AWS Database Encryption SDK zum Schutz Ihrer Datensätze verwendet haben, können Sie immer noch Änderungen an Ihrem Datenmodell vornehmen. Sie können Ihre kryptografischen Aktionen, wie das Hinzufügen oder Entfernen verschlüsselter Felder, in einer einzigen Bereitstellung aktualisieren.

Entwickelt in Open-Source-Repositorys

Das AWS Database Encryption SDK wurde in Open-Source-Repositorien am entwickelt. GitHub Sie können diese Repositorys verwenden, um den Code einzusehen, Probleme zu lesen und zu melden sowie Informationen zu finden, die für Ihre Implementierung spezifisch sind.

Das Tool AWS Datenbankverschlüsselungs-SDK für DynamoDB

- Das aws-database-encryption-sdk-dynamodb-Repository auf GitHub unterstützt die neuesten Versionen des AWS Database Encryption SDK für DynamoDB in Java, .NET und Rust.

  Das AWS Database Encryption SDK für DynamoDB ist ein Produkt von Dafny, einer überprüfungsfähigen Sprache, in der Sie Spezifikationen, den Code zu ihrer Implementierung und die Beweise, um sie zu testen, schreiben. Das Ergebnis ist eine Bibliothek, die die Funktionen des AWS Database Encryption SDK für DynamoDB in einem Framework implementiert, das die funktionale Korrektheit gewährleistet.

Support und Wartung

Das AWS Database Encryption SDK verwendet dieselbe Wartungsrichtlinie wie das AWS SDK und die Tools, einschließlich der Versionierungs- und Lebenszyklusphasen. Als bewährte Methode empfehlen wir, dass Sie die neueste verfügbare Version des AWS Database Encryption SDK für Ihre Datenbankimplementierung verwenden und ein Upgrade durchführen, sobald neue Versionen veröffentlicht werden. Der Lebenszyklus und die Supportphase jeder Version können je nach Programmiersprache variieren. Beispielsweise kann sich eine bestimmte Version des AWS

Database Encryption SDK in einer Programmiersprache in der Phase der allgemeinen Verfügbarkeit (vollständiger Support) befinden, in einer anderen Programmiersprache jedoch in der Endphase des Supports. Informationen zur Lebenszyklusphase der AWS Database Encryption SDK-Versionen für Ihre Programmiersprache finden Sie in der POLICY.rst Datei SUPPORT_ im AWS Database Encryption SDK-Repository.

Weitere Informationen finden Sie in den Wartungsrichtlinien für AWS SDKs und Tools im Referenzhandbuch für AWS SDKs und Tools.

Senden von Feedback

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

Wenn Sie eine potenzielle Sicherheitslücke im AWS Database Encryption SDK entdecken, benachrichtigen AWS Sie bitte den Sicherheitsdienst. Erstellen Sie kein öffentliches GitHub Problem.

Über den auf jeder Seite angezeigten Feedback-Link können Sie Feedback zu dieser Dokumentation bereitstellen.

Melde dich für eine an AWS-Konto

Um loszulegen AWS, benötigen Sie eine AWS-Konto. Informationen zum Erstellen eines AWS-Konto finden Sie unter Erste Schritte mit einem AWS-Konto im AWS -Kontenverwaltung Referenzhandbuch.

AWS SDK-Konzepte für Datenbankverschlüsselung

> Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

In diesem Thema werden die im AWS Database Encryption SDK verwendeten Konzepte und Terminologie erläutert.

Informationen zum Zusammenspiel der Komponenten des AWS Database Encryption SDK finden Sie unterSo funktioniert das AWS Database Encryption SDK.

Weitere Informationen zum AWS Database Encryption SDK finden Sie in den folgenden Themen.

- Erfahren Sie, wie das AWS Database Encryption SDK Umschlagverschlüsselung verwendet, um Ihre Daten zu schützen.
- Erfahren Sie mehr über die Elemente der Umschlagverschlüsselung: die Datenschlüssel, die Ihre Datensätze schützen, und die Umhüllungsschlüssel, die Ihre Datenschlüssel schützen.
- Erfahren Sie mehr über die Schlüsselanhänger, die bestimmen, welche Verpackungsschlüssel Sie verwenden.
- Erfahren Sie mehr über den Verschlüsselungskontext, der Ihrem Verschlüsselungsprozess Integrität verleiht.
- Erfahren Sie mehr über die Materialbeschreibung, die die Verschlüsselungsmethoden Ihrem Datensatz hinzufügen.
- Erfahren Sie mehr über die kryptografischen Aktionen, die dem AWS Database Encryption SDK mitteilen, welche Felder verschlüsselt und signiert werden sollen.

Themen

Umschlagverschlüsselung

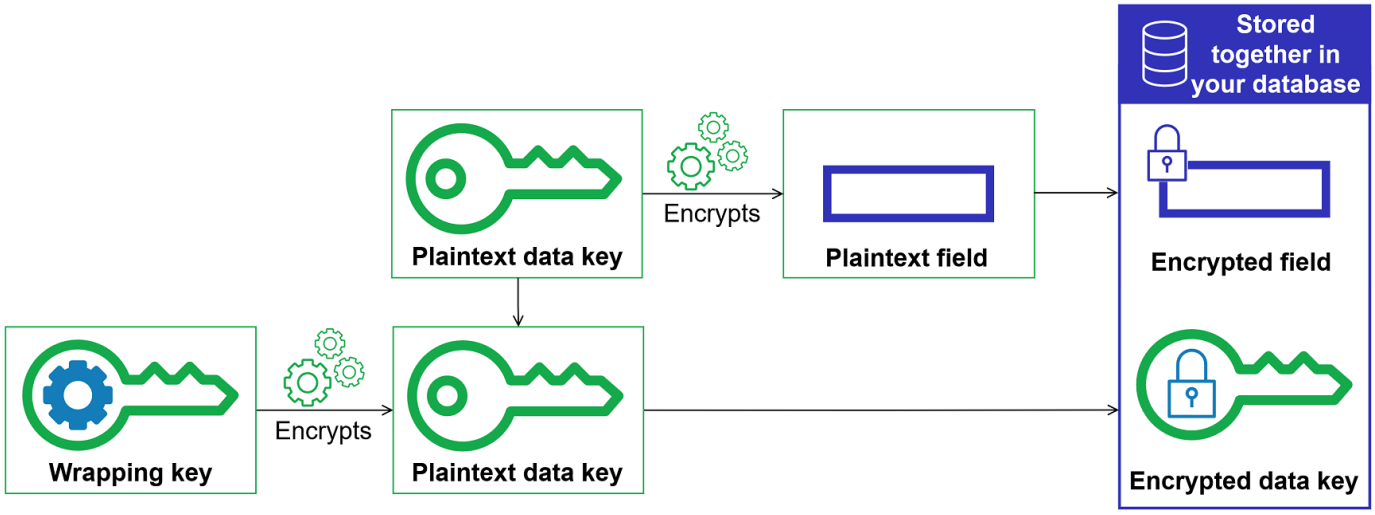
Die Sicherheit Ihrer verschlüsselten Daten hängt teilweise vom Schutz des Datenschlüssels ab, der sie entschlüsseln kann. Eine akzeptierte bewährte Methode zum Schutz des Datenschlüssels ist seine Verschlüsselung. Dazu benötigen Sie einen weiteren Verschlüsselungsschlüssel, einen sogenannten Schlüsselverschlüsselungsschlüssel oder Wrapping-Schlüssel. Die Praxis,

Datenschlüssel mit einem Wrapping-Schlüssel zu verschlüsseln, wird als Envelope-Verschlüsselung bezeichnet.

Schutz von Datenschlüsseln

Das AWS Database Encryption SDK verschlüsselt jedes Feld mit einem eindeutigen Datenschlüssel. Anschließend verschlüsselt es jeden Datenschlüssel unter dem von Ihnen angegebenen Wrapping-Schlüssel. Es speichert die verschlüsselten Datenschlüssel in der Materialbeschreibung.

Um Ihren Verpackungsschlüssel anzugeben, verwenden Sie einen Schlüsselbund.



Verschlüsseln derselben Daten unter mehreren Wrapping-Schlüsseln

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

Um die Daten zu entschlüsseln, müssen Sie mindestens einen Wrapping-Schlüssel angeben, mit dem die verschlüsselten Datenschlüssel entschlüsselt werden können.

Kombination der Stärken mehrerer Algorithmen

Um Ihre Daten zu verschlüsseln, verwendet das AWS Database Encryption SDK standardmäßig eine Algorithmussuite mit symmetrischer AES-GCM-Verschlüsselung, einer HMAC-basierten Schlüsselableitungsfunktion (HKDF) und ECDSA-Signatur. Um den

Datenschlüssel zu verschlüsseln, können Sie einen symmetrischen oder asymmetrischen Verschlüsselungsalgorithmus angeben, der zu Ihrem Wrapping-Schlüssel passt.

Im Allgemeinen sind symmetrische Schlüsselverschlüsselungsalgorithmen schneller und erzeugen kleinere Verschlüsselungstexte als eine asymmetrische Verschlüsselung oder eine Verschlüsselung mit öffentlichem Schlüssel. Algorithmen mit öffentlichen Schlüsseln bieten jedoch eine inhärente Rollentrennung. Um die Stärken der beiden zu kombinieren, können Sie den Datenschlüssel mit einer Verschlüsselung mit öffentlichen Schlüsseln verschlüsseln.

Wir empfehlen, wann immer möglich einen der AWS KMS Schlüsselringe zu verwenden. Wenn Sie den AWS KMS Schlüsselbund verwenden, können Sie die Stärken mehrerer Algorithmen kombinieren, indem Sie einen asymmetrischen RSA AWS KMS key als Umschließungsschlüssel angeben. Sie können auch einen KMS-Schlüssel für die symmetrische Verschlüsselung verwenden.

Datenschlüssel

Ein Datenschlüssel ist ein Verschlüsselungsschlüssel, den das AWS Database Encryption SDK verwendet, um die Felder in einem Datensatz zu verschlüsseln, die `ENCRYPT_AND_SIGN` in den kryptografischen Aktionen markiert sind. Jeder Datenschlüssel ist ein Byte-Array, das die Anforderungen für kryptografische Schlüssel erfüllt. Das AWS Database Encryption SDK verwendet einen eindeutigen Datenschlüssel, um jedes Attribut zu verschlüsseln.

Sie müssen Datenschlüssel nicht spezifizieren, generieren, implementieren, erweitern, schützen oder verwenden. Das AWS Database Encryption SDK erledigt das für Sie, wenn Sie die Verschlüsselungs- und Entschlüsselungsvorgänge aufrufen.

Um Ihre Datenschlüssel zu schützen, verschlüsselt das AWS Database Encryption SDK sie mit einem oder mehreren Schlüsselverschlüsselungsschlüsseln, den sogenannten Wrapping Keys. Nachdem das AWS Database Encryption SDK Ihre Klartext-Datenschlüssel verwendet hat, um Ihre Daten zu verschlüsseln, werden sie so schnell wie möglich aus dem Speicher entfernt. Speichert dann den verschlüsselten Datenschlüssel in der Materialbeschreibung. Details hierzu finden Sie unter So funktioniert das AWS Database Encryption SDK.

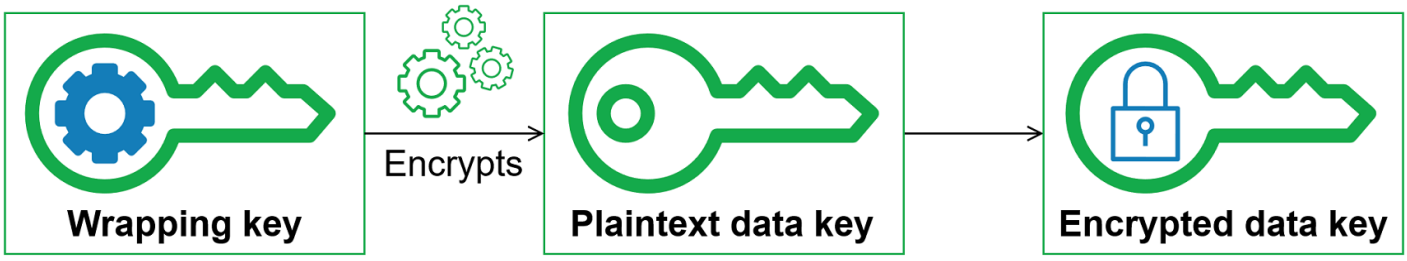
> **Tip**
>
> Im AWS Database Encryption SDK unterscheiden wir Datenschlüssel von Datenverschlüsselungsschlüsseln. Als bewährte Methode verwenden alle unterstützten Algorithmus-Suiten eine Funktion zur Schlüsselableitung. Die Schlüsselableitungsfunktion

verwendet einen Datenschlüssel als Eingabe und gibt die Datenverschlüsselungsschlüssel zurück, die tatsächlich zur Verschlüsselung Ihrer Datensätze verwendet werden. Aus diesem Grund sagen wir oft, dass die Daten „unter“ einem Datenschlüssel verschlüsselt werden, statt „von“ dem Datenschlüssel.

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

Schlüssel zum Umschließen

Ein Wrapping Key ist ein Schlüssel zur Verschlüsselung, den das AWS Database Encryption SDK verwendet, um den Datenschlüssel zu verschlüsseln, der Ihre Datensätze verschlüsselt. Jeder Datenschlüssel kann mit einem oder mehreren Umschließungsschlüsseln verschlüsselt werden. Bei der Konfiguration eines Schlüsselbunds legen Sie fest, welche Umschließungsschlüssel zum Schutz Ihrer Daten verwendet werden.



Das AWS Database Encryption SDK unterstützt mehrere häufig verwendete Wrapping-Schlüssel, wie z. B. AWS Key Management Service(AWS KMS) KMS-Schlüssel mit symmetrischer Verschlüsselung (einschließlich Schlüssel für mehrere Regionen) und asymmetrische AWS KMS RSA-KMS-Schlüssel, AES-GCM-Rohschlüssel (Advanced Encryption Standard/Galois Counter Mode) und RSA-Rohschlüssel. Wir empfehlen, wann immer möglich KMS-Schlüssel zu verwenden. Informationen zur Entscheidung, welchen Wrapping-Schlüssel Sie verwenden sollten, finden Sie unter Auswählen von Wrapping-Schlüsseln.

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

- Verwenden Sie einen Dienst, der für diesen Zweck entwickelt wurde, z. B. AWS Key Management Service (AWS KMS).
- Verwenden Sie ein Hardwaresicherheitsmodul (HSM), wie z. B. die Angebote von AWS CloudHSM.

- Verwenden Sie andere wichtige Verwaltungstools und -dienste.

Wenn Sie kein Schlüsselverwaltungssystem haben, empfehlen wir AWS KMS. Das AWS Database Encryption SDK lässt sich integrieren AWS KMS , damit Sie Ihre Wrapping-Schlüssel schützen und verwenden können.

Schlüsselanhänger

Um die Wrapping-Schlüssel anzugeben, die Sie für die Verschlüsselung und Entschlüsselung verwenden, verwenden Sie einen Schlüsselbund. Sie können die Schlüsselbunde verwenden, die das AWS Database Encryption SDK bereitstellt, oder Ihre eigenen Implementierungen entwerfen.

Ein Schlüsselbund generiert, verschlüsselt und entschlüsselt Datenschlüssel. Es generiert auch die MAC-Schlüssel, die zur Berechnung der Hash-Based Message Authentication Codes (HMACs) in der Signatur verwendet werden. Wenn Sie einen Schlüsselbund definieren, können Sie die Wrapping-Schlüssel angeben, mit denen Ihre Datenschlüssel verschlüsselt werden. Die meisten Schlüsselbunde spezifizieren mindestens einen Umschließungsschlüssel oder einen Dienst, der Schlüssel zum Umschließen bereitstellt und schützt. Bei der Verschlüsselung verwendet das AWS Database Encryption SDK alle im Schlüsselbund angegebenen Umschließungsschlüssel, um den Datenschlüssel zu verschlüsseln. Hilfe zur Auswahl und Verwendung der Schlüsselbunde, die das AWS Database Encryption SDK definiert, finden Sie unter Schlüsselbunde verwenden.

Kryptografische Aktionen

Kryptografische Aktionen teilen dem Verschlüsseler mit, welche Aktionen für jedes Feld in einem Datensatz ausgeführt werden sollen.

Bei den kryptografischen Aktionswerten kann es sich um einen der folgenden Werte handeln:

- Verschlüsseln und signieren — Verschlüsseln Sie das Feld. Schließt das verschlüsselte Feld in die Signatur ein.
- Nur signieren — Schließt das Feld in die Signatur ein.
- Signieren und in den Verschlüsselungskontext aufnehmen — Schließt das Feld in den Signatur- und Verschlüsselungskontext ein.

  Standardmäßig sind die Partitions- und Sortierschlüssel das einzige Attribut, das im Verschlüsselungskontext enthalten ist. Sie könnten erwägen, zusätzliche Felder zu definieren, `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` damit der Anbieter der Branch-Schlüssel-ID

für Ihren AWS KMS hierarchischen Schlüsselbund ermitteln kann, welcher Filialschlüssel für die Entschlüsselung aus dem Verschlüsselungskontext erforderlich ist. Weitere Informationen finden Sie unter Lieferant für die Filialschlüssel-ID.

> **Note**
>
> Um die `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` kryptografische Aktion verwenden zu können, müssen Sie Version 3.3 oder höher des AWS Database Encryption SDK verwenden. Stellen Sie die neue Version für alle Lesegeräte bereit, bevor Sie Ihr Datenmodell so aktualisieren, dass es diese enthält`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

- Nichts tun — Verschlüsseln Sie das Feld nicht und nehmen Sie es nicht in die Signatur auf.

Verwenden Sie für jedes Feld, in dem vertrauliche Daten gespeichert werden können, die Option Verschlüsseln und Signieren. Verwenden Sie für Primärschlüsselwerte (z. B. einen Partitionsschlüssel und einen Sortierschlüssel in einer DynamoDB-Tabelle) im Verschlüsselungskontext nur signieren oder Signieren und einschließen. Wenn Sie Attribute vom Typ „Signieren“ und „Im Verschlüsselungskontext einschließen“ angeben, müssen auch die Partitions- und Sortierattribute „Im Verschlüsselungskontext signieren und einbeziehen“ lauten. Sie müssen keine kryptografischen Aktionen für die Materialbeschreibung angeben. Das AWS Database Encryption SDK signiert automatisch das Feld, in dem die Materialbeschreibung gespeichert ist.

Wählen Sie Ihre kryptografischen Aktionen sorgfältig aus. Verwenden Sie im Zweifelsfall Verschlüsseln und signieren. Nachdem Sie das AWS Database Encryption SDK zum Schutz Ihrer Datensätze verwendet haben, können Sie weder ein vorhandenes `ENCRYPT_AND_SIGN` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Feld noch die einem vorhandenen `DO_NOTHING` Feld zugewiesene kryptografische Aktion ändern. `SIGN_ONLY` `DO_NOTHING` Sie können jedoch weiterhin andere Änderungen an Ihrem Datenmodell vornehmen. Sie können beispielsweise verschlüsselte Felder in einer einzigen Bereitstellung hinzufügen oder entfernen.

Materialbeschreibung

Die Materialbeschreibung dient als Header für einen verschlüsselten Datensatz. Wenn Sie Felder mit dem AWS Database Encryption SDK verschlüsseln und signieren, zeichnet der Verschlüsseler die Materialbeschreibung auf, während er die kryptografischen Materialien zusammenstellt, und speichert die Materialbeschreibung in einem neuen Feld (`aws_dbe_head`), das der Verschlüsseler Ihrem Datensatz hinzufügt.

Bei der Materialbeschreibung handelt es sich um eine übertragbare, formatierte Datenstruktur, die verschlüsselte Kopien der Datenschlüssel und andere Informationen wie Verschlüsselungsalgorithmen, Verschlüsselungskontext sowie Verschlüsselungs- und Signieranweisungen enthält. Der Verschlüsseler zeichnet die Materialbeschreibung auf, während er die kryptografischen Materialien für die Verschlüsselung und Signierung zusammenstellt. Wenn er später kryptografisches Material zusammenstellen muss, um ein Feld zu verifizieren und zu entschlüsseln, verwendet er die Materialbeschreibung als Leitfaden.

Wenn Sie die verschlüsselten Datenschlüssel zusammen mit dem verschlüsselten Feld speichern, wird der Entschlüsselungsvorgang optimiert und Sie müssen keine verschlüsselten Datenschlüssel unabhängig von den Daten, die sie verschlüsseln, speichern und verwalten.

Technische Informationen zur Materialbeschreibung finden Sie unter. Format der Materialbeschreibung

Verschlüsselungskontext

Um die Sicherheit Ihrer kryptografischen Operationen zu verbessern, enthält das AWS Database Encryption SDK in allen Anfragen zum Verschlüsseln und Signieren eines Datensatzes einen Verschlüsselungskontext.

Ein Verschlüsselungskontext ist eine Gruppe von Name-Wert-Paaren mit willkürlichen, nicht geheimen, zusätzlich authentifizierten Daten. Das AWS Database Encryption SDK enthält den logischen Namen für Ihre Datenbank und Primärschlüsselwerte (z. B. einen Partitionsschlüssel und einen Sortierschlüssel in einer DynamoDB-Tabelle) im Verschlüsselungskontext. Wenn Sie ein Feld verschlüsseln und signieren, ist der Verschlüsselungskontext kryptografisch an den verschlüsselten Datensatz gebunden, sodass derselbe Verschlüsselungskontext erforderlich ist, um das Feld zu entschlüsseln.

Wenn Sie einen AWS KMS Schlüsselbund verwenden, verwendet das AWS Database Encryption SDK auch den Verschlüsselungskontext, um zusätzliche authentifizierte Daten (AAD) in den Aufrufen des Schlüsselbunds bereitzustellen. AWS KMS

Immer wenn Sie die Standard-Algorithmus-Suite verwenden, fügt der Cryptographic Materials Manager (CMM) dem Verschlüsselungskontext ein Name-Wert-Paar hinzu, das aus einem reservierten Namen und einem Wert besteht`aws-crypto-public-key`, der den öffentlichen Bestätigungsschlüssel darstellt. Der öffentliche Bestätigungsschlüssel wird in der Materialbeschreibung gespeichert.

Manager von kryptographischen Materialien

Der Cryptographic Materials Manager (CMM) stellt die kryptografischen Materialien zusammen, die zum Verschlüsseln, Entschlüsseln und Signieren Ihrer Daten verwendet werden. Wann immer Sie die Standardalgorithmussuite verwenden, umfassen die kryptografischen Materialien Klartext- und verschlüsselte Datenschlüssel, symmetrische Signaturschlüssel und einen asymmetrischen Signaturschlüssel. Sie interagieren nie direkt mit dem CMM. Die Ver- und Entschlüsselungsmethoden übernehmen das für Sie.

Da das CMM als Bindeglied zwischen dem AWS Database Encryption SDK und einem Schlüsselbund fungiert, ist es ein idealer Ort für Anpassungen und Erweiterungen, z. B. zur Unterstützung der Richtliniendurchsetzung. Sie können ein CMM explizit angeben, dies ist jedoch nicht erforderlich. Wenn Sie einen Schlüsselbund angeben, erstellt das AWS Database Encryption SDK ein Standard-CMM für Sie. Das Standard-CMM ruft die Ver- oder Entschlüsselungsmaterialien aus dem von Ihnen angegebenen Schlüsselbund ab. Dabei könnte es sich um einen Aufruf eines kryptographischen Dienstes handeln, z. B. AWS Key Management Service (AWS KMS).

Symmetrische und asymmetrische Verschlüsselung

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

Asymmetrische Verschlüsselung verwendet ein mathematisch verwandtes Datenschlüsselpaar. Ein Schlüssel des Paares verschlüsselt die Daten; nur der andere Schlüssel im Paar kann die Daten entschlüsseln.

Das AWS Database Encryption SDK verwendet Umschlagverschlüsselung. Es verschlüsselt Ihre Daten mit einem symmetrischen Datenschlüssel. Es verschlüsselt den symmetrischen Datenschlüssel mit einem oder mehreren symmetrischen oder asymmetrischen Wrapping-Schlüsseln. Es fügt dem Datensatz eine Materialbeschreibung hinzu, die mindestens eine verschlüsselte Kopie des Datenschlüssels enthält.

Verschlüsselung Ihrer Daten (symmetrische Verschlüsselung)

> Um Ihre Daten zu verschlüsseln, verwendet das AWS Database Encryption SDK einen symmetrischen Datenschlüssel und eine Algorithmussuite, die einen symmetrischen Verschlüsselungsalgorithmus enthält. Um die Daten zu entschlüsseln, verwendet das AWS Database Encryption SDK denselben Datenschlüssel und dieselbe Algorithmus-Suite.

Verschlüsselung Ihres Datenschlüssels (symmetrische oder asymmetrische Verschlüsselung)

Der Schlüsselbund, den Sie für einen Verschlüsselungs- und Entschlüsselungsvorgang angeben, bestimmt, wie der symmetrische Datenschlüssel ver- und entschlüsselt wird. Sie können einen Schlüsselbund wählen, der symmetrische Verschlüsselung verwendet, z. B. einen AWS KMS Schlüsselbund mit einem symmetrischen Verschlüsselungs-KMS-Schlüssel, oder einen Schlüsselbund, der asymmetrische Verschlüsselung verwendet, z. B. einen Schlüsselbund mit einem asymmetrischen RSA-KMS-Schlüssel. AWS KMS

Wichtiges Engagement

Das AWS Database Encryption SDK unterstützt Key Commitment (manchmal auch als Robustheit bezeichnet), eine Sicherheitseigenschaft, die sicherstellt, dass jeder Chiffretext nur in einen einzigen Klartext entschlüsselt werden kann. Zu diesem Zweck stellt Key Commitment sicher, dass nur der Datenschlüssel, mit dem Ihr Datensatz verschlüsselt wurde, zum Entschlüsseln verwendet wird. Das AWS Database Encryption SDK beinhaltet Key Commitment für alle Verschlüsselungs- und Entschlüsselungsvorgänge.

Die meisten modernen symmetrischen Chiffren (einschließlich AES) verschlüsseln Klartext unter einem einzigen geheimen Schlüssel, wie dem eindeutigen Datenschlüssel, den das AWS Database Encryption SDK verwendet, um jedes in einem Datensatz markierte Klartextfeld zu verschlüsseln. `ENCRYPT_AND_SIGN` Beim Entschlüsseln dieses Datensatzes mit demselben Datenschlüssel wird ein Klartext zurückgegeben, der mit dem Original identisch ist. Die Entschlüsselung mit einem anderen Schlüssel schlägt normalerweise fehl. Obwohl schwierig, ist es technisch möglich, einen Chiffretext unter zwei verschiedenen Schlüsseln zu entschlüsseln. In seltenen Fällen ist es möglich, einen Schlüssel zu finden, der Chiffretext teilweise in einen anderen, aber dennoch verständlichen Klartext entschlüsseln kann.

Das AWS Database Encryption SDK verschlüsselt jedes Attribut immer unter einem eindeutigen Datenschlüssel. Es kann diesen Datenschlüssel unter mehreren Umschließungsschlüsseln verschlüsseln, aber die Umschließungsschlüssel verschlüsseln immer denselben Datenschlüssel. Dennoch kann ein ausgeklügelter, manuell erstellter verschlüsselter Datensatz tatsächlich unterschiedliche Datenschlüssel enthalten, von denen jeder mit einem anderen Umschließungsschlüssel verschlüsselt ist. Wenn beispielsweise ein Benutzer den verschlüsselten Datensatz entschlüsselt, gibt er 0x0 (falsch) zurück, während ein anderer Benutzer, der denselben verschlüsselten Datensatz entschlüsselt, 0x1 (wahr) erhält.

Um dieses Szenario zu verhindern, beinhaltet das AWS Database Encryption SDK wichtige Verpflichtungen beim Verschlüsseln und Entschlüsseln. Bei der Verschlüsselungsmethode wird der eindeutige Datenschlüssel, der den Chiffretext erzeugt hat, kryptografisch mit der Schlüsselzusage verknüpft. Dabei handelt es sich um einen Hash-Based Message Authentication Code (HMAC), der anhand der Materialbeschreibung anhand einer Ableitung des Datenschlüssels berechnet wird. Anschließend wird die Schlüsselzusage in der Materialbeschreibung gespeichert. Wenn es einen Datensatz mit Schlüsselzusage entschlüsselt, überprüft das AWS Database Encryption SDK, ob der Datenschlüssel der einzige Schlüssel für diesen verschlüsselten Datensatz ist. Wenn die Überprüfung des Datenschlüssels fehlschlägt, schlägt der Entschlüsselungsvorgang fehl.

Digitale Signaturen

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

Um dieses Problem zu vermeiden, fügt die Standard-Algorithmus-Suite verschlüsselten Datensätzen eine ECDSA-Signatur (Elliptic Curve Digital Signature Algorithm) hinzu. Die Standard-Algorithmus-Suite verschlüsselt die Felder in Ihrem Datensatz, die mit einem authentifizierten Verschlüsselungsalgorithmus, `ENCRYPT_AND_SIGN` AES-GCM, markiert sind. Anschließend berechnet sie sowohl Hash-basierte Nachrichtenauthentifizierungscodes (HMACs) als auch asymmetrische ECDSA-Signaturen für die mit, und markierten Felder in Ihrem Datensatz. `ENCRYPT_AND_SIGN` `SIGN_ONLY` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Bei der Entschlüsselung wird anhand der Signaturen überprüft, ob ein autorisierter Benutzer den Datensatz verschlüsselt hat.

Wenn die Standard-Algorithmus-Suite verwendet wird, generiert das AWS Database Encryption SDK für jeden verschlüsselten Datensatz ein temporäres Paar aus privatem Schlüssel und öffentlichem Schlüssel. Das AWS Database Encryption SDK speichert den öffentlichen Schlüssel in der Materialbeschreibung und verwirft den privaten Schlüssel. Dadurch wird sichergestellt, dass niemand eine weitere Signatur erstellen kann, die mit dem öffentlichen Schlüssel verifiziert wird. Der Algorithmus bindet den öffentlichen Schlüssel als zusätzliche authentifizierte Daten in der

Materialbeschreibung an den verschlüsselten Datenschlüssel und verhindert so, dass Benutzer, die nur Felder entschlüsseln können, den öffentlichen Schlüssel ändern oder die Signaturüberprüfung beeinträchtigen.

Das AWS Database Encryption SDK beinhaltet immer die HMAC-Verifizierung. Digitale ECDSA-Signaturen sind standardmäßig aktiviert, aber nicht erforderlich. Wenn die Benutzer, die Daten verschlüsseln, und die Benutzer, die Daten entschlüsseln, gleichermaßen vertrauenswürdig sind, sollten Sie die Verwendung einer Algorithmussuite in Betracht ziehen, die keine digitalen Signaturen enthält, um Ihre Leistung zu verbessern. Weitere Informationen zur Auswahl alternativer Algorithmus-Suiten finden Sie unter Auswahl einer Algorithmus-Suite.

> **Note**
>
> Wenn ein Schlüsselbund nicht zwischen Verschlüsselern und Entschlüsselern unterscheidet, bieten digitale Signaturen keinen kryptografischen Wert.

AWS KMS Schlüsselbunde, einschließlich des asymmetrischen RSA-Schlüsselbunds, können auf der Grundlage von AWS KMS Schlüssel- und IAM-Richtlinien zwischen Verschlüsselern und Entschlüsselern unterscheiden. AWS KMS

Aufgrund ihres kryptografischen Charakters können die folgenden Schlüsselbunde nicht zwischen Verschlüsselern und Entschlüsselern unterscheiden:

- AWS KMS Hierarchischer Schlüsselbund
- AWS KMS ECDH-Schlüsselanhänger
- Unformatierter AES-Schlüsselbund
- Unformatierter RSA-Schlüsselbund
- Roher ECDH-Schlüsselanhänger

So funktioniert das AWS Database Encryption SDK

Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt . AWS Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Das AWS Database Encryption SDK bietet clientseitige Verschlüsselungsbibliotheken, die speziell für den Schutz der Daten entwickelt wurden, die Sie in Datenbanken speichern. Die Bibliotheken enthalten sichere Implementierungen, die Sie erweitern oder unverändert verwenden können. Weitere Informationen zur Definition und Verwendung benutzerdefinierter Komponenten finden Sie im GitHub Repository für Ihre Datenbankimplementierung.

In den Workflows in diesem Abschnitt wird erklärt, wie das AWS Database Encryption SDK die Daten in Ihrer Datenbank verschlüsselt, signiert und entschlüsselt und verifiziert. Diese Workflows beschreiben den grundlegenden Prozess unter Verwendung abstrakter Elemente und der Standardfunktionen. Einzelheiten dazu, wie das AWS Database Encryption SDK mit Ihrer Datenbankimplementierung zusammenarbeitet, finden Sie im Thema Was ist verschlüsselt für Ihre Datenbank.

Das AWS Database Encryption SDK verwendet Umschlagverschlüsselung, um Ihre Daten zu schützen. Jeder Datensatz wird unter einem eindeutigen Datenschlüssel verschlüsselt. Der Datenschlüssel wird verwendet, um einen eindeutigen Datenverschlüsselungsschlüssel für jedes Feld abzuleiten, das `ENCRYPT_AND_SIGN` in Ihren kryptografischen Aktionen markiert ist. Anschließend wird eine Kopie des Datenschlüssels mit den von Ihnen angegebenen Wrapping-Schlüsseln verschlüsselt. Um den verschlüsselten Datensatz zu entschlüsseln, verwendet das AWS Database Encryption SDK die von Ihnen angegebenen Wrapping-Schlüssel, um mindestens einen verschlüsselten Datenschlüssel zu entschlüsseln. Anschließend kann es den Chiffretext entschlüsseln und einen Klartexteintrag zurückgeben.

Weitere Hinweise zu den im AWS Database Encryption SDK verwendeten Begriffen finden Sie unter. AWS SDK-Konzepte für Datenbankverschlüsselung

Verschlüsseln und signieren

Im Kern ist das AWS Database Encryption SDK ein Datensatzverschlüsseler, der die Datensätze in Ihrer Datenbank verschlüsselt, signiert, verifiziert und entschlüsselt. Es enthält Informationen über Ihre Datensätze und Anweisungen darüber, welche Felder verschlüsselt und signiert werden müssen. Es ruft die Verschlüsselungsmaterialien und Anweisungen zu ihrer Verwendung von einem Manager für kryptografische Materialien ab, der anhand des von Ihnen angegebenen Verpackungsschlüssels konfiguriert wurde.

In der folgenden exemplarischen Vorgehensweise wird beschrieben, wie das AWS Database Encryption SDK Ihre Dateneinträge verschlüsselt und signiert.

1. Der Cryptographic Materials Manager stellt dem AWS Database Encryption SDK eindeutige Datenverschlüsselungsschlüssel zur Verfügung: einen Klartext-Datenschlüssel, eine Kopie des mit dem angegebenen Wrapping-Schlüssel verschlüsselten Datenschlüssels und einen MAC-Schlüssel.

   > **Note**
   >
   > Sie können den Datenschlüssel unter mehreren Wrapping-Schlüsseln verschlüsseln. Jeder der Umschließungsschlüssel verschlüsselt eine separate Kopie des Datenschlüssels. Das AWS Database Encryption SDK speichert alle verschlüsselten Datenschlüssel in der Materialbeschreibung. Das AWS Database Encryption SDK fügt dem Datensatz, der die Materialbeschreibung speichert, ein neues Feld (`aws_dbe_head`) hinzu.
   >
   > Für jede verschlüsselte Kopie des Datenschlüssels wird ein MAC-Schlüssel abgeleitet. Die MAC-Schlüssel sind nicht in der Materialbeschreibung gespeichert. Stattdessen verwendet die Entschlüsselungsmethode die Wrapping-Schlüssel, um die MAC-Schlüssel erneut abzuleiten.

2. Die Verschlüsselungsmethode verschlüsselt jedes Feld, das `ENCRYPT_AND_SIGN` in den von Ihnen angegebenen kryptografischen Aktionen als markiert ist.
3. Die Verschlüsselungsmethode leitet a `commitKey` aus dem Datenschlüssel ab und generiert daraus einen Wert für die Schlüsselzuweisung. Anschließend wird der Datenschlüssel verworfen.
4. Die Verschlüsselungsmethode fügt dem Datensatz eine Materialbeschreibung hinzu. Die Materialbeschreibung enthält die verschlüsselten Datenschlüssel und die anderen Informationen über den verschlüsselten Datensatz. Eine vollständige Liste der in der Materialbeschreibung enthaltenen Informationen finden Sie unter Format der Materialbeschreibung.
5. Die Verschlüsselungsmethode verwendet die in Schritt 1 zurückgegebenen MAC-Schlüssel, um die HMAC-Werte (Hash-Based Message Authentication Code) anhand der Kanonisierung der Materialbeschreibung, des Verschlüsselungskontextes und aller mit `ENCRYPT_AND_SIGNSIGN_ONLY`, oder markierten Felder `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` in den kryptografischen Aktionen zu berechnen. Die HMAC-Werte werden in einem neuen Feld (`aws_dbe_foot`) gespeichert, das die Verschlüsselungsmethode dem Datensatz hinzufügt.
6. Die Verschlüsselungsmethode berechnet anhand der Kanonisierung der Materialbeschreibung, des Verschlüsselungskontextes und jedes mit, oder markierten `ENCRYPT_AND_SIGN` Felds eine

ECDSA-Signatur `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` und speichert die ECDSA-Signaturen in dem Feld. `SIGN_ONLY` `aws_dbe_foot`

> **Note**
>
> ECDSA-Signaturen sind standardmäßig aktiviert, aber nicht erforderlich.

7. Die Verschlüsselungsmethode speichert den verschlüsselten und signierten Datensatz in Ihrer Datenbank

Entschlüsseln und verifizieren

1. Der Cryptographic Materials Manager (CMM) stellt die Entschlüsselungsmethode mit den in der Materialbeschreibung gespeicherten Entschlüsselungsmaterialien bereit, einschließlich des Klartext-Datenschlüssels und des zugehörigen MAC-Schlüssels.
   - Das CMM entschlüsselt den verschlüsselten Datenschlüssel, wobei die Schlüssel im angegebenen Schlüsselbund eingeschlossen sind, und gibt den Klartext-Datenschlüssel zurück.
2. Bei der Entschlüsselungsmethode wird der in der Materialbeschreibung angegebene Wert für die Schlüsselzusage verglichen und verifiziert.
3. Die Entschlüsselungsmethode überprüft die Signaturen im Signaturfeld.

   Sie identifiziert, welche Felder markiert sind `ENCRYPT_AND_SIGNSIGN_ONLY`, oder `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` anhand der Liste der erlaubten, nicht authentifizierten Felder, die Sie definiert haben. Die Entschlüsselungsmethode verwendet den in Schritt 1 zurückgegebenen MAC-Schlüssel, um die HMAC-Werte für die mit, oder markierten Felder neu zu berechnen und zu vergleichen. `ENCRYPT_AND_SIGN` `SIGN_ONLY` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Anschließend werden die ECDSA-Signaturen anhand des im Verschlüsselungskontext gespeicherten öffentlichen Schlüssels überprüft.
4. Die Entschlüsselungsmethode verwendet den Klartext-Datenschlüssel, um jeden markierten Wert zu entschlüsseln. `ENCRYPT_AND_SIGN` Das AWS Database Encryption SDK verwirft dann den Klartext-Datenschlüssel.
5. Die Entschlüsselungsmethode gibt den Klartext-Datensatz zurück.

Unterstützte Algorithmus-Suiten im AWS Database Encryption SDK

Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Ein Algorithmen-Paket ist eine Sammlung von kryptografischen Algorithmen und zugehörigen Werten. Kryptografische Systeme verwenden die Implementierung des Algorithmus, um den Chiffretext zu generieren.

Das AWS Database Encryption SDK verwendet eine Algorithmus-Suite, um die Felder in Ihrer Datenbank zu verschlüsseln und zu signieren. Alle unterstützten Algorithmus-Suiten verwenden den Advanced Encryption Standard (AES) -Algorithmus mit Galois/Counter Modus (GCM), auch bekannt als AES-GCM, zur Verschlüsselung von Rohdaten. Das AWS Database Encryption SDK unterstützt 256-Bit-Verschlüsselungsschlüssel. Die Länge des Authentifizierungs-Tags beträgt immer 16 Bytes.

AWS Algorithmus-Suiten für das Datenbankverschlüsselungs-SDK

Algorithmus	Verschlüsselungsalgorithmus	Länge des Datenschlüssels (in Bit)	Schlüsselableitungsalgorithmus	Symmetrischer Signaturalgorithmus	Asymmetrischer Signaturalgorithmus	Wichtiges Engagement
Standard	AES-GCM	256	HKDF mit SHA-512	HMAC-SHA-384	ECDSA mit P-384 und SHA-384	HKDF mit SHA-512
AES-GCM ohne digitale ECDSA-Signaturen	AES-GCM	256	HKDF mit SHA-512	HMAC-SHA-384	Keine	HKDF mit SHA-512

Verschlüsselungsalgorithmus

Der Name und der Modus des verwendeten Verschlüsselungsalgorithmus. Die Algorithmus-Suiten im AWS Database Encryption SDK verwenden den Advanced Encryption Standard (AES) - Algorithmus mit Galois/Counter Modus (GCM).

Länge des Datenschlüssels

Die Länge des Datenschlüssels in Bits. Das AWS Database Encryption SDK unterstützt 256-Bit-Datenschlüssel. Der Datenschlüssel wird als Eingabe für eine HMAC-basierte extract-and-expand Schlüsselableitungsfunktion (HKDF) verwendet. Die Ausgabe des HKDF wird als Datenverschlüsselungsschlüssel im Verschlüsselungsalgorithmus verwendet.

Schlüsselableitungsalgorithmus

Die HMAC-basierte extract-and-expand Schlüsselableitungsfunktion (HKDF), die zur Ableitung des Datenverschlüsselungsschlüssels verwendet wird. Das AWS Database Encryption SDK verwendet das in RFC 5869 definierte HKDF.

- Die verwendete Hash-Funktion ist SHA-512
- Für den Extraktionsschritt:
  - Es wird kein Salt verwendet. Gemäß dem RFC ist das Salz auf eine Zeichenfolge aus Nullen gesetzt.
  - Das Eingabematerial ist der Datenschlüssel aus dem Schlüsselbund.
- Für den Expansionsschritt:
  - Der pseudozufällige Eingabeschlüssel ist die Ausgabe aus dem Extraktionsschritt.
  - Die Schlüsselbezeichnung besteht aus den UTF-8-kodierten Bytes der `DERIVEKEY` Zeichenfolge in Big-Endian-Byte-Reihenfolge.
  - Die Eingabeinformationen sind eine Verkettung der Algorithmus-ID und der Schlüsselbezeichnung (in dieser Reihenfolge).
  - Die Länge des Ausgabe-Keying-Materials entspricht der Länge des Datenschlüssels. Diese Ausgabe wird als Datenverschlüsselungsschlüssel im Verschlüsselungsalgorithmus verwendet.

Symmetrischer Signaturalgorithmus

Der HMAC-Algorithmus (Hash-Based Message Authentication Code), der zur Generierung einer symmetrischen Signatur verwendet wird. Alle unterstützten Algorithmus-Suiten beinhalten die HMAC-Verifizierung.

Das AWS Database Encryption SDK serialisiert die Materialbeschreibung und alle mit`ENCRYPT_AND_SIGN`, `SIGN_ONLY` oder markierten Felder. `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Anschließend verwendet es HMAC mit einem kryptografischen Hashfunktionsalgorithmus (SHA-384), um die Kanonisierung zu signieren.

Die symmetrische HMAC-Signatur wird in einem neuen Feld (`aws_dbe_foot`) gespeichert, das das Database Encryption SDK dem AWS Datensatz hinzufügt.

Asymmetrischer Signaturalgorithmus

Der Signaturalgorithmus, der zur Generierung einer asymmetrischen digitalen Signatur verwendet wird.

Das AWS Database Encryption SDK serialisiert die Materialbeschreibung und alle mit`ENCRYPT_AND_SIGN`, `SIGN_ONLY` oder markierten Felder. `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Anschließend verwendet es den Elliptic Curve Digital Signature Algorithm (ECDSA) mit den folgenden Besonderheiten, um die Kanonisierung zu signieren:

- Bei der verwendeten elliptischen Kurve handelt es sich um die P-384, wie sie im Digital Signature Standard (DSS) (FIPS PUB 186-4) definiert ist.
- Die verwendete Hash-Funktion ist SHA-384.

Die asymmetrische ECDSA-Signatur wird zusammen mit der symmetrischen HMAC-Signatur im Feld gespeichert. `aws_dbe_foot`

Digitale ECDSA-Signaturen sind standardmäßig enthalten, aber nicht erforderlich.

Wichtiges Engagement

Die HMAC-basierte extract-and-expand Schlüsselableitungsfunktion (HKDF), die zur Ableitung des Commit-Schlüssels verwendet wird.

- Die verwendete Hash-Funktion ist SHA-512
- Für den Extraktionsschritt:
  - Es wird kein Salt verwendet. Gemäß dem RFC ist das Salz auf eine Zeichenfolge aus Nullen gesetzt.
  - Das Eingabematerial ist der Datenschlüssel aus dem Schlüsselbund.
- Für den Expansionsschritt:
  - Der pseudozufällige Eingabeschlüssel ist die Ausgabe aus dem Extraktionsschritt.

- Die Eingabeinformationen sind die UTF-8-kodierten Bytes der `COMMITKEY` Zeichenfolge in Big-Endian-Byte-Reihenfolge.
- Die Länge des Ausgangs-Keying-Materials beträgt 256 Bit. Diese Ausgabe wird als Commit-Schlüssel verwendet.

Mit dem Commit-Schlüssel wird das Record Commitment, ein eindeutiger 256-Bit-HMAC-Hash (Hash-Based Message Authentication Code), anhand der Materialbeschreibung berechnet. Eine technische Erläuterung des Hinzufügens von Key Commitment zu einer Algorithmus-Suite finden Sie unter Key AEADs Committing in Cryptology ePrint Archive.

Standard-Algorithmus-Suite

Standardmäßig verwendet das AWS Database Encryption SDK eine Algorithmus-Suite mit AES-GCM, einer HMAC-basierten extract-and-expand Schlüsselableitungsfunktion (HKDF), HMAC-Verifizierung, digitalen ECDSA-Signaturen, Key Commitment und einem 256-Bit-Verschlüsselungsschlüssel.

Die Standard-Algorithmus-Suite umfasst HMAC-Verifizierung (symmetrische Signaturen) und digitale ECDSA-Signaturen (asymmetrische Signaturen). Diese Signaturen werden in einem neuen Feld (`aws_dbe_foot`) gespeichert, das das AWS Database Encryption SDK dem Datensatz hinzufügt. Digitale ECDSA-Signaturen sind besonders nützlich, wenn die Autorisierungsrichtlinie es einer Benutzergruppe erlaubt, Daten zu verschlüsseln und einer anderen Benutzergruppe, Daten zu entschlüsseln.

Die standardmäßige Algorithmussuite leitet außerdem eine Schlüsselzusage ab — einen HMAC-Hash, der den Datenschlüssel mit dem Datensatz verknüpft. Der Key Commitment-Wert ist ein HMAC, der anhand der Materialbeschreibung und des Commit-Schlüssels berechnet wird. Der Key Commitment Value wird dann in der Materialbeschreibung gespeichert. Key Commitment stellt sicher, dass jeder Chiffretext nur in einen Klartext entschlüsselt wird. Dazu validieren sie den Datenschlüssel, der als Eingabe für den Verschlüsselungsalgorithmus verwendet wird. Bei der Verschlüsselung leitet die Algorithmus-Suite eine Schlüsselzusage (HMAC) ab. Vor der Entschlüsselung überprüfen sie, ob der Datenschlüssel dieselbe Schlüsselzusage (HMAC) erzeugt. Ist dies nicht der Fall, schlägt der Entschlüsselungsaufruf fehl.

AES-GCM ohne digitale ECDSA-Signaturen

Obwohl die Standard-Algorithmus-Suite wahrscheinlich für die meisten Anwendungen geeignet ist, können Sie auch eine alternative Algorithmussuite wählen. Einige Vertrauensmodelle würden

beispielsweise durch eine Algorithmussuite ohne digitale ECDSA-Signaturen erfüllt. Verwenden Sie diese Suite nur, wenn die Benutzer, die Daten verschlüsseln, und die Benutzer, die Daten entschlüsseln, gleichermaßen vertrauenswürdig sind.

Alle Algorithmus-Suiten des AWS Database Encryption SDK beinhalten HMAC-Verifizierung (symmetrische Signaturen). Der einzige Unterschied besteht darin, dass der AES-GCM-Algorithmussuite ohne digitale ECDSA-Signatur die asymmetrische Signatur fehlt, die eine zusätzliche Ebene für Authentizität und Unwiderlegbarkeit bietet.

Wenn Ihr Schlüsselbund,, und beispielsweise mehrere Schlüssel zum Umschließen enthält und Sie einen Datensatz mithilfe der symmetrischen `wrappingKeyB` HMAC-Signatur entschlüsseln`wrappingKeyC`, bestätigt die symmetrische HMAC-Signatur`wrappingKeyA`, dass der Datensatz von einem Benutzer mit Zugriff auf verschlüsselt wurde. `wrappingKeyA` `wrappingKeyA` Wenn Sie die standardmäßige Algorithmussuite verwendet haben, HMACs stellen sie dieselbe Überprüfung von bereit und verwenden zusätzlich die digitale ECDSA-Signatur`wrappingKeyA`, um sicherzustellen, dass der Datensatz von einem Benutzer mit Verschlüsselungsberechtigungen für verschlüsselt wurde. `wrappingKeyA`

Um die AES-GCM-Algorithmussuite ohne digitale Signaturen auszuwählen, nehmen Sie den folgenden Ausschnitt in Ihre Verschlüsselungskonfiguration auf.

Java

Der folgende Ausschnitt spezifiziert die AES-GCM-Algorithmussuite ohne digitale ECDSA-Signaturen. Weitere Informationen finden Sie unter the section called “Verschlüsselungskonfiguration”.

```
.algorithmSuiteId(
    DBEAlgorithmSuiteId.ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY_SYMSIG_HMAC_SHA384)
```

C# / .NET

Der folgende Ausschnitt spezifiziert die AES-GCM-Algorithmussuite ohne digitale ECDSA-Signaturen. Weitere Informationen finden Sie unter the section called “Verschlüsselungskonfiguration”.

```
AlgorithmSuiteId =
 DBEAlgorithmSuiteId.ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY_SYMSIG_HMAC_SHA384
```

Rust

Der folgende Ausschnitt spezifiziert die AES-GCM-Algorithmussuite ohne digitale ECDSA-Signaturen. Weitere Informationen finden Sie unter the section called “Verschlüsselungskonfiguration”.

```
.algorithm_suite_id(
    DbeAlgorithmSuiteId::AlgAes256GcmHkdfSha512CommitKeyEcdsaP384SymsigHmacSha384,
)
```

Verwendung der AWS Datenbankverschlüsselungs-SDK mit AWS KMS

Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Um das AWS Database Encryption SDK verwenden zu können, müssen Sie einen Schlüsselbund konfigurieren und einen oder mehrere Wrapping-Schlüssel angeben. Wenn Sie keine Schlüsselinfrastruktur haben, empfehlen wir die Verwendung von AWS Key Management Service (AWS KMS).

Das AWS Database Encryption SDK unterstützt zwei Arten von AWS KMS Schlüsselbunden. Der herkömmliche AWS KMS Schlüsselbund wird AWS KMS keyszum Generieren, Verschlüsseln und Entschlüsseln von Datenschlüsseln verwendet. Sie können entweder symmetrische Verschlüsselung (`SYMMETRIC_DEFAULT`) oder asymmetrische RSA-KMS-Schlüssel verwenden. Da das AWS Database Encryption SDK jeden Datensatz mit einem eindeutigen Datenschlüssel verschlüsselt und signiert, muss der AWS KMS Schlüsselbund bei jedem Verschlüsselungs- und AWS KMS Entschlüsselungsvorgang aufgerufen werden. Für Anwendungen, die die Anzahl der Aufrufe minimieren müssen AWS KMS, unterstützt das AWS Database Encryption SDK auch den hierarchischen Schlüsselbund.AWS KMS Der hierarchische Schlüsselbund ist eine Lösung zum Zwischenspeichern von kryptografischem Material, die die Anzahl der AWS KMS Aufrufe reduziert, indem AWS KMS geschützte Branch-Schlüssel verwendet werden, die in einer Amazon DynamoDB-Tabelle gespeichert sind, und anschließend das lokale Zwischenspeichern von Zweigschlüsselmaterialien, die bei Verschlüsselungs- und Entschlüsselungsvorgängen verwendet werden, erfolgt. Wir empfehlen, wann immer möglich, die Schlüsselringe zu verwenden. AWS KMS

Für die Interaktion mit dem AWS KMS AWS Database Encryption SDK ist das AWS KMS Modul von erforderlich. AWS SDK für Java

Um sich auf die Verwendung des vorzubereiten AWS SDK für Datenbankverschlüsselung mit AWS KMS

- Erstellen Sie eine symmetrische Verschlüsselung AWS KMS key. Hilfe finden Sie unter Creating Keys im AWS Key Management Service Developer Guide.

> **Tip**
>
> Um das AWS KMS key programmgesteuert verwenden zu können, benötigen Sie den Amazon-Ressourcennamen (ARN) von. AWS KMS key Hilfe bei der Suche nach dem ARN eines AWS KMS keyfinden Sie unter Suchen der Schlüssel-ID und des ARN im AWS Key Management Service Entwicklerhandbuch.

Konfigurieren von AWS SDK für Datenbankverschlüsselung

Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

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

Themen

Auswahl einer Programmiersprache

Das AWS Database Encryption SDK für DynamoDB ist in mehreren Programmiersprachen verfügbar. Die Sprachimplementierungen sind so konzipiert, dass sie vollständig interoperabel sind und dieselben Funktionen bieten, obwohl sie möglicherweise auf unterschiedliche Weise implementiert werden. In der Regel verwenden Sie die Bibliothek, die mit Ihrer Anwendung kompatibel ist.

Auswahl von Wraping-Schlüsseln

Das AWS Database Encryption SDK generiert einen eindeutigen symmetrischen Datenschlüssel, um jedes Feld zu verschlüsseln. Sie müssen die Datenschlüssel nicht konfigurieren, verwalten oder verwenden. Das AWS Database Encryption SDK erledigt das für Sie.

Sie müssen jedoch einen oder mehrere Wrapping-Schlüssel auswählen, um jeden Datenschlüssel zu verschlüsseln. Das AWS Database Encryption SDK unterstützt AWS Key Management Service(AWS

KMS) symmetrische Verschlüsselungs-KMS-Schlüssel und asymmetrische RSA-KMS-Schlüssel. Es unterstützt auch symmetrische AES-Schlüssel und asymmetrische RSA-Schlüssel, die Sie in verschiedenen Größen bereitstellen. Sie sind für die Sicherheit und Haltbarkeit Ihrer Wrapping-Schlüssel verantwortlich. Wir empfehlen Ihnen daher, einen Verschlüsselungsschlüssel in einem Hardware-Sicherheitsmodul oder einem wichtigen Infrastrukturdienst zu verwenden, wie z. AWS KMS

Um Ihre Verpackungsschlüssel für die Verschlüsselung und Entschlüsselung anzugeben, verwenden Sie einen Schlüsselbund. Je nach Art des verwendeten Schlüsselbundes können Sie einen Wrapping-Schlüssel oder mehrere Wrap-Schlüssel desselben oder verschiedener Typen angeben. Wenn Sie für den Umbruch eines Datenschlüssels mehrere Schlüssel verwenden, verschlüsselt jeder Umbruchschlüssel eine Kopie desselben Datenschlüssels. Die verschlüsselten Datenschlüssel (einer pro Umschließungsschlüssel) werden in der Materialbeschreibung gespeichert, die neben dem verschlüsselten Feld gespeichert wird. Um die Daten zu entschlüsseln, muss das AWS Database Encryption SDK zunächst einen Ihrer Verpackungsschlüssel verwenden, um einen verschlüsselten Datenschlüssel zu entschlüsseln.

Wir empfehlen, wann immer möglich, einen der AWS KMS Schlüsselringe zu verwenden. Das AWS Database Encryption SDK stellt den AWS KMS Schlüsselbund und den AWS KMS hierarchischen Schlüsselbund bereit, wodurch die Anzahl der Aufrufe reduziert wird. AWS KMS Um einen AWS KMS key in einem Schlüsselbund anzugeben, verwenden Sie eine unterstützte Schlüssel-ID. AWS KMS Wenn Sie den AWS KMS hierarchischen Schlüsselbund verwenden, müssen Sie den Schlüssel-ARN angeben. Einzelheiten zu den Schlüsselbezeichnern für einen AWS KMS Schlüssel finden Sie unter Schlüsselkennungen im Entwicklerhandbuch.AWS Key Management Service

- Wenn Sie mit einem AWS KMS Schlüsselbund verschlüsseln, können Sie eine beliebige gültige Schlüssel-ID (Schlüssel-ARN, Aliasname, Alias-ARN oder Schlüssel-ID) für einen KMS-Schlüssel mit symmetrischer Verschlüsselung angeben. Wenn Sie einen asymmetrischen RSA-KMS-Schlüssel verwenden, müssen Sie den Schlüssel-ARN angeben.

  Wenn Sie bei der Verschlüsselung einen Aliasnamen oder Alias-ARN für einen KMS-Schlüssel angeben, speichert das AWS Database Encryption SDK den Schlüssel-ARN, der derzeit mit diesem Alias verknüpft ist; es speichert den Alias nicht. Änderungen am Alias wirken sich nicht auf den KMS-Schlüssel aus, der zum Entschlüsseln Ihrer Datenschlüssel verwendet wird.
- Standardmäßig entschlüsselt der AWS KMS Schlüsselbund Datensätze im strikten Modus (in dem Sie bestimmte KMS-Schlüssel angeben). Sie müssen einen Schlüssel-ARN verwenden, um sich AWS KMS keys für die Entschlüsselung zu identifizieren.

Wenn Sie mit einem AWS KMS Schlüsselbund verschlüsseln, speichert das AWS Database Encryption SDK den Schlüssel ARN von AWS KMS key in der Materialbeschreibung zusammen mit dem verschlüsselten Datenschlüssel. Bei der Entschlüsselung im strikten Modus überprüft das AWS Database Encryption SDK, ob derselbe Schlüssel-ARN im Schlüsselbund erscheint, bevor es versucht, den Wrapping-Schlüssel zum Entschlüsseln des verschlüsselten Datenschlüssels zu verwenden. Wenn Sie eine andere Schlüssel-ID verwenden, erkennt oder verwendet das AWS Database Encryption SDK diese nicht AWS KMS key, auch wenn sich die Kennungen auf denselben Schlüssel beziehen.

- Beim Entschlüsseln im Discovery-Modus geben Sie keine Wrapping-Schlüssel an. Zunächst versucht das AWS Database Encryption SDK, den Datensatz mit dem Schlüssel ARN zu entschlüsseln, der in der Materialbeschreibung gespeichert ist. Wenn das nicht funktioniert, fordert das AWS Database Encryption SDK auf, den Datensatz mit dem KMS-Schlüssel AWS KMS zu entschlüsseln, mit dem er verschlüsselt wurde, unabhängig davon, wem dieser KMS-Schlüssel gehört oder wer Zugriff darauf hat.

Um einen AES-Rohschlüssel oder ein RSA-Rohschlüsselpaar als Umschließungsschlüssel in einem Schlüsselbund anzugeben, müssen Sie einen Namespace und einen Namen angeben. Beim Entschlüsseln müssen Sie für jeden Rohverpackungsschlüssel genau denselben Namespace und denselben Namen verwenden wie beim Verschlüsseln. Wenn Sie einen anderen Namespace oder Namen verwenden, erkennt oder verwendet das AWS Database Encryption SDK den Wrapping-Schlüssel nicht, auch wenn das Schlüsselmaterial identisch ist.

Einen Discovery-Filter erstellen

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

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

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

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

Das folgende Beispiel zeigt, wie Sie einen Discovery-Filter erstellen. Bevor Sie den Code verwenden, ersetzen Sie die Beispielwerte durch gültige Werte für Ihre Partition AWS-Konto und.

Java

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

C# / .NET

```
var discoveryFilter = new DiscoveryFilter
{
    Partition = "aws",
    AccountIds = 111122223333
};
```

Rust

```
// Create discovery filter
let discovery_filter = DiscoveryFilter::builder()
    .partition("aws")
    .account_ids(111122223333)
    .build()?;
```

Arbeiten mit Mehrmandantendatenbanken

Mit dem AWS Database Encryption SDK können Sie die clientseitige Verschlüsselung für Datenbanken mit einem gemeinsamen Schema konfigurieren, indem Sie jeden Mandanten mit unterschiedlichen Verschlüsselungsmaterialien isolieren. Wenn Sie eine mehrinstanzenfähige Datenbank in Betracht ziehen, sollten Sie sich etwas Zeit nehmen, um Ihre Sicherheitsanforderungen zu überprüfen und zu überprüfen, wie sich die Mehrmandantenfähigkeit darauf auswirken könnte. Beispielsweise kann die Verwendung einer Mehrmandantendatenbank Ihre Fähigkeit beeinträchtigen, das AWS Database Encryption SDK mit einer anderen serverseitigen Verschlüsselungslösung zu kombinieren.

Wenn mehrere Benutzer Verschlüsselungsvorgänge in Ihrer Datenbank durchführen, können Sie einen der AWS KMS Schlüsselbunde verwenden, um jedem Benutzer einen eigenen Schlüssel zur Verfügung zu stellen, den er für seine kryptografischen Operationen verwenden kann. Die Verwaltung der Datenschlüssel für eine clientseitige Verschlüsselungslösung mit mehreren Mandanten kann kompliziert sein. Wir empfehlen, Ihre Daten wann immer möglich nach Mandanten zu organisieren. Wenn der Mandant anhand der Primärschlüsselwerte identifiziert wird (z. B. der Partitionsschlüssel in einer Amazon DynamoDB-Tabelle), ist die Verwaltung Ihrer Schlüssel einfacher.

Sie können den AWS KMS Schlüsselbund verwenden, um jeden Mandanten mit einem eigenen AWS KMS Schlüsselbund und zu isolieren. AWS KMS keys Je nach Anzahl der pro Mandant AWS KMS getätigten Anrufe möchten Sie möglicherweise den AWS KMS hierarchischen Schlüsselbund verwenden, um die Anzahl der Anrufe zu minimieren. AWS KMS Der AWS KMS hierarchische Schlüsselbund ist eine Lösung zum Zwischenspeichern von kryptografischem Material, die die Anzahl der AWS KMS Aufrufe reduziert, indem AWS KMS geschützte Branch-Schlüssel verwendet werden, die in einer Amazon DynamoDB-Tabelle gespeichert sind, und anschließend das lokale Zwischenspeichern von Zweigschlüsselmaterialien, die bei Verschlüsselungs- und Entschlüsselungsvorgängen verwendet werden, erfolgt. Sie müssen den hierarchischen Schlüsselbund verwenden, um eine durchsuchbare Verschlüsselung in Ihrer Datenbank zu AWS KMS implementieren.

Signierte Beacons erstellen

Das AWS Database Encryption SDK verwendet Standardbeacons und zusammengesetzte Beacons, um durchsuchbare Verschlüsselungslösungen bereitzustellen, mit denen Sie verschlüsselte Datensätze durchsuchen können, ohne die gesamte abgefragte Datenbank entschlüsseln zu müssen. Das AWS Database Encryption SDK unterstützt jedoch auch signierte Beacons, die

vollständig aus Klartext-signierten Feldern konfiguriert werden können. Signierte Beacons sind eine Art von zusammengesetzten Beacons, die komplexe Abfragen von Feldern indizieren und ausführen. `SIGN_ONLY` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`

Wenn Sie beispielsweise über eine Mehrmandantendatenbank verfügen, möchten Sie möglicherweise ein signiertes Beacon erstellen, mit dem Sie Ihre Datenbank nach Datensätzen abfragen können, die mit einem bestimmten Mandantenschlüssel verschlüsselt wurden. Weitere Informationen finden Sie unter Abfragen von Beacons in einer mandantenfähigen Datenbank.

Sie müssen den AWS KMS hierarchischen Schlüsselbund verwenden, um signierte Beacons zu erstellen.

Um ein signiertes Beacon zu konfigurieren, geben Sie die folgenden Werte an.

Java

Konfiguration eines zusammengesetzten Beacons

Im folgenden Beispiel werden die signierten Teilelisten lokal innerhalb der signierten Beacon-Konfiguration definiert.

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

Definition der Beacon-Version

Das folgende Beispiel definiert die signierten Teilelisten global in der Beacon-Version. Weitere Informationen zur Definition der Beacon-Version finden Sie unter Beacons verwenden.

```
 List<BeaconVersion> beaconVersions = new ArrayList<>();
beaconVersions.add(
    BeaconVersion.builder()
        .standardBeacons(standardBeaconList)
        .compoundBeacons(compoundBeaconList)
        .signedParts(signedPartList)
```

```
        .version(1) // MUST be 1
        .keyStore(keyStore)
        .keySource(BeaconKeySource.builder()
            .single(SingleKeyStore.builder()
                .keyId(branchKeyId)
                .cacheTTL(6000)
                .build())
            .build())
        .build()
);
```

C# / .NET

Sehen Sie sich das vollständige Codebeispiel an: BeaconConfig.cs

Signierte Beacon-Konfiguration

Das folgende Beispiel definiert die signierten Teilelisten lokal innerhalb der signierten Beacon-Konfiguration.

```
var compoundBeaconList = new List<CompoundBeacon>();
var exampleCompoundBeacon = new CompoundBeacon
 {
    Name = "compoundBeaconName",
    Split = ".",
    Signed = signedPartList,
    Constructors = constructorList
 };
compoundBeaconList.Add(exampleCompoundBeacon);
```

Definition der Beacon-Version

Das folgende Beispiel definiert die signierten Teilelisten global in der Beacon-Version. Weitere Informationen zur Definition der Beacon-Version finden Sie unter Beacons verwenden.

```
var beaconVersions = new List<BeaconVersion>
{
    new BeaconVersion
    {
        StandardBeacons = standardBeaconList,
        CompoundBeacons = compoundBeaconList,
        SignedParts = signedPartsList,
        Version = 1, // MUST be 1
```

```
        KeyStore = keyStore,
        KeySource = new BeaconKeySource
        {
            Single = new SingleKeyStore
            {
                KeyId = branchKeyId,
                CacheTTL = 6000
            }
        }
    }
};
```

Sie können Ihre signierten Teile in lokal oder global definierten Listen definieren. Wir empfehlen, Ihre signierten Teile wann immer möglich in einer globalen Liste in der Beacon-Version zu definieren. Durch die globale Definition signierter Teile können Sie jedes Teil einmal definieren und die Teile dann in mehreren Compound-Beacon-Konfigurationen wiederverwenden. Wenn Sie beabsichtigen, ein signiertes Teil nur einmal zu verwenden, können Sie es in einer lokalen Liste in der signierten Beacon-Konfiguration definieren. Sie können in Ihrer Konstruktorliste sowohl auf lokale als auch auf globale Teile verweisen.

Wenn Sie Ihre signierten Teilelisten global definieren, müssen Sie eine Liste von Konstruktorteilen bereitstellen, die alle Möglichkeiten aufzeigt, wie der signierte Beacon die Felder in Ihrer Beacon-Konfiguration zusammenstellen kann.

> **Note**
>
> Um signierte Teilelisten global zu definieren, müssen Sie Version 3.2 oder höher des AWS Database Encryption SDK verwenden. Stellen Sie die neue Version allen Lesern zur Verfügung, bevor Sie neue Teile global definieren.
> Sie können bestehende Beacon-Konfigurationen nicht aktualisieren, um signierte Teilelisten global zu definieren.

Name des Beacons

Der Name, den Sie bei der Abfrage des Beacons verwenden.

Ein signierter Beacon-Name darf nicht derselbe Name wie ein unverschlüsseltes Feld sein. Keine zwei Beacons können denselben Beacon-Namen haben.

Charakter teilen

Das Zeichen, das verwendet wird, um die Teile zu trennen, aus denen Ihr signiertes Beacon besteht.

Das Trennzeichen darf in den Klartextwerten der Felder, aus denen der signierte Beacon aufgebaut ist, nicht vorkommen.

Liste der signierten Teile

Identifiziert die signierten Felder, die im signierten Beacon enthalten sind.

Jeder Teil muss einen Namen, eine Quelle und ein Präfix enthalten. Die Quelle ist das `SIGN_ONLY` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` ODER-Feld, das der Teil identifiziert. Die Quelle muss ein Feldname oder ein Index sein, der auf den Wert eines verschachtelten Felds verweist. Wenn Ihr Teilname die Quelle identifiziert, können Sie die Quelle weglassen und das AWS Database Encryption SDK verwendet den Namen automatisch als Quelle. Wir empfehlen, wann immer möglich, die Quelle als Teilnamen anzugeben. Das Präfix kann eine beliebige Zeichenfolge sein, muss jedoch eindeutig sein. Keine zwei signierten Teile in einem signierten Beacon dürfen dasselbe Präfix haben. Wir empfehlen, einen kurzen Wert zu verwenden, der den Teil von anderen Teilen unterscheidet, die vom Compound Beacon bedient werden.

Wir empfehlen, Ihre signierten Teile nach Möglichkeit global zu definieren. Sie könnten erwägen, ein signiertes Teil lokal zu definieren, wenn Sie es nur in einem Compound Beacon verwenden möchten. Ein lokal definierter Teil kann nicht dasselbe Präfix oder denselben Namen haben wie ein global definierter Teil.

Java

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

C# / .NET

```
var signedPartsList = new List<SignedPart>
{
```

```
    new SignedPart { Name = "signedFieldName1", Prefix = "S-" },
    new SignedPart { Name = "signedFieldName2", Prefix = "SF-" }
};
```

Konstruktorliste (optional)

Identifiziert die Konstruktoren, die die verschiedenen Arten definieren, wie die signierten Teile durch den signierten Beacon zusammengebaut werden können.

Wenn Sie keine Konstruktorliste angeben, baut das AWS Database Encryption SDK den signierten Beacon mit dem folgenden Standardkonstruktor zusammen.

- Alle signierten Teile in der Reihenfolge, in der sie der signierten Teileliste hinzugefügt wurden
- Alle Teile sind erforderlich

Konstruktoren

Jeder Konstruktor ist eine geordnete Liste von Konstruktorteilen, die eine Art und Weise definiert, wie das signierte Beacon zusammengebaut werden kann. Die Konstruktorteile werden in der Reihenfolge zusammengefügt, in der sie der Liste hinzugefügt wurden, wobei jeder Teil durch das angegebene Trennzeichen getrennt wird.

Jeder Konstruktorteil benennt einen Teil mit Vorzeichen und definiert, ob dieser Teil innerhalb des Konstruktors erforderlich oder optional ist. Wenn Sie beispielsweise ein signiertes Beacon auf`Field1`, und abfragen möchten `Field1.Field2``Field1.Field2.Field3`, markieren Sie und `Field3` als optional `Field2` und erstellen Sie einen Konstruktor.

Jeder Konstruktor muss mindestens einen erforderlichen Teil haben. Wir empfehlen, den ersten Teil in jedem Konstruktor als erforderlich festzulegen, damit Sie den `BEGINS_WITH` Operator in Ihren Abfragen verwenden können.

Ein Konstruktor ist erfolgreich, wenn alle erforderlichen Teile im Datensatz vorhanden sind. Wenn Sie einen neuen Datensatz schreiben, bestimmt der signierte Beacon anhand der Konstruktorliste, ob der Beacon aus den bereitgestellten Werten zusammengesetzt werden kann. Es versucht, den Beacon in der Reihenfolge zusammenzustellen, in der die Konstruktoren der Konstruktorliste hinzugefügt wurden, und verwendet den ersten Konstruktor, der erfolgreich ist. Wenn keine Konstruktoren erfolgreich sind, wird der Beacon nicht in den Datensatz geschrieben.

Alle Leser und Autoren sollten die gleiche Reihenfolge der Konstruktoren angeben, um sicherzustellen, dass ihre Abfrageergebnisse korrekt sind.

Verwenden Sie die folgenden Verfahren, um Ihre eigene Konstruktorliste anzugeben.

1. Erstellen Sie für jedes signierte Teil ein Konstruktorteil, um zu definieren, ob dieses Teil erforderlich ist oder nicht.

   Der Name des Konstruktorteils muss der Name des signierten Felds sein.

   Das folgende Beispiel zeigt, wie ein Konstruktorteil für ein signiertes Feld erstellt wird.

   Java

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

   C# / .NET

   ```
   var field1ConstructorPart = new ConstructorPart { Name = "Field1", Required
    = true };
   ```

2. Erstellen Sie einen Konstruktor für jede mögliche Art und Weise, wie das signierte Beacon zusammengebaut werden kann. Verwenden Sie dazu die Konstruktorteile, die Sie in Schritt 1 erstellt haben.

   Wenn Sie beispielsweise nach `Field1.Field2.Field3` und abfragen möchten`Field4.Field2.Field3`, müssen Sie zwei Konstruktoren erstellen. `Field1`und `Field4` können beide erforderlich sein, da sie in zwei separaten Konstruktoren definiert sind.

   Java

   ```
   // 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();
```

C# / .NET

```
// Create a list for Field1.Field2.Field3 queries
 var field123ConstructorPartList = new Constructor
{
    Parts = new List<ConstructorPart> { field1ConstructorPart,
 field2ConstructorPart, field3ConstructorPart }
};
// Create a list for Field4.Field2.Field1 queries
var field421ConstructorPartList = new Constructor
{
    Parts = new List<ConstructorPart> { field4ConstructorPart,
 field2ConstructorPart, field1ConstructorPart }
};
```

3. Erstellen Sie eine Konstruktorliste, die alle Konstruktoren enthält, die Sie in Schritt 2 erstellt haben.

   Java

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

   C# / .NET

   ```
   var constructorList = new List<Constructor>
   {
       field123Constructor,
       field421Constructor
   };
   ```

4. Geben Sie an`constructorList`, wann Sie Ihr signiertes Beacon erstellen.

Schlüsselspeicher im AWS Database Encryption SDK

Im AWS Database Encryption SDK ist ein Schlüsselspeicher eine Amazon DynamoDB-Tabelle, die hierarchische Daten speichert, die vom hierarchischen Schlüsselbund verwendet werden.AWS KMS Der Schlüsselspeicher trägt dazu bei, die Anzahl der Aufrufe zu reduzieren, die Sie tätigen müssen, um kryptografische Operationen mit AWS KMS dem hierarchischen Schlüsselbund durchzuführen.

Der Schlüsselspeicher bleibt erhalten und verwaltet die Zweigschlüssel, die der hierarchische Schlüsselbund für die Umschlagverschlüsselung und den Schutz von Datenverschlüsselungsschlüsseln verwendet. Der Schlüsselspeicher speichert den aktiven Branch-Schlüssel und alle vorherigen Versionen des Branch-Schlüssels. Der aktive Zweigschlüssel ist die neueste Version des Zweigschlüssels. Der hierarchische Schlüsselbund verwendet für jede Verschlüsselungsanforderung einen eindeutigen Datenverschlüsselungsschlüssel und verschlüsselt jeden Datenverschlüsselungsschlüssel mit einem eindeutigen Umschließungsschlüssel, der vom aktiven Filialschlüssel abgeleitet wird. Der hierarchische Schlüsselbund hängt von der Hierarchie ab, die zwischen aktiven Zweigschlüsseln und ihren abgeleiteten Umschließungsschlüsseln festgelegt wurde.

Terminologie und Konzepte von Key Stores

Key Store (Schlüsselspeicher)

: Die DynamoDB-Tabelle, die hierarchische Daten wie Verzweigungsschlüssel und Beacon-Schlüssel persistiert.

Root-Schlüssel

: Ein KMS-Schlüssel mit symmetrischer Verschlüsselung, der die Branch- und Beacon-Schlüssel in Ihrem Schlüsselspeicher generiert und schützt.

Filialschlüssel

: Ein Datenschlüssel, der wiederverwendet wird, um einen eindeutigen Verpackungsschlüssel für die Umschlagverschlüsselung abzuleiten. Sie können mehrere Zweigschlüssel in einem Schlüsselspeicher erstellen, aber für jeden Zweigschlüssel kann jeweils nur eine aktive Version des Zweigschlüssels vorhanden sein. Der aktive Zweigschlüssel ist die neueste Version des Zweigschlüssels.

  Verzweigungsschlüssel werden AWS KMS keys mithilfe der GenerateDataKeyWithoutPlaintext Operation kms: abgeleitet.

Schlüssel umschließen

Ein eindeutiger Datenschlüssel, der zur Verschlüsselung des bei Verschlüsselungsvorgängen verwendeten Datenverschlüsselungsschlüssels verwendet wird.

Wrapping-Schlüssel werden von Zweigschlüsseln abgeleitet. Weitere Informationen zur Schlüsselableitung finden Sie unter Technische Details zum AWS KMS hierarchischen Schlüsselbund.

Schlüssel zur Datenverschlüsselung

Ein Datenschlüssel, der bei Verschlüsselungsvorgängen verwendet wird. Der hierarchische Schlüsselbund verwendet für jede Verschlüsselungsanforderung einen eindeutigen Datenverschlüsselungsschlüssel.

Beacon-Schlüssel

Ein Datenschlüssel, der zur Generierung von Beacons für eine durchsuchbare Verschlüsselung verwendet wird. Weitere Informationen finden Sie unter Durchsuchbare Verschlüsselung.

Implementieren der geringsten Berechtigungen

Bei der Verwendung eines Schlüsselspeichers und AWS KMS hierarchischer Schlüsselbunde empfehlen wir, dass Sie dem Prinzip der geringsten Rechte folgen, indem Sie die folgenden Rollen definieren:

Schlüsselspeicher-Administrator

Schlüsselspeicheradministratoren sind für die Erstellung und Verwaltung des Schlüsselspeichers und der Filialschlüssel verantwortlich, die dieser speichert und schützt. Key-Store-Administratoren sollten die einzigen Benutzer mit Schreibberechtigungen für die Amazon DynamoDB-Tabelle sein, die als Ihr Schlüsselspeicher dient. Sie sollten die einzigen Benutzer sein, die Zugriff auf privilegierte Administratoroperationen wie `CreateKey`und haben. `VersionKey` Sie können diese Operationen nur ausführen, wenn Sie Ihre Schlüsselspeicher-Aktionen statisch konfigurieren.

`CreateKey`ist eine privilegierte Operation, die Ihrer Schlüsselspeicher-Zulassungsliste einen neuen KMS-Schlüssel-ARN hinzufügen kann. Mit diesem KMS-Schlüssel können neue aktive Zweigschlüssel erstellt werden. Wir empfehlen, den Zugriff auf diesen Vorgang einzuschränken, da ein KMS-Schlüssel, der einmal dem Zweigschlüsselspeicher hinzugefügt wurde, nicht gelöscht werden kann.

Schlüsselspeicher-Benutzer

In den meisten Anwendungsfällen interagiert der Schlüsselspeicher-Benutzer beim Verschlüsseln, Entschlüsseln, Signieren und Überprüfen von Daten nur über den hierarchischen Schlüsselbund mit dem Schlüsselspeicher. Daher benötigen sie nur Leseberechtigungen für die Amazon DynamoDB-Tabelle, die als Ihr Schlüsselspeicher dient. Key-Store-Benutzer sollten nur Zugriff auf die Verwendungsvorgänge benötigen, die kryptografische Operationen ermöglichen, wie`GetActiveBranchKey`, und`GetBranchKeyVersion`. `GetBeaconKey` Sie benötigen keine Berechtigungen, um die von ihnen verwendeten Branch-Schlüssel zu erstellen oder zu verwalten.

Sie können Verwendungsvorgänge ausführen, wenn Ihre Schlüsselspeicher-Aktionen statisch konfiguriert sind oder wenn sie für die Erkennung konfiguriert sind. Sie können keine Administratoroperationen (`CreateKey`und`VersionKey`) ausführen, wenn Ihre Schlüsselspeicher-Aktionen für die Erkennung konfiguriert sind.

Wenn Ihr Filialschlüsselspeicheradministrator mehrere KMS-Schlüssel in Ihrem Zweigschlüsselspeicher zugelassen hat, empfehlen wir Ihren Schlüsselspeicher-Benutzern, ihre Schlüsselspeicher-Aktionen für die Erkennung so zu konfigurieren, dass ihr hierarchischer Schlüsselbund mehrere KMS-Schlüssel verwenden kann.

Einen Schlüsselspeicher erstellen

Bevor Sie Branch-Schlüssel erstellen oder einen AWS KMS hierarchischen Schlüsselbund verwenden können, müssen Sie Ihren Schlüsselspeicher erstellen, eine Amazon DynamoDB-Tabelle, die Ihre Branch-Schlüssel verwaltet und schützt.

> **Important**
>
> Löschen Sie nicht die DynamoDB-Tabelle, in der Ihre Branch-Schlüssel gespeichert sind. Wenn Sie diese Tabelle löschen, können Sie keine Daten entschlüsseln, die mit dem hierarchischen Schlüsselbund verschlüsselt wurden.

Folgen Sie den Verfahren zum Erstellen einer Tabelle im Amazon DynamoDB Developer Guide und verwenden Sie dabei die folgenden erforderlichen Zeichenkettenwerte für den Partitionsschlüssel und den Sortierschlüssel.

	Partitionsschlüssel	Sortierschlüssel
Basistabelle	`branch-key-id`	`type`

Name des logischen Schlüsselspeichers

Bei der Benennung der DynamoDB-Tabelle, die als Schlüsselspeicher dient, ist es wichtig, den logischen Schlüsselspeicher-Namen, den Sie bei der Konfiguration Ihrer Schlüsselspeicheraktionen angeben, sorgfältig zu berücksichtigen. Der Name des logischen Schlüsselspeichers dient als Kennung für Ihren Schlüsselspeicher und kann nicht geändert werden, nachdem er ursprünglich vom ersten Benutzer definiert wurde. Sie müssen in Ihren Schlüsselspeicher-Aktionen immer denselben logischen Schlüsselspeicher-Namen angeben.

Es muss eine one-to-one Zuordnung zwischen dem DynamoDB-Tabellennamen und dem Namen des logischen Schlüsselspeichers bestehen. Der Name des logischen Schlüsselspeichers ist kryptografisch an alle in der Tabelle gespeicherten Daten gebunden, um DynamoDB-Wiederherstellungsvorgänge zu vereinfachen. Der Name des logischen Schlüsselspeichers kann sich zwar von Ihrem DynamoDB-Tabellennamen unterscheiden, wir empfehlen jedoch dringend, Ihren DynamoDB-Tabellennamen als logischen Schlüsselspeichername anzugeben. Falls sich Ihr Tabellenname nach dem Wiederherstellen Ihrer DynamoDB-Tabelle aus einer Sicherung ändert, kann der Name des logischen Schlüsselspeichers dem neuen DynamoDB-Tabellennamen zugeordnet werden, um sicherzustellen, dass der hierarchische Schlüsselbund weiterhin auf Ihren Schlüsselspeicher zugreifen kann.

Nehmen Sie keine vertraulichen oder sensiblen Informationen in den Namen Ihres logischen Schlüsselspeichers auf. Der Name des logischen Schlüsselspeichers wird in AWS KMS CloudTrail Ereignissen im Klartext als. `tablename`

Nächste Schritte

1. the section called “Schlüsselspeicheraktionen konfigurieren”
2. the section called “Erstellen Sie Zweigschlüssel”
3. Erstellen Sie einen AWS KMS hierarchischen Schlüsselbund

Schlüsselspeicheraktionen konfigurieren

Schlüsselspeicher-Aktionen bestimmen, welche Operationen Ihre Benutzer ausführen können und wie ihr AWS KMS hierarchischer Schlüsselbund die KMS-Schlüssel verwendet, die in Ihrem Schlüsselspeicher zugelassen sind. Das AWS Database Encryption SDK unterstützt die folgenden Konfigurationen für Schlüsselspeicher-Aktionen.

Statisch

Wenn Sie Ihren Schlüsselspeicher statisch konfigurieren, kann der Schlüsselspeicher nur den KMS-Schlüssel verwenden, der dem KMS-Schlüssel-ARN zugeordnet ist, den Sie `kmsConfiguration` bei der Konfiguration Ihrer Schlüsselspeicheraktionen angeben. Eine Ausnahme wird ausgelöst, wenn beim Erstellen, Versionieren oder Abrufen eines Zweigschlüssels auf einen anderen KMS-Schlüssel-ARN gestoßen wird.

Sie können einen KMS-Schlüssel für mehrere Regionen in Ihrem angeben`kmsConfiguration`, aber der gesamte ARN des Schlüssels, einschließlich der Region, wird in den vom KMS-Schlüssel abgeleiteten Zweigschlüsseln beibehalten. Sie können keinen Schlüssel in einer anderen Region angeben. Sie müssen exakt denselben Schlüssel für mehrere Regionen angeben, damit die Werte übereinstimmen.

Wenn Sie Ihre Schlüsselspeicher-Aktionen statisch konfigurieren, können Sie Verwendungsvorgänge (`GetActiveBranchKey``GetBranchKeyVersion`,`GetBeaconKey`) und Verwaltungsvorgänge (`CreateKey`und`VersionKey`) ausführen. `CreateKey`ist eine privilegierte Operation, die Ihrer Schlüsselspeicher-Zulassungsliste einen neuen KMS-Schlüssel-ARN hinzufügen kann. Mit diesem KMS-Schlüssel können neue aktive Zweigschlüssel erstellt werden. Wir empfehlen, den Zugriff auf diesen Vorgang einzuschränken, da ein KMS-Schlüssel, der einmal dem Schlüsselspeicher hinzugefügt wurde, nicht gelöscht werden kann.

Erkennung

Wenn Sie Ihre Schlüsselspeicheraktionen für die Erkennung konfigurieren, kann der Schlüsselspeicher jeden AWS KMS key ARN verwenden, der in Ihrem Schlüsselspeicher zugelassen ist. Es wird jedoch eine Ausnahme ausgelöst, wenn ein KMS-Schlüssel mit mehreren Regionen gefunden wird und die Region im ARN des Schlüssels nicht mit der Region des verwendeten AWS KMS Clients übereinstimmt.

Wenn Sie Ihren Schlüsselspeicher für die Erkennung konfigurieren, können Sie keine administrativen Operationen wie `CreateKey` und `VersionKey` ausführen. Sie können nur die

Verwendungsvorgänge ausführen, die Verschlüsselungs-, Entschlüsselungs-, Signierungs- und Überprüfungsvorgänge ermöglichen. Weitere Informationen finden Sie unter the section called “Implementieren der geringsten Berechtigungen”.

Konfigurieren Sie Ihre Schlüsselspeicher-Aktionen

Bevor Sie Ihre Schlüsselspeicher-Aktionen konfigurieren, stellen Sie sicher, dass die folgenden Voraussetzungen erfüllt sind.

- Ermitteln Sie, welche Operationen Sie ausführen müssen. Weitere Informationen finden Sie unter the section called “Implementieren der geringsten Berechtigungen”.
- Wählen Sie einen Namen für den logischen Schlüsselspeicher

  Es muss eine one-to-one Zuordnung zwischen dem DynamoDB-Tabellennamen und dem Namen des logischen Schlüsselspeichers bestehen. Der Name des logischen Schlüsselspeichers ist kryptografisch an alle in der Tabelle gespeicherten Daten gebunden, um DynamoDB-Wiederherstellungsvorgänge zu vereinfachen. Er kann nicht geändert werden, nachdem er ursprünglich vom ersten Benutzer definiert wurde. Sie müssen in Ihren Schlüsselspeicheraktionen immer denselben logischen Schlüsselspeicher-Namen angeben. Weitere Informationen finden Sie unter logical key store name.

Statische Konfiguration

Im folgenden Beispiel werden Schlüsselspeicheraktionen statisch konfiguriert. Sie müssen den Namen der DynamoDB-Tabelle angeben, die als Ihr Schlüsselspeicher dient, einen logischen Namen für den Schlüsselspeicher und den KMS-Schlüssel-ARN, der einen KMS-Schlüssel mit symmetrischer Verschlüsselung identifiziert.

> **Note**
>
> Berücksichtigen Sie sorgfältig den KMS-Schlüssel-ARN, den Sie bei der statischen Konfiguration Ihres Schlüsselspeicherdienstes angeben. Der `CreateKey` Vorgang fügt den KMS-Schlüssel ARN zu Ihrer Zulassungsliste für den Branch Key Store hinzu. Sobald ein KMS-Schlüssel dem Branch-Schlüsselspeicher hinzugefügt wurde, kann er nicht gelöscht werden.

Java

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

C# / .NET

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

Rust

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

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

Erkennungskonfiguration

Im folgenden Beispiel werden Schlüsselspeicher-Aktionen für die Erkennung konfiguriert. Sie müssen den Namen der DynamoDB-Tabelle, die als Ihr Schlüsselspeicher dient, und einen logischen Schlüsselspeicher-Namen angeben.

Java

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

C# / .NET

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

Rust

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

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

Erstellen Sie einen aktiven Filialschlüssel

Ein Verzweigungsschlüssel ist ein Datenschlüssel AWS KMS key , der von einem abgeleitet ist und den der AWS KMS hierarchische Schlüsselbund verwendet, um die Anzahl der Aufrufe zu reduzieren. AWS KMS Der aktive Zweigschlüssel ist die neueste Version des Zweigschlüssels. Der hierarchische Schlüsselbund generiert für jede Verschlüsselungsanforderung einen eindeutigen Datenschlüssel und verschlüsselt jeden Datenschlüssel mit einem eindeutigen Umschließungsschlüssel, der vom aktiven Zweigschlüssel abgeleitet wird.

Um einen neuen aktiven Zweigschlüssel zu erstellen, müssen Sie Ihre Schlüsselspeicher-Aktionen statisch konfigurieren. `CreateKey`ist eine privilegierte Operation, die den in Ihrer Konfiguration für Schlüsselspeicheraktionen angegebenen KMS-Schlüssel-ARN zu Ihrer Schlüsselspeicher-Zulassungsliste hinzufügt. Anschließend wird der KMS-Schlüssel verwendet, um den neuen aktiven Branch-Schlüssel zu generieren. Wir empfehlen, den Zugriff auf diesen Vorgang einzuschränken, da ein KMS-Schlüssel, der einmal zum Schlüsselspeicher hinzugefügt wurde, nicht gelöscht werden kann.

Wir empfehlen, den `CreateKey` Vorgang über die KeyStore Admin-Oberfläche in der Steuerungsebene Ihrer Anwendung zu verwenden. Dieser Ansatz entspricht den Best Practices für das Schlüsselmanagement.

Erstellen Sie keine Zweigschlüssel auf der Datenebene. Diese Vorgehensweise kann zu folgenden Ergebnissen führen:

- Unnötige Anrufe an AWS KMS
- Mehrere gleichzeitige Aufrufe AWS KMS in Umgebungen mit hoher Parallelität
- Mehrere TransactWriteItems Aufrufe der zugrunde liegenden DynamoDB-Tabelle.

Der `CreateKey` Vorgang beinhaltet eine Zustandsprüfung im `TransactWriteItems` Aufruf, um zu verhindern, dass vorhandene Verzweigungsschlüssel überschrieben werden. Das Erstellen von Schlüsseln auf der Datenebene kann jedoch immer noch zu ineffizienter Ressourcennutzung und potenziellen Leistungsproblemen führen.

Sie können einen KMS-Schlüssel in Ihrem Schlüsselspeicher zulassen, oder Sie können mehrere KMS-Schlüssel zulassen, indem Sie den KMS-Schlüssel-ARN, den Sie in Ihrer Konfiguration für Schlüsselspeicher-Aktionen angeben, aktualisieren und erneut aufrufen`CreateKey`. Wenn Sie mehrere KMS-Schlüssel auf die Zulassungsliste setzen, sollten Ihre Schlüsselspeicher-Benutzer

ihre Schlüsselspeicher-Aktionen für die Erkennung so konfigurieren, dass sie alle Schlüssel auf der Zulassungsliste im Schlüsselspeicher verwenden können, auf die sie Zugriff haben. Weitere Informationen finden Sie unter the section called “Schlüsselspeicheraktionen konfigurieren”.

Erforderliche -Berechtigungen

Um Branch-Schlüssel zu erstellen, benötigen Sie die ReEncrypt Berechtigungen kms: GenerateDataKeyWithoutPlaintext und kms: für den KMS-Schlüssel, der in Ihren Schlüsselspeicher-Aktionen angegeben ist.

Erstellen Sie einen Zweigschlüssel

Der folgende Vorgang erstellt einen neuen aktiven Branch-Schlüssel unter Verwendung des KMS-Schlüssels, den Sie in Ihrer Konfiguration für Schlüsselspeicher-Aktionen angegeben haben, und fügt den aktiven Branch-Schlüssel zur DynamoDB-Tabelle hinzu, die als Ihr Schlüsselspeicher dient.

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

- `branchKeyIdentifier`: definiert ein benutzerdefiniertes`branch-key-id`.

  Um einen benutzerdefinierten zu erstellen`branch-key-id`, müssen Sie dem `encryptionContext` Parameter auch einen zusätzlichen Verschlüsselungskontext hinzufügen.
- `encryptionContext`: definiert einen optionalen Satz nicht geheimer Schlüssel-Wert-Paare, der zusätzliche authentifizierte Daten (AAD) in dem Verschlüsselungskontext bereitstellt, der im kms: -Aufruf enthalten ist. GenerateDataKeyWithoutPlaintext

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

Java

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

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

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

C# / .NET

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

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

Rust

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

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

Zunächst generiert die `CreateKey` Operation die folgenden Werte.

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

Dann ruft der `CreateKey` Vorgang kms: GenerateDataKeyWithoutPlaintext mit der folgenden Anforderung auf.

```
{
```

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

> **Note**
>
> Der `CreateKey` Vorgang erstellt einen aktiven Branch-Schlüssel und einen Beacon-Schlüssel, auch wenn Sie Ihre Datenbank nicht für durchsuchbare Verschlüsselung konfiguriert haben. Beide Schlüssel werden in Ihrem Schlüsselspeicher gespeichert. Weitere Informationen finden Sie unter Verwenden des hierarchischen Schlüsselbundes für durchsuchbare Verschlüsselung.

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

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

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

Drehe deinen aktiven Filialschlüssel

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

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

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

> **Warning**
>
> Das Löschen von Zweigschlüsseln in Testumgebungen ist irreversibel. Sie können gelöschte Zweigschlüssel nicht wiederherstellen. Wenn Sie in Testumgebungen Zweigschlüssel mit derselben ID löschen und neu erstellen, können die folgenden Probleme auftreten:
>
> - Materialien aus früheren Testläufen verbleiben möglicherweise im Cache
> - Einige Testhosts oder Threads verschlüsseln möglicherweise Daten mit gelöschten Zweigschlüsseln
> - Daten, die mit gelöschten Branches verschlüsselt wurden, können nicht entschlüsselt werden
>
> Gehen Sie wie folgt vor, um Verschlüsselungsfehler bei Integrationstests zu verhindern:
>
> - Setzen Sie die hierarchische Schlüsselbundreferenz zurück, bevor Sie neue Zweigschlüssel erstellen ODER

- Verwenden Sie IDs für jeden Test einen eindeutigen Zweigschlüssel

Erforderliche Berechtigungen

Um Zweigschlüssel rotieren zu können, benötigen Sie die ReEncrypt Berechtigungen kms: GenerateDataKeyWithoutPlaintext und kms: für den KMS-Schlüssel, der in Ihren Schlüsselspeicher-Aktionen angegeben ist.

Rotiert einen aktiven Zweigschlüssel

Verwenden Sie die `VersionKey` Operation, um Ihren aktiven Zweigschlüssel zu drehen. Wenn Sie den aktiven Abzweigschlüssel rotieren, wird ein neuer Abzweigschlüssel erstellt, der die vorherige Version ersetzt. Das `branch-key-id` ändert sich nicht, wenn Sie den aktiven Abzweigschlüssel drehen. Sie müssen den Schlüssel angeben`branch-key-id`, der den aktuell aktiven Abzweigschlüssel identifiziert, wenn Sie anrufen`VersionKey`.

Java

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

C# / .NET

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

Rust

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

Schlüsselringe

Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Das AWS Database Encryption SDK verwendet Schlüsselringe, um die Envelope-Verschlüsselung durchzuführen. Schlüsselbunde generieren, verschlüsseln und entschlüsseln Datenschlüssel. Schlüsselringe bestimmen die Quelle der eindeutigen Datenschlüssel, die jeden verschlüsselten Datensatz schützen, und der Umschließungsschlüssel, mit denen dieser Datenschlüssel verschlüsselt wird. Sie geben bei der Verschlüsselung einen Schlüsselbund und bei der Entschlüsselung denselben oder einen anderen Schlüsselbund an.

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

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

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

Das AWS Database Encryption SDK bietet verschiedene Schlüsselringe und Schlüsselbundkonfigurationen, und Sie können Ihre eigenen benutzerdefinierten Schlüsselbunde erstellen. Sie können auch einen Mehrfachschlüsselbund erstellen, der einen oder mehrere Schlüsselanhänger desselben oder eines anderen Typs enthält.

Themen

Funktionsweise von Schlüsselbunden

Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Wenn Sie ein Feld in Ihrer Datenbank verschlüsseln und signieren, fragt das AWS Database Encryption SDK den Schlüsselbund nach Verschlüsselungsmaterialien. Der Schlüsselbund gibt einen Klartext-Datenschlüssel zurück, eine Kopie des Datenschlüssels, der durch jeden der Umschließungsschlüssel im Schlüsselbund verschlüsselt wurde, und einen MAC-Schlüssel, der dem Datenschlüssel zugeordnet ist. Das AWS Database Encryption SDK verwendet den Klartext-Schlüssel, um die Daten zu verschlüsseln, und entfernt dann den Klartext-Datenschlüssel so schnell wie möglich aus dem Speicher. Anschließend fügt das AWS Database Encryption SDK eine Materialbeschreibung hinzu, die die verschlüsselten Datenschlüssel und andere Informationen wie Verschlüsselungs- und Signieranweisungen enthält. Das AWS Database Encryption SDK verwendet den MAC-Schlüssel, um Hash-Based Message Authentication Codes (HMACs) über die Kanonisierung der Materialbeschreibung und aller mit oder markierten Feldern zu berechnen. `ENCRYPT_AND_SIGN` `SIGN_ONLY`

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

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

Sie können einen einzelnen Schlüsselbund verwenden oder Schlüsselbunde desselben Typs oder eines anderen Typs in einem Multi-Schlüsselbund kombinieren. Wenn Sie Daten verschlüsseln, gibt der Mehrfachschlüsselbund eine Kopie des Datenschlüssels zurück, der mit allen Schlüsseln in allen Schlüsselbunden, aus denen der Mehrfachschlüsselbund besteht, und einem MAC-Schlüssel, der dem Datenschlüssel zugeordnet ist, verschlüsselt wurde. Sie können die Daten mithilfe eines Schlüsselbundes entschlüsseln, wobei jeder der Schlüssel im Mehrfachschlüsselbund eingeschlossen ist.

AWS KMS Schlüsselringe

> Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

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

Sie können auch einen symmetrischen KMS-Schlüssel für mehrere Regionen in einem Schlüsselbund verwenden. AWS KMS Weitere Informationen und Beispiele zur Verwendung von AWS KMS keys Multiregion finden Sie unter. Multi-Region verwenden AWS KMS keys Informationen zu Schlüsseln für mehrere Regionen finden Sie unter Verwenden von Schlüsseln für mehrere Regionen im AWS Key Management Service Entwicklerhandbuch.

AWS KMS Schlüsselanhänger können zwei Arten von Schlüssellösungen beinhalten:

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

Sie benötigen einen Generatorschlüssel, um Datensätze zu verschlüsseln. Wenn ein AWS KMS Schlüsselbund nur einen AWS KMS Schlüssel hat, wird dieser Schlüssel verwendet, um den Datenschlüssel zu generieren und zu verschlüsseln.

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

Themen

AWS KMS Erforderliche Berechtigungen für Schlüsselanhänger

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

- Um mit einem AWS KMS Schlüsselbund zu verschlüsseln, benötigen Sie die kms: GenerateDataKey -Berechtigung für den Generatorschlüssel. Sie benötigen die kms:Encrypt-Berechtigung für alle zusätzlichen Schlüssel im Schlüsselbund. AWS KMS
- Um mit einem AWS KMS Schlüsselbund zu entschlüsseln, benötigen Sie die kms:Decrypt-Berechtigung für mindestens einen Schlüssel im Schlüsselbund. AWS KMS
- Um mit einem Mehrfachschlüsselbund zu verschlüsseln, der aus Schlüsselbunden besteht, benötigen Sie die kms-Berechtigung für den AWS KMS Generatorschlüssel im Generator-

Schlüsselbund. GenerateDataKey Sie benötigen die kms:Encrypt-Berechtigung für alle anderen Schlüssel in allen anderen Schlüsselbunden. AWS KMS

- Um mit einem asymmetrischen AWS KMS RSA-Schlüsselbund zu verschlüsseln, benötigen Sie kms: GenerateDataKey oder kms:Encrypt nicht, da Sie bei der Erstellung des Schlüsselbunds das Material der öffentlichen Schlüssel angeben müssen, das Sie für die Verschlüsselung verwenden möchten. Bei der Verschlüsselung mit diesem Schlüsselbund werden keine Anrufe getätigt. AWS KMS Um mit einem asymmetrischen AWS KMS RSA-Schlüsselbund zu entschlüsseln, benötigen Sie die kms:Decrypt-Berechtigung.

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

Identifizierung AWS KMS keys in einem AWS KMS Schlüsselbund

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

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

- Um mit einem AWS KMS Schlüsselbund zu verschlüsseln, können Sie eine Schlüssel-ID, einen Schlüssel-ARN, einen Aliasnamen oder einen Alias-ARN verwenden, um Daten zu verschlüsseln.

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

- Um mit einem AWS KMS Schlüsselbund zu entschlüsseln, müssen Sie einen Schlüssel-ARN zur Identifizierung verwenden. AWS KMS keys Details hierzu finden Sie unter Auswahl von Wraping-Schlüsseln.

- In einem Schlüsselbund, der für die Verschlüsselung und Entschlüsselung verwendet wird, müssen Sie einen Schlüssel-ARN verwenden, um AWS KMS keys zu identifizieren.

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

- Das AWS Database Encryption SDK ruft den Schlüssel ARN des Schlüssels ab AWS KMS key , der den Datenschlüssel verschlüsselt hat, aus den Metadaten der Materialbeschreibung.
- Das AWS Database Encryption SDK durchsucht den Schlüsselbund für die Entschlüsselung nach einem AWS KMS key ARN mit einem passenden Schlüssel.
- Wenn es einen ARN AWS KMS key mit einem passenden Schlüssel im Schlüsselbund findet, fordert das AWS Database Encryption SDK auf, den KMS-Schlüssel AWS KMS zum Entschlüsseln des verschlüsselten Datenschlüssels zu verwenden.
- Andernfalls springt er zum nächsten verschlüsselten Datenschlüssel, falls vorhanden.

Einen Schlüsselbund erstellen AWS KMS

Sie können jeden AWS KMS Schlüsselbund mit einem AWS KMS key oder mehreren Schlüsselbändern AWS KMS keys im selben oder einem anderen AWS-Konten und konfigurieren. AWS-Regionen Der AWS KMS key muss ein symmetrischer Verschlüsselungsschlüssel (`SYMMETRIC_DEFAULT`) oder ein asymmetrischer RSA-KMS-Schlüssel sein. Sie können auch einen KMS-Schlüssel für mehrere Regionen mit symmetrischer Verschlüsselung verwenden. Sie können einen oder mehrere AWS KMS Schlüsselbunde in einem Mehrfachschlüsselbund verwenden.

Sie können einen AWS KMS Schlüsselbund erstellen, der Daten ver- und entschlüsselt, oder Sie können AWS KMS Schlüsselbunde speziell für das Verschlüsseln oder Entschlüsseln erstellen. Wenn Sie einen AWS KMS Schlüsselbund zum Verschlüsseln von Daten erstellen, müssen Sie einen Generatorschlüssel angeben. Dieser wird verwendet, um einen Klartext-Datenschlüssel zu generieren und AWS KMS key diesen zu verschlüsseln. Der Datenschlüssel hat mathematisch nichts mit dem KMS-Schlüssel zu tun. Wenn Sie möchten, können Sie dann weitere angeben, AWS KMS keys die denselben Klartext-Datenschlüssel verschlüsseln. Um ein durch diesen Schlüsselbund geschütztes verschlüsseltes Feld zu entschlüsseln, muss der von Ihnen verwendete Entschlüsselungsschlüsselbund mindestens einen der im Schlüsselbund AWS KMS keys definierten

Werte enthalten, oder nein. AWS KMS keys(Ein AWS KMS Schlüsselbund ohne AWS KMS keys wird als Discovery-Schlüsselbund bezeichnet.)AWS KMS

Alle Schlüssel, die in einen Verschlüsselungsschlüsselbund oder einen Mehrfachschlüsselbund eingeschlossen werden, müssen in der Lage sein, den Datenschlüssel zu verschlüsseln. Wenn ein Umschließungsschlüssel nicht verschlüsselt werden kann, schlägt die Verschlüsselungsmethode fehl. Daher muss der Anrufer über die erforderlichen Berechtigungen für alle Schlüssel im Schlüsselbund verfügen. Wenn Sie einen Discovery-Schlüsselbund verwenden, um Daten allein oder in einem Mehrfachschlüsselbund zu verschlüsseln, schlägt der Verschlüsselungsvorgang fehl.

In den folgenden Beispielen wird die `CreateAwsKmsMrkMultiKeyring` Methode verwendet, um einen AWS KMS Schlüsselbund mit einem symmetrischen Verschlüsselungs-KMS-Schlüssel zu erstellen. Die `CreateAwsKmsMrkMultiKeyring` Methode erstellt den AWS KMS Client automatisch und stellt sicher, dass der Schlüsselbund sowohl Schlüssel mit einer Region als auch Schlüssel mit mehreren Regionen korrekt verarbeitet. In diesen Beispielen wird ein Schlüssel verwendet, ARNs um die KMS-Schlüssel zu identifizieren. Details hierzu finden Sie unter Identifizierung AWS KMS keys in einem AWS KMS Schlüsselbund

Java

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

C# / .NET

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var createAwsKmsMrkMultiKeyringInput = new CreateAwsKmsMrkMultiKeyringInput
{
    Generator = kmsKeyArn
};
var awsKmsMrkMultiKeyring =
 matProv.CreateAwsKmsMrkMultiKeyring(createAwsKmsMrkMultiKeyringInput);
```

Rust

```
let provider_config = MaterialProvidersConfig::builder().build()?;
let mat_prov = client::Client::from_conf(provider_config)?;
let kms_keyring = mat_prov
    .create_aws_kms_mrk_multi_keyring()
    .generator(kms_key_id)
    .send()
    .await?;
```

In den folgenden Beispielen `CreateAwsKmsRsaKeyring` wird die Methode verwendet, um einen AWS KMS Schlüsselbund mit einem asymmetrischen RSA-KMS-Schlüssel zu erstellen. Um einen asymmetrischen AWS KMS RSA-Schlüsselbund zu erstellen, geben Sie die folgenden Werte an.

- `kmsClient`: einen neuen Client erstellen AWS KMS
- `kmsKeyID`: der Schlüssel-ARN, der Ihren asymmetrischen RSA-KMS-Schlüssel identifiziert
- `publicKey`: eine Datei ByteBuffer aus einer UTF-8-codierten PEM-Datei, die den öffentlichen Schlüssel des Schlüssels darstellt, an den Sie übergeben haben `kmsKeyID`
- `encryptionAlgorithm`: Der Verschlüsselungsalgorithmus muss oder sein `RSAES_OAEP_SHA_256` `RSAES_OAEP_SHA_1`

Java

```
 final MaterialProviders matProv = MaterialProviders.builder()
    .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
    .build();
final CreateAwsKmsRsaKeyringInput createAwsKmsRsaKeyringInput =
    CreateAwsKmsRsaKeyringInput.builder()
        .kmsClient(KmsClient.create())
        .kmsKeyId(rsaKMSKeyArn)
        .publicKey(publicKey)
        .encryptionAlgorithm(EncryptionAlgorithmSpec.RSAES_OAEP_SHA_256)
        .build();
IKeyring awsKmsRsaKeyring =
 matProv.CreateAwsKmsRsaKeyring(createAwsKmsRsaKeyringInput);
```

C# / .NET

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

```
var createAwsKmsRsaKeyringInput = new CreateAwsKmsRsaKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = rsaKMSKeyArn,
    PublicKey = publicKey,
    EncryptionAlgorithm = EncryptionAlgorithmSpec.RSAES_OAEP_SHA_256
};
IKeyring awsKmsRsaKeyring =
 matProv.CreateAwsKmsRsaKeyring(createAwsKmsRsaKeyringInput);
```

Rust

```
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;
let sdk_config =
 aws_config::load_defaults(aws_config::BehaviorVersion::latest()).await;
let kms_rsa_keyring = mpl
    .create_aws_kms_rsa_keyring()
    .kms_key_id(rsa_kms_key_arn)
    .public_key(public_key)

 .encryption_algorithm(aws_sdk_kms::types::EncryptionAlgorithmSpec::RsaesOaepSha256)
    .kms_client(aws_sdk_kms::Client::new(&sdk_config))
    .send()
    .await?;
```

Multi-Region verwenden AWS KMS keys

Sie können Multiregion AWS KMS keys als Schlüssel im AWS Database Encryption SDK verwenden. Wenn Sie mit einem Schlüssel für mehrere Regionen in einem verschlüsseln AWS-Region, können Sie die Verschlüsselung mit einem zugehörigen Schlüssel für mehrere Regionen in einer anderen Region durchführen. AWS-Region

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

mehrere Regionen finden Sie unter Verwenden von Schlüsseln für mehrere Regionen im Entwicklerhandbuch.AWS Key Management Service

Um Schlüssel für mehrere Regionen zu unterstützen, enthält das AWS Database Encryption SDK Schlüsselringe. AWS KMS multi-Region-aware Die `CreateAwsKmsMrkMultiKeyring` Methode unterstützt sowohl Schlüssel mit einer Region als auch Schlüssel mit mehreren Regionen.

- Bei Schlüsseln mit nur einer Region verhält sich das multi-Region-aware Symbol genauso wie der Schlüsselbund mit nur einer Region. AWS KMS Es versucht, Chiffretext nur mit dem Schlüssel für eine einzelne Region zu entschlüsseln, mit dem die Daten verschlüsselt wurden. Um den Umgang mit dem AWS KMS Schlüsselbund zu vereinfachen, empfehlen wir, `CreateAwsKmsMrkMultiKeyring` diese Methode immer dann zu verwenden, wenn Sie einen KMS-Schlüssel mit symmetrischer Verschlüsselung verwenden.
- Bei Schlüsseln mit mehreren Regionen versucht das multi-Region-aware Symbol, Chiffretext mit demselben Schlüssel für mehrere Regionen zu entschlüsseln, mit dem die Daten verschlüsselt wurden, oder mit dem zugehörigen Schlüssel für mehrere Regionen in der von Ihnen angegebenen Region.

In den multi-Region-aware Schlüsselbunden, die mehr als einen KMS-Schlüssel benötigen, können Sie mehrere Schlüssel für eine Region und mehrere Regionen angeben. Sie können jedoch nur einen Schlüssel aus jedem Satz verwandter Schlüssel für mehrere Regionen angeben. Wenn Sie mehr als einen Schlüsselbezeichner mit derselben Schlüssel-ID angeben, schlägt der Konstruktoraufruf fehl.

In den folgenden Beispielen wird ein AWS KMS Schlüsselbund mit einem KMS-Schlüssel für mehrere Regionen erstellt. In den Beispielen wird ein Schlüssel mit mehreren Regionen als Generatorschlüssel und ein Schlüssel mit nur einer Region als untergeordneter Schlüssel angegeben.

Java

```
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);
```

C# / .NET

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var createAwsKmsMrkMultiKeyringInput = new CreateAwsKmsMrkMultiKeyringInput
{
    Generator = multiRegionKeyArn,
    KmsKeyIds = new List<String> { kmsKeyArn }
};
var awsKmsMrkMultiKeyring =
 matProv.CreateAwsKmsMrkMultiKeyring(createAwsKmsMrkMultiKeyringInput);
```

Rust

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

let aws_kms_mrk_multi_keyring = mpl
    .create_aws_kms_mrk_multi_keyring()
    .generator(multiRegion_key_arn)
    .kms_key_ids(vec![key_arn.to_string()])
    .send()
    .await?;
```

Wenn Sie AWS KMS Schlüsselringe mit mehreren Regionen verwenden, können Sie Chiffretext im strikten Modus oder im Discover-Modus entschlüsseln. Um den Chiffretext im strikten Modus zu entschlüsseln, instanziieren Sie das multi-Region-aware Symbol mit dem Schlüssel ARN des zugehörigen Multi-Region-Schlüssels in der Region, in der Sie den Chiffretext entschlüsseln. Wenn Sie den Schlüssel-ARN eines zugehörigen Multi-Region-Schlüssels in einer anderen Region angeben (z. B. der Region, in der der Datensatz verschlüsselt wurde), ruft das multi-Region-aware Symbol diesen Schlüssel regionsübergreifend auf. AWS KMS key

Bei der Entschlüsselung im strikten Modus benötigt das multi-Region-aware Symbol einen Schlüssel-ARN. Es akzeptiert nur einen Schlüssel-ARN aus jedem Satz verwandter Schlüssel für mehrere Regionen.

Sie können auch im Discovery-Modus mit Schlüsseln für AWS KMS mehrere Regionen entschlüsseln. Beim Entschlüsseln im Discovery-Modus geben Sie keine an. AWS KMS

keys(Informationen zu Schlüsselanhängern für die AWS KMS Erkennung einzelner Regionen finden Sie unter.) Verwenden Sie einen Discovery-Schlüsselbund AWS KMS

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

Verwenden Sie einen Discovery-Schlüsselbund AWS KMS

Beim Entschlüsseln empfiehlt es sich, die Umschließungsschlüssel anzugeben, die das AWS Database Encryption SDK verwenden kann. Um dieser bewährten Methode zu folgen, verwenden Sie einen Schlüsselbund für die AWS KMS Entschlüsselung, der die AWS KMS Umschließungsschlüssel auf die von Ihnen angegebenen beschränkt. Sie können jedoch auch einen AWS KMS Discovery-Schlüsselbund erstellen, d. h. einen Schlüsselbund, der keine AWS KMS Schlüssel zum Umschließen von Schlüsseln festlegt.

Das AWS Database Encryption SDK bietet einen AWS KMS Standard-Discovery-Schlüsselbund und einen Discovery-Schlüsselbund für Schlüssel mit mehreren Regionen. AWS KMS Hinweise zur Verwendung von Schlüsseln für mehrere Regionen mit dem AWS Database Encryption SDK finden Sie unter. Multi-Region verwenden AWS KMS keys

Da er keine Umschließungsschlüssel spezifiziert, kann ein Discovery-Schlüsselbund keine Daten verschlüsseln. Wenn Sie einen Discovery-Schlüsselbund verwenden, um Daten allein oder in einem Mehrfachschlüsselbund zu verschlüsseln, schlägt der Verschlüsselungsvorgang fehl.

Beim Entschlüsseln ermöglicht ein Discovery-Schlüsselbund dem AWS Database Encryption SDK, jeden verschlüsselten Datenschlüssel mithilfe des verschlüsselten Schlüssels AWS KMS zu entschlüsseln, unabhängig davon, wem AWS KMS key dieser gehört oder wer Zugriff darauf hat. AWS KMS key Der Anruf ist nur erfolgreich, wenn der Anrufer die Erlaubnis für hat. `kms:Decrypt` AWS KMS key

> **Important**
>
> Wenn Sie einen AWS KMS Discovery-Schlüsselbund in einen Mehrschlüsselbund für die Entschlüsselung aufnehmen, setzt der Discovery-Schlüsselbund alle KMS-Schlüsseleinschränkungen außer Kraft, die durch andere Schlüsselbunde im Mehrfachschlüsselbund festgelegt wurden. Der Mehrfachschlüsselbund verhält sich wie

sein am wenigsten restriktiver Schlüsselbund. Wenn Sie einen Discovery-Schlüsselbund verwenden, um Daten allein oder in einem Mehrfachschlüsselbund zu verschlüsseln, schlägt der Verschlüsselungsvorgang fehl

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

- Authentizität — Ein AWS KMS Discovery-Schlüsselbund kann jeden Schlüsselbund verwenden AWS KMS key , der zur Verschlüsselung eines Datenschlüssels in der Materialbeschreibung verwendet wurde, sofern der Anrufer die Erlaubnis hat, diesen Schlüssel zum Entschlüsseln zu verwenden. AWS KMS key Dies ist möglicherweise nicht der AWS KMS key , den der Anrufer verwenden möchte. Beispielsweise könnte einer der verschlüsselten Datenschlüssel unter einer weniger sicheren Methode verschlüsselt worden sein AWS KMS key , die jeder verwenden kann.
- Latenz und Leistung — Ein AWS KMS Discovery-Schlüsselbund ist möglicherweise deutlich langsamer als andere Schlüsselbunde, da das AWS Database Encryption SDK versucht, alle verschlüsselten Datenschlüssel zu entschlüsseln, einschließlich der Schlüssel, die AWS KMS keys in anderen AWS-Konten und Regionen verschlüsselt wurden, und für die der Anrufer keine Berechtigung hat, sie zur Entschlüsselung zu verwenden. AWS KMS keys

Wenn Sie einen Discovery-Schlüsselbund verwenden, empfehlen wir Ihnen, einen Discovery-Filter zu verwenden, um die KMS-Schlüssel, die verwendet werden können, auf diejenigen in bestimmten Partitionen und Partitionen zu beschränken. AWS-Konten Hilfe bei der Suche nach Ihrer Konto-ID und Partition finden Sie unter Ihre AWS-Konto Identifikatoren und das ARN-Format in der Allgemeine AWS-Referenz.

In den folgenden Codebeispielen wird ein AWS KMS Discovery-Schlüsselbund mit einem Discovery-Filter instanziiert, der die KMS-Schlüssel, die das AWS Database Encryption SDK verwenden kann, auf diejenigen in der `aws` Partition und im Beispielkonto beschränkt. `111122223333`

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

Java

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

C# / .NET

```
// Create discovery filter
var discoveryFilter = new DiscoveryFilter
{
    Partition = "aws",
    AccountIds = 111122223333
};
// Create the discovery keyring
var createAwsKmsMrkDiscoveryMultiKeyringInput = new
 CreateAwsKmsMrkDiscoveryMultiKeyringInput
{
    DiscoveryFilter = discoveryFilter
};
var decryptKeyring =
 matProv.CreateAwsKmsMrkDiscoveryMultiKeyring(createAwsKmsMrkDiscoveryMultiKeyringInput);
```

Rust

```
// Create discovery filter
let discovery_filter = DiscoveryFilter::builder()
    .partition("aws")
    .account_ids(111122223333)
    .build()?;

// Create the discovery keyring
let decrypt_keyring = mpl
    .create_aws_kms_mrk_discovery_multi_keyring()
```

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

Verwenden Sie einen AWS KMS Regional Discovery-Schlüsselbund

Ein AWS KMS Regional Discovery-Schlüsselbund ist ein Schlüsselbund, der ARNs die KMS-Schlüssel nicht spezifiziert. Stattdessen ermöglicht er dem AWS Database Encryption SDK die Entschlüsselung, wobei nur die KMS-Schlüssel verwendet werden. AWS-Regionen

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

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

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

Der regionale Discovery-Schlüsselbund im AWS Database Encryption SDK versucht, nur mit KMS-Schlüsseln in der angegebenen Region zu entschlüsseln. Wenn Sie einen Discovery-Schlüsselbund verwenden, konfigurieren Sie die Region auf dem Client. AWS KMS Diese Implementierungen des

AWS Database Encryption SDK filtern KMS-Schlüssel nicht nach Region, aber AWS KMS eine Entschlüsselungsanforderung für KMS-Schlüssel außerhalb der angegebenen Region schlägt fehl.

Wenn Sie einen Discovery-Schlüsselbund verwenden, empfehlen wir die Verwendung eines Discovery-Filters, um die bei der Entschlüsselung verwendeten KMS-Schlüssel auf die in den angegebenen Partitionen verwendeten KMS-Schlüssel zu beschränken. AWS-Konten

Mit dem folgenden Code wird beispielsweise ein AWS KMS regionaler Discovery-Schlüsselbund mit einem Discovery-Filter erstellt. Dieser Schlüsselbund beschränkt das AWS Database Encryption SDK auf KMS-Schlüssel im Konto 111122223333 in der Region USA West (Oregon) (us-west-2).

Java

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

C# / .NET

```
// Create discovery filter
var discoveryFilter = new DiscoveryFilter
{
    Partition = "aws",
    AccountIds = 111122223333
};
// Create the discovery keyring
var createAwsKmsMrkDiscoveryMultiKeyringInput = new
 CreateAwsKmsMrkDiscoveryMultiKeyringInput
{
    DiscoveryFilter = discoveryFilter,
    Regions = us-west-2
};
```

```
var decryptKeyring =
 matProv.CreateAwsKmsMrkDiscoveryMultiKeyring(createAwsKmsMrkDiscoveryMultiKeyringInput);
```

Rust

```
// Create discovery filter
let discovery_filter = DiscoveryFilter::builder()
    .partition("aws")
    .account_ids(111122223333)
    .build()?;

// Create the discovery keyring
let decrypt_keyring = mpl
    .create_aws_kms_mrk_discovery_multi_keyring()
    .discovery_filter(discovery_filter)
    .regions(us-west-2)
    .send()
    .await?;
```

AWS KMS Hierarchische Schlüsselanhänger

Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt . AWS Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

> **Note**
>
> Seit dem 24. Juli 2023 werden Branch-Schlüssel, die während der Developer Preview erstellt wurden, nicht unterstützt. Erstellen Sie neue Branch-Schlüssel, um den Schlüsselspeicher, den Sie während der Developer Preview erstellt haben, weiterhin verwenden zu können.

Mit dem AWS KMS hierarchischen Schlüsselbund können Sie Ihre kryptografischen Materialien mit einem KMS-Schlüssel mit symmetrischer Verschlüsselung schützen, ohne AWS KMS jedes Mal, wenn Sie einen Datensatz ver- oder entschlüsseln, erneut aufrufen zu müssen. Es ist eine gute Wahl für Anwendungen, bei denen die Anzahl der Aufrufe minimiert werden muss AWS

KMS, und für Anwendungen, die kryptografisches Material wiederverwenden können, ohne ihre Sicherheitsanforderungen zu verletzen.

Der hierarchische Schlüsselbund ist eine Lösung zum Zwischenspeichern von kryptografischem Material, die die Anzahl der AWS KMS Aufrufe reduziert, indem AWS KMS geschützte Branch-Schlüssel verwendet werden, die in einer Amazon DynamoDB-Tabelle gespeichert sind, und anschließend das bei Verschlüsselungs- und Entschlüsselungsvorgängen verwendete Zweigschlüsselmaterial lokal zwischengespeichert wird. Die DynamoDB-Tabelle dient als Schlüsselspeicher für die Verwaltung und den Schutz von Zweigschlüsseln. Sie speichert den aktiven Branch-Schlüssel und alle vorherigen Versionen des Branch-Schlüssels. Der aktive Zweigschlüssel ist die neueste Version des Zweigschlüssels. Der hierarchische Schlüsselbund verwendet für jede Verschlüsselungsanforderung einen eindeutigen Datenverschlüsselungsschlüssel und verschlüsselt jeden Datenverschlüsselungsschlüssel mit einem eindeutigen Umschließungsschlüssel, der vom aktiven Filialschlüssel abgeleitet wird. Der hierarchische Schlüsselbund hängt von der Hierarchie ab, die zwischen aktiven Zweigschlüsseln und ihren abgeleiteten Umschließungsschlüsseln festgelegt wurde.

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

Wenn Sie den hierarchischen Schlüsselbund instanziieren, erstellt er einen lokalen Cache. Sie geben ein Cache-Limit an, das die maximale Zeitspanne definiert, für die die Branch-Schlüsselmaterialien im lokalen Cache gespeichert werden, bevor sie ablaufen und aus dem Cache entfernt werden. Der hierarchische Schlüsselbund führt einen AWS KMS Aufruf durch, um den Zweigschlüssel zu entschlüsseln und die Zweigschlüsselmaterialien zusammenzustellen, wenn a zum ersten Mal in einem Vorgang angegeben `branch-key-id` wird. Anschließend werden die Materialien der Verzweigungsschlüssel im lokalen Cache gespeichert und für alle Verschlüsselungs- und Entschlüsselungsvorgänge, die dies spezifizieren, wiederverwendet, bis das Cache-Limit abläuft. `branch-key-id` Das Speichern von Zweigschlüsselmaterialien im lokalen Cache reduziert die Anzahl der Aufrufe. AWS KMS Stellen Sie sich beispielsweise ein Cache-Limit von 15 Minuten vor. Wenn Sie innerhalb dieses Cache-Limits 10.000 Verschlüsselungsvorgänge ausführen, müsste der herkömmliche AWS KMS Schlüsselbund 10.000 AWS KMS Aufrufe tätigen, um 10.000 Verschlüsselungsvorgänge zu erfüllen. Wenn Sie einen aktiven Schlüsselbund haben`branch-`

`key-id`, muss der hierarchische Schlüsselbund nur einen AWS KMS Aufruf tätigen, um 10.000 Verschlüsselungsvorgänge abzuwickeln.

Der lokale Cache trennt Verschlüsselungsmaterialien von Entschlüsselungsmaterialien. Die Verschlüsselungsmaterialien werden aus dem aktiven Zweigschlüssel zusammengesetzt und für alle Verschlüsselungsvorgänge wiederverwendet, bis das Cache-Limit abgelaufen ist. Die Entschlüsselungsmaterialien werden aus der Zweigschlüssel-ID und der Version zusammengestellt, die in den Metadaten des verschlüsselten Felds identifiziert wurden, und sie werden für alle Entschlüsselungsvorgänge im Zusammenhang mit der Branch-Schlüssel-ID und -version wiederverwendet, bis das Cache-Limit abläuft. Im lokalen Cache können mehrere Versionen desselben Zweigschlüssels gleichzeitig gespeichert werden. Wenn der lokale Cache für die Verwendung von konfiguriert istbranch key ID supplier, kann er auch Zweigschlüsselmaterial von mehreren aktiven Zweigschlüsseln gleichzeitig speichern.

> **Note**
>
> Alle Erwähnungen des hierarchischen Schlüsselbundes im AWS Database Encryption SDK beziehen sich auf den AWS KMS hierarchischen Schlüsselbund.

Themen

Funktionsweise

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

Verschlüsseln und signieren

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

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

Die Verschlüsselungsmethode verwendet die Verschlüsselungsmaterialien, um den Datensatz zu verschlüsseln und zu signieren. Weitere Informationen darüber, wie Datensätze im AWS Database Encryption SDK verschlüsselt und signiert werden, finden Sie unter Verschlüsseln und Signieren.

Entschlüsseln und verifizieren

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

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

   Weitere Informationen finden Sie unter AWS KMS Technische Details zum hierarchischen Schlüsselbund.
3. Mit dem hierarchischen Schlüsselbund wird geprüft, ob sich im lokalen Cache gültiges Zweigschlüsselmaterial befindet, das mit der in Schritt 2 identifizierten Version des Zweigschlüssels übereinstimmt. Wenn gültiges Schlüsselmaterial für die Zweige vorhanden ist, fährt der Schlüsselbund mit Schritt 6 fort.
4. Wenn kein gültiges Material für Zweigschlüssel vorhanden ist, fragt der hierarchische Schlüsselbund den Schlüsselspeicher nach dem Zweigschlüssel ab, der mit der in Schritt 2 identifizierten Version des Zweigschlüssels übereinstimmt.

   a. Der Schlüsselspeicher ruft AWS KMS zur Entschlüsselung des Zweigschlüssels auf und gibt den aktiven Zweigschlüssel im Klartext zurück. Daten, die den aktiven Zweigschlüssel identifizieren, werden serialisiert, um zusätzliche authentifizierte Daten (AAD) beim Entschlüsselungsaufruf von bereitzustellen. AWS KMS

   b. Der Schlüsselspeicher gibt den Klartext-Zweigschlüssel und die ihn identifizierenden Daten zurück, z. B. die Version des Zweigschlüssels.
5. Der hierarchische Schlüsselbund stellt die Schlüsselmaterialien der Zweige zusammen (die Version mit dem Zweigschlüssel im Klartext und der Zweigschlüsselversion) und speichert eine Kopie davon im lokalen Cache.
6. Der hierarchische Schlüsselbund verwendet die zusammengestellten Zweigschlüsselmaterialien und das in Schritt 2 identifizierte 16-Byte-Salt, um den eindeutigen Wrapping-Schlüssel zu reproduzieren, mit dem der Datenschlüssel verschlüsselt wurde.
7. Der hierarchische Schlüsselbund verwendet den reproduzierten Wrapping-Schlüssel, um den Datenschlüssel zu entschlüsseln, und gibt den Klartext-Datenschlüssel zurück.

Bei der Entschlüsselungsmethode werden die Entschlüsselungsmaterialien und der Klartext-Datenschlüssel verwendet, um den Datensatz zu entschlüsseln und zu verifizieren. Weitere

Informationen darüber, wie Datensätze im AWS Database Encryption SDK entschlüsselt und verifiziert werden, finden Sie unter Entschlüsseln und Überprüfen.

Voraussetzungen

Bevor Sie einen hierarchischen Schlüsselbund erstellen und verwenden, stellen Sie sicher, dass die folgenden Voraussetzungen erfüllt sind.

- Sie oder Ihr Schlüsselspeicheradministrator haben einen Schlüsselspeicher und mindestens einen aktiven Zweigschlüssel erstellt.
- Sie haben Ihre Schlüsselspeicheraktionen konfiguriert.

> **Note**
>
> Wie Sie Ihre Schlüsselspeicher-Aktionen konfigurieren, bestimmt, welche Operationen Sie ausführen können und welche KMS-Schlüssel der hierarchische Schlüsselbund verwenden kann. Weitere Informationen finden Sie unter Schlüsselspeicher-Aktionen.

- Sie verfügen über die erforderlichen AWS KMS Berechtigungen, um auf den Schlüsselspeicher und die Zweigschlüssel zuzugreifen und diese zu verwenden. Weitere Informationen finden Sie unter the section called “Erforderliche Berechtigungen”.
- Sie haben die unterstützten Cachetypen überprüft und den Cachetyp konfiguriert, der Ihren Anforderungen am besten entspricht. Weitere Informationen finden Sie unter the section called “Wählen Sie einen Cache”.

Erforderliche Berechtigungen

Das AWS Database Encryption SDK benötigt kein AWS-Konto und ist auch von keinem abhängig AWS-Service. Um einen hierarchischen Schlüsselbund verwenden zu können, benötigen Sie jedoch mindestens die folgenden Mindestberechtigungen für die symmetrische (n) Verschlüsselung AWS KMS key(en) in Ihrem Schlüsselspeicher. AWS-Konto

- Um Daten mit dem hierarchischen Schlüsselbund zu ver- und entschlüsseln, benötigen Sie kms:Decrypt.
- Um Zweigschlüssel zu erstellen und zu rotieren, benötigen Sie kms: und kms:. GenerateDataKeyWithoutPlaintext ReEncrypt

Weitere Informationen zur Steuerung des Zugriffs auf Ihre Filialschlüssel und Ihren Schlüsselspeicher finden Sie unterthe section called “Implementieren der geringsten Berechtigungen”.

Wählen Sie einen Cache

Durch den hierarchischen Schlüsselbund wird die Anzahl der Aufrufe reduziert, AWS KMS indem die für Ver- und Entschlüsselungsvorgänge verwendeten Zweigschlüsselmaterialien lokal zwischengespeichert werden. Bevor Sie Ihren hierarchischen Schlüsselbund erstellen, müssen Sie entscheiden, welche Art von Cache Sie verwenden möchten. Sie können den Standard-Cache verwenden oder den Cache an Ihre Bedürfnisse anpassen.

Der hierarchische Schlüsselbund unterstützt die folgenden Cachetypen:

- the section called “Standard-Cache”
- the section called “MultiThreaded Cache”
- the section called “StormTracking Zwischenspeicher”
- the section called “Gemeinsam genutzter Cache”

Standard-Cache

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

Standard und StormTracking Caches unterstützen dasselbe Threading-Modell, aber Sie müssen nur die Eingangskapazität angeben, um den Standard-Cache verwenden zu können. Für detailliertere Cache-Anpassungen verwenden Sie den. the section called “StormTracking Zwischenspeicher”

Sofern Sie nicht die Anzahl der Materialeinträge für Branch Key anpassen möchten, die im lokalen Cache gespeichert werden können, müssen Sie bei der Erstellung des hierarchischen Schlüsselbunds keinen Cachetyp angeben. Wenn Sie keinen Cachetyp angeben, verwendet der hierarchische Schlüsselbund den Standard-Cachetyp und legt die Eintragskapazität auf 1000 fest.

Um den Standard-Cache anzupassen, geben Sie die folgenden Werte an:

- Eintragskapazität: Schränkt die Anzahl der Einträge für wichtige Materialien der Branche ein, die im lokalen Cache gespeichert werden können.

Java

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

C# / .NET

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

Rust

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

MultiThreaded Cache

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

Um den MultiThreaded Cache zu verwenden, geben Sie die folgenden Werte an:

- Eintragskapazität: Beschränkt die Anzahl der Einträge für Branch-Schlüsselmaterialien, die im lokalen Cache gespeichert werden können.
- Größe des Endstücks des Eintrags: Definiert die Anzahl der Einträge, die beschnitten werden müssen, wenn die Eingangskapazität erreicht ist.

Java

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

C# / .NET

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

Rust

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

StormTracking Zwischenspeicher

Der StormTracking Cache ist so konzipiert, dass er Umgebungen mit vielen Threads unterstützt. Wenn ein Eintrag für Branch-Schlüssel-Materialien abläuft, verhindert der StormTracking Cache den Aufruf mehrerer Threads, AWS KMS indem ein Thread im Voraus darüber informiert wird, dass der Eintrag für Branch-Schlüssel-Materialien abläuft. Dadurch wird sichergestellt, dass nur ein Thread eine Anfrage AWS KMS zur Aktualisierung des Caches sendet.

Um den StormTracking Cache zu verwenden, geben Sie die folgenden Werte an:

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

Standardwert: 1000 Einträge

- Größe des Eintrags zum Beschneiden: Definiert die Anzahl der Einträge für wichtige Materialien in der Branche, die gleichzeitig beschnitten werden sollen.

  Standardwert: 1 Eintrag

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

  Standardwert: 10 Sekunden

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

  Standardwert: 1 Sekunden

- Fan Out: Definiert die Anzahl der gleichzeitigen Versuche, die Schlüsselmaterialien der Filiale zu aktualisieren.

  Standardwert: 20 Versuche

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

  Standardwert: 10 Sekunden

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

  Standardwert: 20 Millisekunden

Java

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

```
        .inFlightTTL(10)
        .sleepMilli(20)
        .build())
```

C# / .NET

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

Rust

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

Gemeinsam genutzter Cache

Standardmäßig erstellt der hierarchische Schlüsselbund jedes Mal, wenn Sie den Schlüsselbund instanziieren, einen neuen lokalen Cache. Der Shared Cache kann jedoch dabei helfen, Speicherplatz zu sparen, indem er es Ihnen ermöglicht, einen Cache für mehrere hierarchische Schlüsselbunde gemeinsam zu nutzen. Anstatt für jeden hierarchischen Schlüsselbund, den Sie instanziieren, einen neuen Cache für kryptografisches Material zu erstellen, speichert der Shared Cache nur einen Cache im Arbeitsspeicher, der von allen hierarchischen Schlüsselbunden verwendet

werden kann, die auf ihn verweisen. Der gemeinsam genutzte Cache trägt zur Optimierung der Speichernutzung bei, indem verhindert wird, dass kryptografisches Material in mehreren Schlüsselbunden doppelt vorhanden ist. Stattdessen können die hierarchischen Schlüsselbunde auf denselben zugrunde liegenden Cache zugreifen, wodurch der Gesamtspeicherbedarf reduziert wird.

Wenn Sie Ihren Shared Cache erstellen, definieren Sie immer noch den Cachetyp. Sie können einen the section called “Standard-Cache”the section called “MultiThreaded Cache”, oder the section called “StormTracking Zwischenspeicher” als Cachetyp angeben oder einen beliebigen kompatiblen benutzerdefinierten Cache ersetzen.

Partitionen

Ein einziger gemeinsam genutzter Cache kann von mehreren hierarchischen Schlüsselbunden verwendet werden. Wenn Sie einen hierarchischen Schlüsselbund mit einem gemeinsam genutzten Cache erstellen, können Sie eine optionale Partitions-ID definieren. Die Partitions-ID unterscheidet, welcher hierarchische Schlüsselbund in den Cache schreibt. Wenn zwei hierarchische Schlüsselbunde auf dieselbe Partitions-ID und dieselbe Zweigschlüssel-ID verweisenlogical key store name, teilen sich die beiden Schlüsselbunde dieselben Cache-Einträge im Cache. Wenn Sie zwei hierarchische Schlüsselbunde mit demselben Shared Cache, aber unterschiedlicher Partition IDs erstellen, greift jeder Schlüsselbund nur auf die Cache-Einträge von der eigenen zugewiesenen Partition innerhalb des Shared Caches zu. Die Partitionen dienen als logische Unterteilungen innerhalb des gemeinsam genutzten Caches, sodass jeder hierarchische Schlüsselbund unabhängig auf seiner eigenen zugewiesenen Partition betrieben werden kann, ohne die in der anderen Partition gespeicherten Daten zu beeinträchtigen.

Wenn Sie beabsichtigen, die Cache-Einträge in einer Partition wiederzuverwenden oder gemeinsam zu nutzen, müssen Sie Ihre eigene Partitions-ID definieren. Wenn Sie die Partitions-ID an Ihren hierarchischen Schlüsselbund übergeben, kann der Schlüsselbund die Cache-Einträge wiederverwenden, die bereits im Shared Cache vorhanden sind, anstatt die Branch-Schlüsselmaterialien erneut abrufen und autorisieren zu müssen. Wenn Sie keine Partitions-ID angeben, wird dem Schlüsselbund bei jeder Instanziierung des hierarchischen Schlüsselbunds automatisch eine eindeutige Partitions-ID zugewiesen.

Die folgenden Verfahren veranschaulichen, wie ein gemeinsam genutzter Cache mit dem Standard-Cachetyp erstellt und an einen hierarchischen Schlüsselbund übergeben wird.

1. Erstellen Sie einen `CryptographicMaterialsCache` (CMC) mithilfe der Material Providers Library (MPL).

Java

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

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

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

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

C# / .NET

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

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

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

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

Rust

```
// Instantiate the MPL
```

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

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

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

2. Erstellen Sie ein `CacheType` Objekt für den Shared Cache.

   Übergeben `sharedCryptographicMaterialsCache` Sie das, was Sie in Schritt 1 erstellt haben, an das neue `CacheType` Objekt.

   Java

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

   C# / .NET

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

   Rust

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

3. Übergeben Sie das `sharedCache` Objekt aus Schritt 2 an Ihren hierarchischen Schlüsselbund.

   Wenn Sie einen hierarchischen Schlüsselbund mit einem gemeinsam genutzten Cache erstellen, können Sie optional a definieren, um Cache-Einträge für mehrere hierarchische Schlüsselbunde gemeinsam `partitionID` zu nutzen. Wenn Sie keine Partitions-ID angeben, weist der hierarchische Schlüsselbund dem Schlüsselbund automatisch eine eindeutige Partitions-ID zu.

   > **Note**
   >
   > Ihre hierarchischen Schlüsselbunde verwenden dieselben Cacheeinträge in einem gemeinsam genutzten Cache, wenn Sie zwei oder mehr Schlüsselbunde erstellen, die auf dieselbe Partitions-ID und Verzweigungsschlüssel-ID verweisen. logical key store name Wenn Sie nicht möchten, dass sich mehrere Schlüsselbunde dieselben Cache-Einträge teilen, müssen Sie für jeden hierarchischen Schlüsselbund eine eindeutige Partitions-ID verwenden.

   Im folgenden Beispiel wird ein hierarchischer Schlüsselbund mit einem und einem branch key ID supplier Cache-Limit von 600 Sekunden erstellt. Weitere Informationen zu den Werten, die in der folgenden hierarchischen Schlüsselbundkonfiguration definiert sind, finden Sie unter. the section called “Erstellen Sie einen hierarchischen Schlüsselbund”

   Java

   ```
   // Create the Hierarchical keyring
   final CreateAwsKmsHierarchicalKeyringInput keyringInput =
       CreateAwsKmsHierarchicalKeyringInput.builder()
           .keyStore(keystore)
           .branchKeyIdSupplier(branchKeyIdSupplier)
           .ttlSeconds(600)
           .cache(sharedCache)
           .partitionID(partitionID)
           .build();
   final IKeyring hierarchicalKeyring =
    matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);
   ```

   C# / .NET

   ```
   // Create the Hierarchical keyring
   var createKeyringInput = new CreateAwsKmsHierarchicalKeyringInput
   ```

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

Rust

```
// Create the Hierarchical keyring
let keyring1 = mpl
    .create_aws_kms_hierarchical_keyring()
    .key_store(key_store1)
    .branch_key_id(branch_key_id.clone())
    // CryptographicMaterialsCacheRef is an Rc (Reference Counted), so if you
 clone it to
    // pass it to different Hierarchical Keyrings, it will still point to the
 same
    // underlying cache, and increment the reference count accordingly.
    .cache(shared_cache.clone())
    .ttl_seconds(600)
    .partition_id(partition_id.clone())
    .send()
    .await?;
```

Erstellen Sie einen hierarchischen Schlüsselbund

Um einen hierarchischen Schlüsselbund zu erstellen, müssen Sie die folgenden Werte angeben:

- Ein Name für den Schlüsselspeicher

  Der Name der DynamoDB-Tabelle, die Sie oder Ihr Schlüsselspeicheradministrator als Schlüsselspeicher erstellt haben.

-

  Ein Cache-Limit Time to Live (TTL)

Die Zeitspanne in Sekunden, in der ein Eintrag für Branch-Schlüsselmaterialien im lokalen Cache verwendet werden kann, bevor er abläuft. Das Cache-Limit TTL bestimmt, wie oft der Client anruft, AWS KMS um die Verwendung der Branch-Schlüssel zu autorisieren. Dieser Wert muss größer als null sein. Nach Ablauf des Cache-Limits TTL wird der Eintrag nicht mehr bearbeitet und aus dem lokalen Cache entfernt.

- Eine Schlüssel-ID für eine Zweigstelle

  Sie können den entweder statisch konfigurieren`branch-key-id`, der einen einzelnen aktiven Zweigschlüssel in Ihrem Schlüsselspeicher identifiziert, oder einen Lieferanten für die Zweigschlüssel-ID angeben.

  Der Anbieter der Zweigschlüssel-ID bestimmt anhand der im Verschlüsselungskontext gespeicherten Felder, welcher Filialschlüssel zum Entschlüsseln eines Datensatzes erforderlich ist. Standardmäßig sind nur die Partitions- und Sortierschlüssel im Verschlüsselungskontext enthalten. Sie können die `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` kryptografische Aktion jedoch verwenden, um zusätzliche Felder in den Verschlüsselungskontext aufzunehmen.

  Wir empfehlen dringend, für Mehrmandantendatenbanken, bei denen jeder Mandant über einen eigenen Branch-Schlüssel verfügt, einen Branch-Schlüssel-ID-Anbieter zu verwenden. Sie können den Anbieter für die Branch-Schlüssel-ID verwenden, um einen benutzerfreundlichen Namen für Ihren Branch-Schlüssel IDs zu erstellen, damit Sie die richtige Branch-Schlüssel-ID für einen bestimmten Mandanten leicht erkennen können. Mit dem Anzeigenamen können Sie beispielsweise auf einen Zweigschlüssel als `tenant1` statt auf verweisen`b3f61619-4d35-48ad-a275-050f87e15122`.

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

- (Optional) Ein Cache

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

Der hierarchische Schlüsselbund unterstützt die folgenden Cachetypen: Standard, MultiThreaded, StormTracking und Shared. Weitere Informationen und Beispiele zur Definition der einzelnen Cachetypen finden Sie unter. the section called “Wählen Sie einen Cache”

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

- (Optional) Eine Partitions-ID

  Wenn Sie die angebenthe section called “Gemeinsam genutzter Cache”, können Sie optional eine Partitions-ID definieren. Die Partitions-ID unterscheidet, welcher hierarchische Schlüsselbund in den Cache schreibt. Wenn Sie beabsichtigen, die Cache-Einträge in einer Partition wiederzuverwenden oder gemeinsam zu nutzen, müssen Sie Ihre eigene Partitions-ID definieren. Sie können eine beliebige Zeichenfolge für die Partitions-ID angeben. Wenn Sie keine Partitions-ID angeben, wird dem Schlüsselbund bei der Erstellung automatisch eine eindeutige Partitions-ID zugewiesen.

  Weitere Informationen finden Sie unter Partitions.

  > **Note**
  >
  > Ihre hierarchischen Schlüsselbunde verwenden dieselben Cache-Einträge in einem gemeinsam genutzten Cache, wenn Sie zwei oder mehr Schlüsselbunde erstellen, die auf dieselbe Partitions-ID und Verzweigungsschlüssel-ID verweisen. logical key store name Wenn Sie nicht möchten, dass sich mehrere Schlüsselbunde dieselben Cache-Einträge teilen, müssen Sie für jeden hierarchischen Schlüsselbund eine eindeutige Partitions-ID verwenden.

- (Optional) Eine Liste von Grant-Tokens

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

Erstellen Sie einen hierarchischen Schlüsselbund mit einer statischen Zweigschlüssel-ID

Die folgenden Beispiele zeigen, wie Sie einen hierarchischen Schlüsselbund mit einer statischen Zweigschlüssel-ID, derthe section called “Standard-Cache”, und einem Cache-Limit von 600 Sekunden erstellen.

Java

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

C# / .NET

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var keyringInput = new CreateAwsKmsHierarchicalKeyringInput
{
   KeyStore = keystore,
   BranchKeyIdSupplier = branchKeyIdSupplier,
   TtlSeconds = 600
};
var hierarchicalKeyring = matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);
```

Rust

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

let hierarchical_keyring = mpl
    .create_aws_kms_hierarchical_keyring()
    .branch_key_id(branch_key_id)
    .key_store(branch_key_store_name)
    .ttl_seconds(600)
    .send()
    .await?;
```

Erstellen Sie einen hierarchischen Schlüsselbund mit einem Lieferanten für die Zweigschlüssel-ID

Die folgenden Verfahren zeigen, wie Sie einen hierarchischen Schlüsselbund mit einem Branchenschlüssel-ID-Lieferanten erstellen.

1. Erstellen Sie einen Lieferanten für die Zweigschlüssel-ID

   Im folgenden Beispiel werden benutzerfreundliche Namen für die beiden in Schritt 1 erstellten Verzweigungsschlüssel erstellt, und es wird aufgerufen`CreateDynamoDbEncryptionBranchKeyIdSupplier`, mit dem AWS Database Encryption SDK für DynamoDB-Client einen Branch-Schlüssel-ID-Lieferanten zu erstellen.

   Java

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

    public ExampleBranchKeyIdSupplier(String tenant1Id, String tenant2Id) {
        this.branchKeyIdForTenant1 = tenant1Id;
        this.branchKeyIdForTenant2 = tenant2Id;
    }
// Create the branch key ID supplier
final DynamoDbEncryption ddbEnc = DynamoDbEncryption.builder()
        .DynamoDbEncryptionConfig(DynamoDbEncryptionConfig.builder().build())
        .build();
final BranchKeyIdSupplier branchKeyIdSupplier =
 ddbEnc.CreateDynamoDbEncryptionBranchKeyIdSupplier(
    CreateDynamoDbEncryptionBranchKeyIdSupplierInput.builder()
            .ddbKeyBranchKeyIdSupplier(new ExampleBranchKeyIdSupplier(branch-
key-ID-tenant1, branch-key-ID-tenant2))
            .build()).branchKeyIdSupplier();
```

   C# / .NET

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

    public ExampleBranchKeyIdSupplier(String tenant1Id, String tenant2Id) {
```

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

Rust

```
// Create friendly names for each branch_key_id
pub struct ExampleBranchKeyIdSupplier {
    branch_key_id_for_tenant1: String,
    branch_key_id_for_tenant2: String,
}

impl ExampleBranchKeyIdSupplier {
    pub fn new(tenant1_id: &str, tenant2_id: &str) -> Self {
        Self {
            branch_key_id_for_tenant1: tenant1_id.to_string(),
            branch_key_id_for_tenant2: tenant2_id.to_string(),
        }
    }
}

// Create the branch key ID supplier
let dbesdk_config = DynamoDbEncryptionConfig::builder().build()?;
let dbesdk = dbesdk_client::Client::from_conf(dbesdk_config)?;
let supplier = ExampleBranchKeyIdSupplier::new(tenant1_branch_key_id,
 tenant2_branch_key_id);

let branch_key_id_supplier = dbesdk
    .create_dynamo_db_encryption_branch_key_id_supplier()
    .ddb_key_branch_key_id_supplier(supplier)
    .send()
    .await?
    .branch_key_id_supplier
    .unwrap();
```

2. Erstellen Sie einen hierarchischen Schlüsselbund

In den folgenden Beispielen wird ein hierarchischer Schlüsselbund mit dem in Schritt 1 erstellten Branch-Schlüssel-ID-Lieferanten, einem Cache-Limit von 600 Sekunden und einer maximalen Cachegröße von 1000 initialisiert.

Java

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

C# / .NET

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

Rust

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

let hierarchical_keyring = mpl
    .create_aws_kms_hierarchical_keyring()
    .branch_key_id_supplier(branch_key_id_supplier)
    .key_store(key_store)
    .ttl_seconds(600)
    .send()
    .await?;
```

Verwendung des hierarchischen Schlüsselbunds für durchsuchbare Verschlüsselung

Mit der durchsuchbaren Verschlüsselung können Sie verschlüsselte Datensätze durchsuchen, ohne die gesamte Datenbank zu entschlüsseln. Dies wird erreicht, indem der Klartextwert eines verschlüsselten Felds mit einem Beacon indexiert wird. Um eine durchsuchbare Verschlüsselung zu implementieren, müssen Sie einen hierarchischen Schlüsselbund verwenden.

Der *`CreateKey`* Schlüsselspeichervorgang generiert sowohl einen Zweigschlüssel als auch einen Beacon-Schlüssel. Der Zweigschlüssel wird bei der Verschlüsselung und Entschlüsselung von Datensätzen verwendet. Der Beacon-Schlüssel wird zur Generierung von Beacons verwendet.

Der Branch-Schlüssel und der Beacon-Schlüssel sind durch dasselbe geschützt AWS KMS key , das Sie bei der Erstellung Ihres Schlüsselspeicherdienstes angegeben haben. Nachdem der `CreateKey` Vorgang AWS KMS zur Generierung des Branch-Schlüssels aufgerufen hat, ruft er kms: GenerateDataKeyWithoutPlaintext ein zweites Mal auf, um den Beacon-Schlüssel mithilfe der folgenden Anforderung zu generieren.

```
{
   "EncryptionContext": {
      "branch-key-id" : "branch-key-id",
      "type" : type,
      "create-time" : "timestamp",
      "tablename" : "the logical table name for your key store",
      "kms-arn" : the KMS key ARN,
      "hierarchy-version" : 1
```

```
    },
    "KeyId": "the KMS key ARN",
    "NumberOfBytes": "32"
}
```

Nachdem beide Schlüssel generiert wurden, ruft die `CreateKey` Operation ddb: TransactWriteItems auf, um zwei neue Elemente zu schreiben, die den Branch-Schlüssel und den Beacon-Schlüssel in Ihrem Branch-Schlüsselspeicher speichern.

Wenn Sie ein Standard-Beacon konfigurieren, fragt das AWS Database Encryption SDK den Schlüsselspeicher nach dem Beacon-Schlüssel ab. Anschließend verwendet es eine HMAC-basierte extract-and-expand Schlüsselableitungsfunktion (HKDF), um den Beacon-Schlüssel mit dem Namen des Standard-Beacons zu kombinieren, um den HMAC-Schlüssel für einen bestimmten Beacon zu erstellen.

Im Gegensatz zu Zweigschlüsseln gibt es in einem Schlüsselspeicher nur eine Beacon-Schlüsselversion pro Beacon-Schlüssel. `branch-key-id` Der Beacon-Schlüssel wird niemals gedreht.

Definieren Sie Ihre Beacon-Schlüsselquelle

Wenn Sie die Beacon-Version für Ihre Standard- und Verbund-Beacons definieren, müssen Sie den Beacon-Schlüssel identifizieren und ein Cache-Limit für die Gültigkeitsdauer (Time to Live, TTL) für die Beacon-Schlüsselmaterialien definieren. Beacon-Schlüsselmaterialien werden in einem von den Branch-Schlüsseln getrennten lokalen Cache gespeichert. Der folgende Ausschnitt zeigt, wie die `keySource` für eine Single-Tenant-Datenbank definiert wird. Identifizieren Sie Ihren Beacon-Schlüssel anhand dessen, mit dem `branch-key-id` er verknüpft ist.

Java

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

C# / .NET

```
KeySource = new BeaconKeySource
{
```

```
    Single = new SingleKeyStore
    {
       KeyId = branch-key-id,
       CacheTTL = 6000
    }
}
```

Rust

```
 .key_source(BeaconKeySource::Single(
    SingleKeyStore::builder()
        // `keyId` references a beacon key.
        // For every branch key we create in the keystore,
        // we also create a beacon key.
        // This beacon key is not the same as the branch key,
        // but is created with the same ID as the branch key.
        .key_id(branch_key_id)
        .cache_ttl(6000)
        .build()?,
))
```

Definition der Beacon-Quelle in einer mandantenfähigen Datenbank

Wenn Sie über eine Multitenant-Datenbank verfügen, müssen Sie bei der Konfiguration der die folgenden Werte angeben. `keySource`

- keyFieldName

  Definiert den Namen des Felds, in dem der dem Beacon `branch-key-id` zugeordnete Schlüssel gespeichert wird, der zur Generierung von Beacons für einen bestimmten Mandanten verwendet wurde. Dabei `keyFieldName` kann es sich um eine beliebige Zeichenfolge handeln, sie muss jedoch für alle anderen Felder in Ihrer Datenbank eindeutig sein. Wenn Sie neue Datensätze in Ihre Datenbank schreiben, wird der Beacon-Schlüssel`branch-key-id`, der zur Generierung von Beacons für diesen Datensatz verwendet wurde, in diesem Feld gespeichert. Sie müssen dieses Feld in Ihre Beacon-Abfragen aufnehmen und die entsprechenden Beacon-Schlüsselmaterialien identifizieren, die für die Neuberechnung des Beacons erforderlich sind. Weitere Informationen finden Sie unter Abfragen von Beacons in einer mandantenfähigen Datenbank.

- CacheTTL

  Der Zeitraum in Sekunden, in dem ein Eintrag für Beacon-Schlüsselmaterialien im lokalen Beacon-Cache verwendet werden kann, bevor er abläuft. Dieser Wert muss größer als null sein. Wenn das Cache-Limit TTL abläuft, wird der Eintrag aus dem lokalen Cache entfernt.

- (Optional) Ein Cache

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

  Der hierarchische Schlüsselbund unterstützt die folgenden Cachetypen: Standard, MultiThreaded, StormTracking und Shared. Weitere Informationen und Beispiele zur Definition der einzelnen Cachetypen finden Sie unter. the section called “Wählen Sie einen Cache”

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

Im folgenden Beispiel wird ein hierarchischer Schlüsselbund mit einem Branch-Schlüssel-ID-Lieferanten, einem Cache-Limit (TLL) von 600 Sekunden und einer Eingabekapazität von 1000 erstellt.

Java

```
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
final CreateAwsKmsHierarchicalKeyringInput keyringInput =
 CreateAwsKmsHierarchicalKeyringInput.builder()
        .keyStore(branchKeyStoreName)
        .branchKeyIdSupplier(branchKeyIdSupplier)
        .ttlSeconds(600)
        .cache(CacheType.builder() //OPTIONAL
                .Default(DefaultCache.builder()
                        .entryCapacity(1000)
                        .build())
                .build());
final IKeyring hierarchicalKeyring =
 matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);
```

C# / .NET

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var keyringInput = new CreateAwsKmsHierarchicalKeyringInput
{
   KeyStore = keystore,
   BranchKeyIdSupplier = branchKeyIdSupplier,
   TtlSeconds = 600,
   Cache = new CacheType
   {
        Default = new DefaultCache { EntryCapacity = 1000 }
   }
};
var hierarchicalKeyring = matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);
```

Rust

```
let provider_config = MaterialProvidersConfig::builder().build()?;
    let mat_prov = client::Client::from_conf(provider_config)?;
    let kms_keyring = mat_prov
        .create_aws_kms_hierarchical_keyring()
        .branch_key_id(branch_key_id)
        .key_store(key_store)
        .ttl_seconds(600)
        .send()
        .await?;
```

AWS KMS ECDH-Schlüsselanhänger

Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt . AWS Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Important

Der AWS KMS ECDH-Schlüsselbund ist nur mit Version 1.5.0 oder höher der Material Providers Library verfügbar.

Ein AWS KMS ECDH-Schlüsselbund verwendet eine asymmetrische Schlüsselvereinbarung, AWS KMS keysum einen gemeinsamen symmetrischen Wrapping-Schlüssel zwischen zwei Parteien abzuleiten. Zunächst verwendet der Schlüsselbund den Schlüsselvereinbarungsalgorithmus Elliptic Curve Diffie-Hellman (ECDH), um ein gemeinsames Geheimnis aus dem privaten Schlüssel im KMS-Schlüsselpaar des Absenders und dem öffentlichen Schlüssel des Empfängers abzuleiten. Anschließend leitet der Schlüsselbund anhand des gemeinsamen geheimen Schlüssels den gemeinsamen Wrapping-Schlüssel ab, der Ihre Datenverschlüsselungsschlüssel schützt. Die Schlüsselableitungsfunktion, die das AWS Database Encryption SDK (`KDF_CTR_HMAC_SHA384`) verwendet, um den gemeinsamen Wrapping-Schlüssel abzuleiten, entspricht den NIST-Empfehlungen für die Schlüsselableitung.

Die Funktion zur Schlüsselableitung gibt 64 Byte an Schlüsselmaterial zurück. Um sicherzustellen, dass beide Parteien das richtige Schlüsselmaterial verwenden, verwendet das AWS Database Encryption SDK die ersten 32 Byte als Commitment-Schlüssel und die letzten 32 Byte als gemeinsamen Wrapping-Schlüssel. Wenn der Schlüsselbund beim Entschlüsseln nicht denselben Commitment-Schlüssel und denselben gemeinsamen Wrapping-Schlüssel reproduzieren kann, die im Materialbeschreibungsfeld des verschlüsselten Datensatzes gespeichert sind, schlägt der Vorgang fehl. Wenn Sie beispielsweise einen Datensatz mit einem Schlüsselbund verschlüsseln, der mit Alices privatem Schlüssel und Bobs öffentlichem Schlüssel konfiguriert ist, reproduziert ein Schlüsselbund, der mit Bobs privatem Schlüssel und Alices öffentlichem Schlüssel konfiguriert ist, denselben Commitment-Schlüssel und gemeinsamen Wrapping-Schlüssel und kann den Datensatz entschlüsseln. Wenn Bobs öffentlicher Schlüssel nicht von einem KMS-Schlüsselpaar stammt, kann Bob einen Raw ECDH-Schlüsselbund erstellen, um den Datensatz zu entschlüsseln.

Der AWS KMS ECDH-Schlüsselbund verschlüsselt Datensätze mit einem symmetrischen Schlüssel unter Verwendung von AES-GCM. Der Datenschlüssel wird dann mit dem abgeleiteten gemeinsamen Wrapping-Schlüssel unter Verwendung von AES-GCM umhüllt. Jeder AWS KMS ECDH-Schlüsselbund kann nur einen gemeinsamen Wrapping-Schlüssel haben, aber Sie können mehrere AWS KMS ECDH-Schlüsselanhänger, einzeln oder zusammen mit anderen Schlüsselbunden, in einen Mehrfachschlüsselbund aufnehmen.

Themen

- AWS KMS Erforderliche Berechtigungen für ECDH-Schlüsselanhänger
- Einen ECDH-Schlüsselbund AWS KMS erstellen
- Einen AWS KMS ECDH-Discovery-Schlüsselbund erstellen

AWS KMS Erforderliche Berechtigungen für ECDH-Schlüsselanhänger

Für das AWS Database Encryption SDK ist kein AWS Konto erforderlich und es ist von keinem AWS Dienst abhängig. Um einen AWS KMS ECDH-Schlüsselbund verwenden zu können, benötigen Sie jedoch ein AWS Konto und die folgenden Mindestberechtigungen für AWS KMS keys den Schlüsselbund. Die Berechtigungen variieren je nachdem, welches Schlüsselvereinbarungsschema Sie verwenden.

- Um Datensätze mithilfe des `KmsPrivateKeyToStaticPublicKey` Schlüsselvereinbarungsschemas zu verschlüsseln und zu entschlüsseln, benötigen Sie kms: GetPublicKey und kms: DeriveSharedSecret auf dem asymmetrischen KMS-Schlüsselpaar des Absenders. Wenn Sie den DER-codierten öffentlichen Schlüssel des Absenders direkt angeben, wenn Sie Ihren Schlüsselbund instanziieren, benötigen Sie nur die kms: DeriveSharedSecret - Berechtigung für das asymmetrische KMS-Schlüsselpaar des Absenders.
- Um Datensätze mithilfe des `KmsPublicKeyDiscovery` Schlüsselvereinbarungsschemas zu entschlüsseln, benötigen Sie die GetPublicKey Berechtigungen kms: DeriveSharedSecret und kms: für das angegebene asymmetrische KMS-Schlüsselpaar.

Einen ECDH-Schlüsselbund AWS KMS erstellen

Um einen AWS KMS ECDH-Schlüsselbund zu erstellen, der Daten ver- und entschlüsselt, müssen Sie das Schlüsselvereinbarungsschema verwenden. `KmsPrivateKeyToStaticPublicKey` Um einen AWS KMS ECDH-Schlüsselbund mit dem Schlüsselvereinbarungsschema zu initialisieren, geben Sie die folgenden `KmsPrivateKeyToStaticPublicKey` Werte an:

- ID des Absenders AWS KMS key

  Muss ein von NIST empfohlenes asymmetrisches KMS-Schlüsselpaar mit elliptischer Kurve (ECC) mit einem Wert von identifizieren. `KeyUsage` `KEY_AGREEMENT` Der private Schlüssel des Absenders wird verwendet, um den gemeinsamen geheimen Schlüssel abzuleiten.

- (Optional) Der öffentliche Schlüssel des Absenders

  Muss ein DER-codierter öffentlicher X.509-Schlüssel sein, auch bekannt als `SubjectPublicKeyInfo` (SPKI), wie in RFC 5280 definiert.

  Die AWS KMS GetPublicKeyOperation gibt den öffentlichen Schlüssel eines asymmetrischen KMS-Schlüsselpaars im erforderlichen DER-codierten Format zurück.

Um die Anzahl der AWS KMS Anrufe zu reduzieren, die Ihr Schlüsselbund tätigt, können Sie den öffentlichen Schlüssel des Absenders direkt angeben. Wenn kein Wert für den öffentlichen Schlüssel des Absenders angegeben wird, ruft der Schlüsselbund auf, AWS KMS um den öffentlichen Schlüssel des Absenders abzurufen.

- Der öffentliche Schlüssel des Empfängers

  Sie müssen den DER-codierten öffentlichen X.509-Schlüssel des Empfängers, auch bekannt als `SubjectPublicKeyInfo` (SPKI), wie in RFC 5280 definiert, angeben.

  Die AWS KMS GetPublicKeyOperation gibt den öffentlichen Schlüssel eines asymmetrischen KMS-Schlüsselpaars im erforderlichen DER-codierten Format zurück.

- Kurvenspezifikation

  Identifiziert die Spezifikation für elliptische Kurven in den angegebenen Schlüsselpaaren. Sowohl die Schlüsselpaare des Absenders als auch des Empfängers müssen dieselbe Kurvenspezifikation haben.

  Zulässige Werte: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

- (Optional) Eine Liste von Grant-Tokens

  Wenn Sie den Zugriff auf den KMS-Schlüssel in Ihrem AWS KMS ECDH-Schlüsselbund mit Grants steuern, müssen Sie bei der Initialisierung des Schlüsselbunds alle erforderlichen Grant-Token angeben.

C# / .NET

Im folgenden Beispiel wird ein AWS KMS ECDH-Schlüsselbund mit dem KMS-Schlüssel des Absenders, dem öffentlichen Schlüssel des Absenders und dem öffentlichen Schlüssel des Empfängers erstellt. In diesem Beispiel wird der optionale `senderPublicKey` Parameter verwendet, um den öffentlichen Schlüssel des Absenders bereitzustellen. Wenn Sie den öffentlichen Schlüssel des Absenders nicht angeben, ruft der Schlüsselbund auf, AWS KMS um den öffentlichen Schlüssel des Absenders abzurufen. Sowohl die Schlüsselpaare des Absenders als auch des Empfängers befinden sich auf der `ECC_NIST_P256` Kurve.

```
// Instantiate material providers
var materialProviders = new MaterialProviders(new MaterialProvidersConfig());

// Must be DER-encoded X.509 public keys
```

```
var BobPublicKey = new MemoryStream(new byte[] { });
var AlicePublicKey = new MemoryStream(new byte[] { });

// Create the AWS KMS ECDH static keyring
var staticConfiguration = new KmsEcdhStaticConfigurations
{
    KmsPrivateKeyToStaticPublicKey = new KmsPrivateKeyToStaticPublicKeyInput
    {
        SenderKmsIdentifier = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        SenderPublicKey = BobPublicKey,
        RecipientPublicKey = AlicePublicKey
    }
};

var createKeyringInput = new CreateAwsKmsEcdhKeyringInput
{
    CurveSpec = ECDHCurveSpec.ECC_NIST_P256,
    KmsClient = new AmazonKeyManagementServiceClient(),
    KeyAgreementScheme = staticConfiguration
};

var keyring = materialProviders.CreateAwsKmsEcdhKeyring(createKeyringInput);
```

Java

Im folgenden Beispiel wird ein AWS KMS ECDH-Schlüsselbund mit dem KMS-Schlüssel des Absenders, dem öffentlichen Schlüssel des Absenders und dem öffentlichen Schlüssel des Empfängers erstellt. In diesem Beispiel wird der optionale `senderPublicKey` Parameter verwendet, um den öffentlichen Schlüssel des Absenders bereitzustellen. Wenn Sie den öffentlichen Schlüssel des Absenders nicht angeben, ruft der Schlüsselbund auf, AWS KMS um den öffentlichen Schlüssel des Absenders abzurufen. Sowohl die Schlüsselpaare des Absenders als auch des Empfängers befinden sich auf der `ECC_NIST_P256` Kurve.

```
// Retrieve public keys
// Must be DER-encoded X.509 public keys
ByteBuffer BobPublicKey = getPublicKeyBytes("arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab");
        ByteBuffer AlicePublicKey = getPublicKeyBytes("arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321");

// Create the AWS KMS ECDH static keyring
```

```
        final CreateAwsKmsEcdhKeyringInput senderKeyringInput =
          CreateAwsKmsEcdhKeyringInput.builder()
            .kmsClient(KmsClient.create())
            .curveSpec(ECDHCurveSpec.ECC_NIST_P256)
            .KeyAgreementScheme(
              KmsEcdhStaticConfigurations.builder()
                .KmsPrivateKeyToStaticPublicKey(
                  KmsPrivateKeyToStaticPublicKeyInput.builder()
                    .senderKmsIdentifier("arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab")
                    .senderPublicKey(BobPublicKey)
                    .recipientPublicKey(AlicePublicKey)
                    .build()).build()).build();
```

Rust

Im folgenden Beispiel wird ein AWS KMS ECDH-Schlüsselbund mit dem KMS-Schlüssel des Absenders, dem öffentlichen Schlüssel des Absenders und dem öffentlichen Schlüssel des Empfängers erstellt. In diesem Beispiel wird der optionale `sender_public_key` Parameter verwendet, um den öffentlichen Schlüssel des Absenders bereitzustellen. Wenn Sie den öffentlichen Schlüssel des Absenders nicht angeben, ruft der Schlüsselbund auf, AWS KMS um den öffentlichen Schlüssel des Absenders abzurufen.

```
// Retrieve public keys
// Must be DER-encoded X.509 keys
let public_key_file_content_sender =
 std::fs::read_to_string(Path::new(EXAMPLE_KMS_ECC_PUBLIC_KEY_FILENAME_SENDER))?;
let parsed_public_key_file_content_sender = parse(public_key_file_content_sender)?;
let public_key_sender_utf8_bytes = parsed_public_key_file_content_sender.contents();

let public_key_file_content_recipient =
 std::fs::read_to_string(Path::new(EXAMPLE_KMS_ECC_PUBLIC_KEY_FILENAME_RECIPIENT))?;
let parsed_public_key_file_content_recipient =
 parse(public_key_file_content_recipient)?;
let public_key_recipient_utf8_bytes =
 parsed_public_key_file_content_recipient.contents();

// Create KmsPrivateKeyToStaticPublicKeyInput
let kms_ecdh_static_configuration_input =
    KmsPrivateKeyToStaticPublicKeyInput::builder()
        .sender_kms_identifier(arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab)
        // Must be a UTF8 DER-encoded X.509 public key
```

```
        .sender_public_key(public_key_sender_utf8_bytes)
        // Must be a UTF8 DER-encoded X.509 public key
        .recipient_public_key(public_key_recipient_utf8_bytes)
        .build()?;

let kms_ecdh_static_configuration =
 KmsEcdhStaticConfigurations::KmsPrivateKeyToStaticPublicKey(kms_ecdh_static_configuration_i

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

// Create AWS KMS ECDH keyring
let kms_ecdh_keyring = mpl
    .create_aws_kms_ecdh_keyring()
    .kms_client(kms_client)
    .curve_spec(ecdh_curve_spec)
    .key_agreement_scheme(kms_ecdh_static_configuration)
    .send()
    .await?;
```

Einen AWS KMS ECDH-Discovery-Schlüsselbund erstellen

Beim Entschlüsseln empfiehlt es sich, die Schlüssel anzugeben, die das AWS Database Encryption SDK verwenden kann. Um dieser bewährten Methode zu folgen, verwenden Sie einen AWS KMS ECDH-Schlüsselbund mit dem `KmsPrivateKeyToStaticPublicKey` Schlüsselvereinbarungsschema. Sie können jedoch auch einen AWS KMS ECDH-Discovery-Schlüsselbund erstellen, d. h. einen AWS KMS ECDH-Schlüsselbund, der jeden Datensatz entschlüsseln kann, bei dem der öffentliche Schlüssel des angegebenen KMS-Schlüsselpaars mit dem öffentlichen Schlüssel des Empfängers übereinstimmt, der im Materialbeschreibungsfeld des verschlüsselten Datensatzes gespeichert ist.

> **Important**
>
> Wenn Sie Datensätze mithilfe des `KmsPublicKeyDiscovery` Schlüsselvereinbarungsschemas entschlüsseln, akzeptieren Sie alle öffentlichen Schlüssel, unabhängig davon, wem sie gehören.

Um einen AWS KMS ECDH-Schlüsselbund mit dem Schlüsselvereinbarungsschema zu initialisieren, geben Sie die `KmsPublicKeyDiscovery` folgenden Werte an:

- ID des Empfängers AWS KMS key

  Muss ein von NIST empfohlenes asymmetrisches KMS-Schlüsselpaar mit elliptischer Kurve (ECC) mit einem Wert von identifizieren. `KeyUsage` `KEY_AGREEMENT`
- Spezifikation der Kurve

  Identifiziert die elliptische Kurvenspezifikation im KMS-Schlüsselpaar des Empfängers.

  Zulässige Werte: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`
- (Optional) Eine Liste von Grant-Tokens

  Wenn Sie den Zugriff auf den KMS-Schlüssel in Ihrem AWS KMS ECDH-Schlüsselbund mit Grants steuern, müssen Sie bei der Initialisierung des Schlüsselbunds alle erforderlichen Grant-Token angeben.

C# / .NET

Im folgenden Beispiel wird ein AWS KMS ECDH-Discovery-Schlüsselbund mit einem KMS-Schlüsselpaar auf der `ECC_NIST_P256` Kurve erstellt. Sie müssen über die DeriveSharedSecret Berechtigungen kms: GetPublicKey und kms: für das angegebene KMS-Schlüsselpaar verfügen. Dieser Schlüsselbund kann jeden Datensatz entschlüsseln, bei dem der öffentliche Schlüssel des angegebenen KMS-Schlüsselpaars mit dem öffentlichen Schlüssel des Empfängers übereinstimmt, der im Materialbeschreibungsfeld des verschlüsselten Datensatzes gespeichert ist.

```
// Instantiate material providers
var materialProviders = new MaterialProviders(new MaterialProvidersConfig());

// Create the AWS KMS ECDH discovery keyring
var discoveryConfiguration = new KmsEcdhStaticConfigurations
{
    KmsPublicKeyDiscovery = new KmsPublicKeyDiscoveryInput
    {
        RecipientKmsIdentifier = "arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321"
    }

};
```

```
var createKeyringInput = new CreateAwsKmsEcdhKeyringInput
{
    CurveSpec = ECDHCurveSpec.ECC_NIST_P256,
    KmsClient = new AmazonKeyManagementServiceClient(),
    KeyAgreementScheme = discoveryConfiguration
};
var keyring = materialProviders.CreateAwsKmsEcdhKeyring(createKeyringInput);
```

Java

Im folgenden Beispiel wird ein AWS KMS ECDH-Discovery-Schlüsselbund mit einem KMS-Schlüsselpaar auf der ECC_NIST_P256 Kurve erstellt. Sie müssen über die DeriveSharedSecret Berechtigungen kms: GetPublicKey und kms: für das angegebene KMS-Schlüsselpaar verfügen. Dieser Schlüsselbund kann jeden Datensatz entschlüsseln, bei dem der öffentliche Schlüssel des angegebenen KMS-Schlüsselpaars mit dem öffentlichen Schlüssel des Empfängers übereinstimmt, der im Materialbeschreibungsfeld des verschlüsselten Datensatzes gespeichert ist.

```
// Create the AWS KMS ECDH discovery keyring
final CreateAwsKmsEcdhKeyringInput recipientKeyringInput =
  CreateAwsKmsEcdhKeyringInput.builder()
    .kmsClient(KmsClient.create())
    .curveSpec(ECDHCurveSpec.ECC_NIST_P256)
    .KeyAgreementScheme(
      KmsEcdhStaticConfigurations.builder()
        .KmsPublicKeyDiscovery(
          KmsPublicKeyDiscoveryInput.builder()
            .recipientKmsIdentifier("arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321").build()
        ).build())
    .build();
```

Rust

```
// Create KmsPublicKeyDiscoveryInput
let kms_ecdh_discovery_static_configuration_input =
    KmsPublicKeyDiscoveryInput::builder()
        .recipient_kms_identifier(ecc_recipient_key_arn)
        .build()?;

let kms_ecdh_discovery_static_configuration =
 KmsEcdhStaticConfigurations::KmsPublicKeyDiscovery(kms_ecdh_discovery_static_configuration_
```

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

// Create AWS KMS ECDH discovery keyring
let kms_ecdh_discovery_keyring = mpl
    .create_aws_kms_ecdh_keyring()
    .kms_client(kms_client.clone())
    .curve_spec(ecdh_curve_spec)
    .key_agreement_scheme(kms_ecdh_discovery_static_configuration)
    .send()
    .await?;
```

Unformatierte AES-Schlüsselbunde

Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Mit dem AWS Database Encryption SDK können Sie einen symmetrischen AES-Schlüssel verwenden, den Sie als Wrapping-Schlüssel angeben, der Ihren Datenschlüssel schützt. Sie müssen das Schlüsselmaterial generieren, speichern und schützen, vorzugsweise in einem Hardware-Sicherheitsmodul (HSM) oder einem Schlüsselverwaltungssystem. Verwenden Sie einen RAW-AES-Schlüsselbund, wenn Sie den Wrap-Schlüssel bereitstellen und die Datenschlüssel lokal oder offline verschlüsseln müssen.

Der Raw AES-Schlüsselbund verschlüsselt Daten mithilfe des AES-GCM-Algorithmus und eines Wrapping-Schlüssels, den Sie als Byte-Array angeben. Sie können in jedem Raw-AES-Schlüsselbund nur einen Wrap-Schlüssel angeben, aber Sie können mehrere Raw AES-Schlüsselanhänger, allein oder zusammen mit anderen Schlüsselbunden, in einen Mehrfachschlüsselbund aufnehmen.

Wichtige Namespaces und Namen

Um den AES-Schlüssel in einem Schlüsselbund zu identifizieren, verwendet der Raw AES-Schlüsselbund einen Schlüsselnamespace und einen Schlüsselnamen, die Sie angeben. Diese Werte sind nicht geheim. Sie erscheinen im Klartext in der Materialbeschreibung, die das AWS Database Encryption SDK dem Datensatz hinzufügt. Wir empfehlen, für Ihr HSM- oder

Schlüsselverwaltungssystem einen Schlüsselnamespace und einen Schlüsselnamen zu verwenden, der den AES-Schlüssel in diesem System identifiziert.

> **Note**
>
> Der Schlüsselnamespace und der Schlüsselname entsprechen den Feldern Provider-ID (oder Provider) und Key-ID in. `JceMasterKey`

Wenn Sie verschiedene Schlüsselbunde zum Verschlüsseln und Entschlüsseln eines bestimmten Felds erstellen, sind die Namespace- und Namenswerte entscheidend. Wenn der Schlüsselnamespace und der Schlüsselname im Schlüsselbund für die Entschlüsselung nicht exakt und unter Berücksichtigung der Groß- und Kleinschreibung mit dem Schlüsselnamespace und dem Schlüsselnamen im Verschlüsselungsschlüsselbund übereinstimmen, wird der Schlüsselbund nicht verwendet, auch wenn die Schlüsselmaterial-Bytes identisch sind.

Sie könnten beispielsweise einen RAW-AES-Schlüsselbund mit Schlüsselnamespace und Schlüsselname definieren. `HSM_01` `AES_256_012` Anschließend verwenden Sie diesen Schlüsselbund, um einige Daten zu verschlüsseln. Um diese Daten zu entschlüsseln, erstellen Sie einen RAW-AES-Schlüsselbund mit demselben Schlüsselnamespace, demselben Schlüsselnamen und demselben Schlüsselmaterial.

Die folgenden Beispiele zeigen, wie Sie einen Raw AES-Schlüsselbund erstellen. Die `AESWrappingKey` Variable steht für das von Ihnen bereitgestellte Schlüsselmaterial.

Java

```
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);
```

C# / .NET

```
var keyNamespace = "HSM_01";
var keyName = "AES_256_012";

// This example uses the key generator in Bouncy Castle to generate the key
 material.
// In production, use key material from a secure source.
var aesWrappingKey = new
 MemoryStream(GeneratorUtilities.GetKeyGenerator("AES256").GenerateKey());

// Create the keyring
var keyringInput = new CreateRawAesKeyringInput
{
    KeyNamespace = keyNamespace,
    KeyName = keyName,
    WrappingKey = AESWrappingKey,
    WrappingAlg = AesWrappingAlg.ALG_AES256_GCM_IV12_TAG16
};

var matProv = new MaterialProviders(new MaterialProvidersConfig());
IKeyring rawAesKeyring = matProv.CreateRawAesKeyring(keyringInput);
```

Rust

```
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;
let raw_aes_keyring = mpl
    .create_raw_aes_keyring()
    .key_name("AES_256_012")
    .key_namespace("HSM_01")
    .wrapping_key(aes_key_bytes)
    .wrapping_alg(AesWrappingAlg::AlgAes256GcmIv12Tag16)
    .send()
    .await?;
```

Unformatierte RSA-Schlüsselbunde

Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Der Raw RSA Keyring führt eine asymmetrische Verschlüsselung und Entschlüsselung von Datenschlüsseln im lokalen Speicher mit den von Ihnen bereitgestellten öffentlichen und privaten RSA-Schlüsseln durch. Sie müssen den privaten Schlüssel generieren, speichern und schützen, vorzugsweise in einem Hardware-Sicherheitsmodul (HSM) oder einem Schlüsselverwaltungssystem. Die Verschlüsselungsfunktion verschlüsselt den Datenschlüssel unter dem öffentlichen RSA-Schlüssel. Die Entschlüsselungsfunktion entschlüsselt den Datenschlüssel mithilfe des privaten Schlüssels. Sie können aus mehreren RSA-Padding-Modi auswählen.

Ein unformatierter RSA-Schlüsselbund, der verschlüsselt und entschlüsselt, muss ein asymmetrisches öffentliches und privates Schlüsselpaar enthalten. Sie können Daten jedoch mit einem Raw RSA Keyring verschlüsseln, der nur über einen öffentlichen Schlüssel verfügt, und Sie können Daten mit einem Raw RSA Schlüsselbund entschlüsseln, der nur über einen privaten Schlüssel verfügt. Sie können einen beliebigen Raw RSA-Schlüsselbund in einen Mehrfachschlüsselbund aufnehmen. Wenn Sie einen Raw RSA-Schlüsselbund mit einem öffentlichen und einem privaten Schlüssel konfigurieren, stellen Sie sicher, dass sie Teil desselben key pair sind.

Der Raw RSA-Schlüsselbund entspricht den JceMasterKeyin und arbeitet mit ihnen zusammen, AWS-Verschlüsselungs-SDK for Java wenn sie mit asymmetrischen RSA-Verschlüsselungsschlüsseln verwendet werden.

> **Note**
>
> Der Raw RSA-Schlüsselbund unterstützt keine asymmetrischen KMS-Schlüssel. Um asymmetrische RSA-KMS-Schlüssel zu verwenden, erstellen Sie einen Schlüsselbund.AWS KMS

Namespaces und Namen

Um das RSA-Schlüsselmaterial in einem Schlüsselbund zu identifizieren, verwendet der RSA-RSA-Schlüsselbund einen Schlüsselnamespace und einen Schlüsselnamen, die Sie angeben. Diese

Werte sind nicht geheim. Sie erscheinen im Klartext in der Materialbeschreibung, die das AWS Database Encryption SDK dem Datensatz hinzufügt. Wir empfehlen, den Schlüsselnamespace und den Schlüsselnamen zu verwenden, die das RSA-Schlüsselpaar (oder seinen privaten Schlüssel) in Ihrem HSM oder Schlüsselverwaltungssystem identifizieren.

> **Note**
>
> Der Schlüsselnamespace und der Schlüsselname entsprechen den Feldern Provider-ID (oder Provider) und Key-ID in. `JceMasterKey`

Wenn Sie verschiedene Schlüsselbunde zum Verschlüsseln und Entschlüsseln eines bestimmten Datensatzes erstellen, sind die Namespace- und Namenswerte entscheidend. Wenn der Schlüsselnamespace und der Schlüsselname im Entschlüsselungsschlüsselbund nicht exakt und unter Berücksichtigung der Groß- und Kleinschreibung für den Schlüsselnamespace und den Schlüsselnamen im Verschlüsselungsschlüsselbund übereinstimmen, wird der Entschlüsselungsschlüsselbund nicht verwendet, auch wenn die Schlüssel aus demselben key pair stammen.

Der Schlüsselnamespace und der Schlüsselname des Schlüsselmaterials in den Verschlüsselungs- und Entschlüsselungsschlüsselbunden müssen identisch sein, unabhängig davon, ob der Schlüsselbund den öffentlichen RSA-Schlüssel, den privaten RSA-Schlüssel oder beide Schlüssel im key pair enthält. Nehmen wir beispielsweise an, Sie verschlüsseln Daten mit einem RSA-Rohschlüsselbund für einen öffentlichen RSA-Schlüssel mit Schlüsselnamespace und Schlüsselname. `HSM_01` `RSA_2048_06` Um diese Daten zu entschlüsseln, erstellen Sie einen RSA-Rohschlüsselbund mit dem privaten Schlüssel (oder key pair) und demselben Schlüsselnamespace und Namen.

Padding-Modus

Sie müssen einen Füllmodus für RSA-Rohschlüsselringe angeben, die für die Verschlüsselung und Entschlüsselung verwendet werden, oder Funktionen Ihrer Sprachimplementierung verwenden, die ihn für Sie spezifizieren.

Der AWS Encryption SDK unterstützt die folgenden Füllmodi, die den Einschränkungen der jeweiligen Sprache unterliegen. Wir empfehlen einen OAEP-Padding-Modus, insbesondere OAEP mit SHA-256 und mit SHA-256 Padding. MGF1 Der Padding-Modus wird nur aus Gründen der Abwärtskompatibilität unterstützt. PKCS1

- OAEP mit SHA-1 und mit SHA-1 Padding MGF1
- OAEP mit SHA-256 und mit SHA-256-Padding MGF1
- OAEP mit SHA-384 und mit SHA-384-Padding MGF1
- OAEP mit SHA-512 und mit SHA-512-Padding MGF1
- PKCS1 v1.5 Polsterung

Das folgende Java-Beispiel zeigt, wie ein Raw RSA-Schlüsselbund mit dem öffentlichen und privaten Schlüssel eines RSA-Schlüsselpaars und dem OAEP mit SHA-256 und dem SHA-256-Padding-Modus erstellt wird. MGF1 Die Variablen und stellen das von Ihnen bereitgestellte Schlüsselmaterial dar. `RSAPublicKey` `RSAPrivateKey`

Java

```
final CreateRawRsaKeyringInput keyringInput = CreateRawRsaKeyringInput.builder()
        .keyName("RSA_2048_06")
        .keyNamespace("HSM_01")
        .paddingScheme(PaddingScheme.OAEP_SHA256_MGF1)
        .publicKey(RSAPublicKey)
        .privateKey(RSAPrivateKey)
        .build();
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
IKeyring rawRsaKeyring = matProv.CreateRawRsaKeyring(keyringInput);
```

C# / .NET

```
var keyNamespace = "HSM_01";
var keyName = "RSA_2048_06";

// Get public and private keys from PEM files
var publicKey = new
 MemoryStream(System.IO.File.ReadAllBytes("RSAKeyringExamplePublicKey.pem"));
var privateKey = new
 MemoryStream(System.IO.File.ReadAllBytes("RSAKeyringExamplePrivateKey.pem"));

// Create the keyring input
var keyringInput = new CreateRawRsaKeyringInput
{
    KeyNamespace = keyNamespace,
```

```
    KeyName = keyName,
    PaddingScheme = PaddingScheme.OAEP_SHA512_MGF1,
    PublicKey = publicKey,
    PrivateKey = privateKey
};

// Create the keyring
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var rawRsaKeyring = matProv.CreateRawRsaKeyring(keyringInput);
```

Rust

```
let mpl_config = MaterialProvidersConfig::builder().build()?;
let mpl = mpl_client::Client::from_conf(mpl_config)?;
let raw_rsa_keyring = mpl
    .create_raw_rsa_keyring()
    .key_name("RSA_2048_06")
    .key_namespace("HSM_01")
    .padding_scheme(PaddingScheme::OaepSha256Mgf1)
    .public_key(RSA_public_key)
    .private_key(RSA_private_key)
    .send()
    .await?;
```

Raw ECDH Schlüsselanhänger

Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt . AWS Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Important

Der Raw ECDH-Schlüsselbund ist nur mit Version 1.5.0 der Material Providers Library verfügbar.

Der Raw ECDH-Schlüsselbund verwendet die öffentlich-privaten Schlüsselpaare mit elliptischer Kurve, die Sie angeben, um einen gemeinsamen Wrapping-Schlüssel zwischen zwei Parteien

abzuleiten. Zunächst leitet der Schlüsselbund mithilfe des privaten Schlüssels des Absenders, des öffentlichen Schlüssels des Empfängers und des Schlüsselvereinbarungsalgorithmus Elliptic Curve Diffie-Hellman (ECDH) ein gemeinsames Geheimnis ab. Anschließend leitet der Schlüsselbund anhand des gemeinsamen geheimen Schlüssels den gemeinsamen Wrapping-Schlüssel ab, der Ihre Datenverschlüsselungsschlüssel schützt. Die Schlüsselableitungsfunktion, mit der das AWS Database Encryption SDK (`KDF_CTR_HMAC_SHA384`) den gemeinsamen Wrapping-Schlüssel ableitet, entspricht den NIST-Empfehlungen für die Schlüsselableitung.

Die Funktion zur Schlüsselableitung gibt 64 Byte an Schlüsselmaterial zurück. Um sicherzustellen, dass beide Parteien das richtige Schlüsselmaterial verwenden, verwendet das AWS Database Encryption SDK die ersten 32 Byte als Commitment-Schlüssel und die letzten 32 Byte als gemeinsamen Wrapping-Schlüssel. Wenn der Schlüsselbund beim Entschlüsseln nicht denselben Commitment-Schlüssel und denselben gemeinsamen Wrapping-Schlüssel reproduzieren kann, die im Materialbeschreibungsfeld des verschlüsselten Datensatzes gespeichert sind, schlägt der Vorgang fehl. Wenn Sie beispielsweise einen Datensatz mit einem Schlüsselbund verschlüsseln, der mit Alices privatem Schlüssel und Bobs öffentlichem Schlüssel konfiguriert ist, reproduziert ein Schlüsselbund, der mit Bobs privatem Schlüssel und Alices öffentlichem Schlüssel konfiguriert ist, denselben Commitment-Schlüssel und gemeinsamen Wrapping-Schlüssel und kann den Datensatz entschlüsseln. Wenn Bobs öffentlicher Schlüssel aus einem AWS KMS key Paar stammt, kann Bob einen AWS KMS ECDH-Schlüsselbund erstellen, um den Datensatz zu entschlüsseln.

Der Raw ECDH-Schlüsselbund verschlüsselt Datensätze mit einem symmetrischen Schlüssel mithilfe von AES-GCM. Der Datenschlüssel wird dann mit dem abgeleiteten gemeinsamen Wrapping-Schlüssel unter Verwendung von AES-GCM umhüllt. Jeder Raw ECDH-Schlüsselbund kann nur einen gemeinsamen Wrap-Schlüssel haben, aber Sie können mehrere Raw ECDH-Schlüsselanhänger, einzeln oder zusammen mit anderen Schlüsselbunden, in einen Mehrfachschlüsselbund aufnehmen.

Sie sind dafür verantwortlich, Ihre privaten Schlüssel zu generieren, zu speichern und zu schützen, vorzugsweise in einem Hardware-Sicherheitsmodul (HSM) oder einem Schlüsselverwaltungssystem. Die Schlüsselpaare des Absenders und des Empfängers müssen sich auf derselben elliptischen Kurve befinden. Das AWS Database Encryption SDK unterstützt die folgenden Spezifikationen für elliptische Kurven:

- `ECC_NIST_P256`
- `ECC_NIST_P384`
- `ECC_NIST_P512`

Einen RAW-ECDH-Schlüsselbund erstellen

Der Raw ECDH-Schlüsselbund unterstützt drei wichtige Vereinbarungsschemata:, und. `RawPrivateKeyToStaticPublicKey` `EphemeralPrivateKeyToStaticPublicKey` `PublicKeyDiscovery` Das von Ihnen gewählte Schlüsselvereinbarungsschema bestimmt, welche kryptografischen Operationen Sie ausführen können und wie die Schlüsselmaterialien zusammengestellt werden.

Themen

- RawPrivateKeyToStaticPublicKey
- EphemeralPrivateKeyToStaticPublicKey
- PublicKeyDiscovery

RawPrivateKeyToStaticPublicKey

Verwenden Sie das `RawPrivateKeyToStaticPublicKey` Schlüsselvereinbarungsschema, um den privaten Schlüssel des Absenders und den öffentlichen Schlüssel des Empfängers im Schlüsselbund statisch zu konfigurieren. Dieses Schlüsselvereinbarungsschema kann Datensätze ver- und entschlüsseln.

Um einen RAW-ECDH-Schlüsselbund mit dem Schlüsselvereinbarungsschema zu initialisieren, `RawPrivateKeyToStaticPublicKey` geben Sie die folgenden Werte an:

- Der private Schlüssel des Absenders

  Sie müssen den PEM-codierten privaten Schlüssel des Absenders (PKCS #8 PrivateKeyInfo - Strukturen) gemäß der Definition in RFC 5958 angeben.

- Der öffentliche Schlüssel des Empfängers

  Sie müssen den DER-codierten öffentlichen X.509-Schlüssel des Empfängers, auch bekannt als `SubjectPublicKeyInfo` (SPKI), wie in RFC 5280 definiert, angeben.

  Sie können den öffentlichen Schlüssel eines KMS-Schlüsselpaars mit asymmetrischer Schlüsselvereinbarung oder den öffentlichen Schlüssel eines außerhalb von AWS generierten key pair angeben.

- Spezifikation der Kurve

Identifiziert die Spezifikation für elliptische Kurven in den angegebenen Schlüsselpaaren. Sowohl die Schlüsselpaare des Absenders als auch des Empfängers müssen dieselbe Kurvenspezifikation haben.

Zulässige Werte: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

C# / .NET

```
// Instantiate material providers
var materialProviders = new MaterialProviders(new MaterialProvidersConfig());
     var BobPrivateKey = new MemoryStream(new byte[] { });
     var AlicePublicKey = new MemoryStream(new byte[] { });

     // Create the Raw ECDH static keyring
     var staticConfiguration = new RawEcdhStaticConfigurations()
     {
      RawPrivateKeyToStaticPublicKey = new RawPrivateKeyToStaticPublicKeyInput
      {
       SenderStaticPrivateKey = BobPrivateKey,
       RecipientPublicKey = AlicePublicKey
      }
     };

     var createKeyringInput = new CreateRawEcdhKeyringInput()
     {
      CurveSpec = ECDHCurveSpec.ECC_NIST_P256,
      KeyAgreementScheme = staticConfiguration
     };

     var keyring = materialProviders.CreateRawEcdhKeyring(createKeyringInput);
```

Java

Das folgende Java-Beispiel verwendet das `RawPrivateKeyToStaticPublicKey` Schlüsselvereinbarungsschema, um den privaten Schlüssel des Absenders und den öffentlichen Schlüssel des Empfängers statisch zu konfigurieren. Beide Schlüsselpaare befinden sich auf der `ECC_NIST_P256` Kurve.

```
private static void StaticRawKeyring() {
    // Instantiate material providers
    final MaterialProviders materialProviders =
```

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

    KeyPair senderKeys = GetRawEccKey();
    KeyPair recipient = GetRawEccKey();

    // Create the Raw ECDH static keyring
    final CreateRawEcdhKeyringInput rawKeyringInput =
      CreateRawEcdhKeyringInput.builder()
        .curveSpec(ECDHCurveSpec.ECC_NIST_P256)
        .KeyAgreementScheme(
          RawEcdhStaticConfigurations.builder()
            .RawPrivateKeyToStaticPublicKey(
                RawPrivateKeyToStaticPublicKeyInput.builder()
                  // Must be a PEM-encoded private key

 .senderStaticPrivateKey(ByteBuffer.wrap(senderKeys.getPrivate().getEncoded()))
                  // Must be a DER-encoded X.509 public key

 .recipientPublicKey(ByteBuffer.wrap(recipient.getPublic().getEncoded()))
                  .build()
            )
            .build()
        ).build();

    final IKeyring staticKeyring =
 materialProviders.CreateRawEcdhKeyring(rawKeyringInput);
}
```

Rust

Das folgende Python-Beispiel verwendet das `raw_ecdh_static_configuration` Schlüsselvereinbarungsschema, um den privaten Schlüssel des Absenders und den öffentlichen Schlüssel des Empfängers statisch zu konfigurieren. Beide Schlüsselpaare müssen sich auf derselben Kurve befinden.

```
// Create keyring input
let raw_ecdh_static_configuration_input =
    RawPrivateKeyToStaticPublicKeyInput::builder()
        // Must be a UTF8 PEM-encoded private key
        .sender_static_private_key(private_key_sender_utf8_bytes)
        // Must be a UTF8 DER-encoded X.509 public key
```

```
        .recipient_public_key(public_key_recipient_utf8_bytes)
        .build()?;

let raw_ecdh_static_configuration =
 RawEcdhStaticConfigurations::RawPrivateKeyToStaticPublicKey(raw_ecdh_static_configuration_i

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

// Create raw ECDH static keyring
let raw_ecdh_keyring = mpl
    .create_raw_ecdh_keyring()
    .curve_spec(ecdh_curve_spec)
    .key_agreement_scheme(raw_ecdh_static_configuration)
    .send()
    .await?;
```

EphemeralPrivateKeyToStaticPublicKey

Mit dem Schlüsselvereinbarungsschema konfigurierte `EphemeralPrivateKeyToStaticPublicKey` Schlüsselringe erstellen lokal ein neues key pair und leiten für jeden Verschlüsselungsaufruf einen eindeutigen gemeinsamen Wrapping-Schlüssel ab.

Dieses Schlüsselvereinbarungsschema kann nur Datensätze verschlüsseln. Um mit dem `EphemeralPrivateKeyToStaticPublicKey` Schlüsselvereinbarungsschema verschlüsselte Datensätze zu entschlüsseln, müssen Sie ein Discovery-Schlüsselvereinbarungsschema verwenden, das mit dem öffentlichen Schlüssel desselben Empfängers konfiguriert ist. Zum Entschlüsseln können Sie einen RAW-ECDH-Schlüsselbund mit dem `PublicKeyDiscovery`Schlüsselvereinbarungsalgorithmus verwenden, oder, falls der öffentliche Schlüssel des Empfängers aus einem KMS-Schlüsselpaar mit asymmetrischer Schlüsselvereinbarung stammt, können Sie einen AWS KMS ECDH-Schlüsselbund mit dem Schlüsselvereinbarungsschema verwenden. KmsPublicKeyDiscovery

Um einen Raw-ECDH-Schlüsselbund mit dem Schlüsselvereinbarungsschema zu initialisieren, geben Sie die folgenden Werte an`EphemeralPrivateKeyToStaticPublicKey`:

- Der öffentliche Schlüssel des Empfängers

Sie müssen den DER-codierten öffentlichen X.509-Schlüssel des Empfängers, auch bekannt als `SubjectPublicKeyInfo` (SPKI), wie in RFC 5280 definiert, angeben.

Sie können den öffentlichen Schlüssel eines KMS-Schlüsselpaars mit asymmetrischer Schlüsselvereinbarung oder den öffentlichen Schlüssel eines außerhalb von AWS generierten key pair angeben.

- Spezifikation der Kurve

  Identifiziert die Spezifikation für elliptische Kurven im angegebenen öffentlichen Schlüssel.

  Beim Verschlüsseln erstellt der Schlüsselbund ein neues key pair auf der angegebenen Kurve und verwendet den neuen privaten Schlüssel und den angegebenen öffentlichen Schlüssel, um einen gemeinsamen Wrapping-Schlüssel abzuleiten.

  Zulässige Werte: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

C# / .NET

Im folgenden Beispiel wird ein RAW-ECDH-Schlüsselbund mit dem Schlüsselvereinbarungsschema erstellt. `EphemeralPrivateKeyToStaticPublicKey` Beim Verschlüsseln erstellt der Schlüsselbund lokal auf der angegebenen `ECC_NIST_P256` Kurve ein neues key pair.

```
// Instantiate material providers
var materialProviders = new MaterialProviders(new MaterialProvidersConfig());
     var AlicePublicKey = new MemoryStream(new byte[] { });

     // Create the Raw ECDH ephemeral keyring
     var ephemeralConfiguration = new RawEcdhStaticConfigurations()
     {
      EphemeralPrivateKeyToStaticPublicKey = new
 EphemeralPrivateKeyToStaticPublicKeyInput
      {
       RecipientPublicKey = AlicePublicKey
      }
     };

     var createKeyringInput = new CreateRawEcdhKeyringInput()
     {
      CurveSpec = ECDHCurveSpec.ECC_NIST_P256,
```

```
      KeyAgreementScheme = ephemeralConfiguration
     };

     var keyring = materialProviders.CreateRawEcdhKeyring(createKeyringInput);
```

Java

Im folgenden Beispiel wird ein RAW-ECDH-Schlüsselbund mit dem `EphemeralPrivateKeyToStaticPublicKey` Schlüsselvereinbarungsschema erstellt. Beim Verschlüsseln erstellt der Schlüsselbund lokal auf der angegebenen `ECC_NIST_P256` Kurve ein neues key pair.

```
private static void EphemeralRawEcdhKeyring() {
    // Instantiate material providers
    final MaterialProviders materialProviders =
      MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();

    ByteBuffer recipientPublicKey = getPublicKeyBytes();

    // Create the Raw ECDH ephemeral keyring
    final CreateRawEcdhKeyringInput ephemeralInput =
      CreateRawEcdhKeyringInput.builder()
        .curveSpec(ECDHCurveSpec.ECC_NIST_P256)
        .KeyAgreementScheme(
          RawEcdhStaticConfigurations.builder()
            .EphemeralPrivateKeyToStaticPublicKey(
              EphemeralPrivateKeyToStaticPublicKeyInput.builder()
                .recipientPublicKey(recipientPublicKey)
                .build()
            )
            .build()
        ).build();

    final IKeyring ephemeralKeyring =
 materialProviders.CreateRawEcdhKeyring(ephemeralInput);
}
```

Rust

Im folgenden Beispiel wird ein RAW-ECDH-Schlüsselbund mit dem `ephemeral_raw_ecdh_static_configuration` Schlüsselvereinbarungsschema erstellt.

Beim Verschlüsseln erstellt der Schlüsselbund lokal auf der angegebenen Kurve ein neues key pair.

```
// Create EphemeralPrivateKeyToStaticPublicKeyInput
let ephemeral_raw_ecdh_static_configuration_input =
    EphemeralPrivateKeyToStaticPublicKeyInput::builder()
        // Must be a UTF8 DER-encoded X.509 public key
        .recipient_public_key(public_key_recipient_utf8_bytes)
        .build()?;

let ephemeral_raw_ecdh_static_configuration =

 RawEcdhStaticConfigurations::EphemeralPrivateKeyToStaticPublicKey(ephemeral_raw_ecdh_static

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

// Create raw ECDH ephemeral private key keyring
let ephemeral_raw_ecdh_keyring = mpl
    .create_raw_ecdh_keyring()
    .curve_spec(ecdh_curve_spec)
    .key_agreement_scheme(ephemeral_raw_ecdh_static_configuration)
    .send()
    .await?;
```

PublicKeyDiscovery

Beim Entschlüsseln empfiehlt es sich, die Umschließungsschlüssel anzugeben, die das AWS Database Encryption SDK verwenden kann. Um dieser bewährten Methode zu folgen, verwenden Sie einen ECDH-Schlüsselbund, der sowohl den privaten Schlüssel eines Absenders als auch den öffentlichen Schlüssel des Empfängers angibt. Sie können jedoch auch einen Raw ECDH Discovery-Schlüsselbund erstellen, d. h. einen Raw ECDH-Schlüsselbund, der jeden Datensatz entschlüsseln kann, bei dem der öffentliche Schlüssel des angegebenen Schlüssels mit dem öffentlichen Schlüssel des Empfängers übereinstimmt, der im Materialbeschreibungsfeld des verschlüsselten Datensatzes gespeichert ist. Dieses Schlüsselvereinbarungsschema kann nur Datensätze entschlüsseln.

> **Important**
>
> Wenn Sie Datensätze mithilfe des `PublicKeyDiscovery` Schlüsselvereinbarungsschemas entschlüsseln, akzeptieren Sie alle öffentlichen Schlüssel, unabhängig davon, wem sie gehören.

Um einen RAW-ECDH-Schlüsselbund mit dem Schlüsselvereinbarungsschema zu initialisieren, geben Sie die `PublicKeyDiscovery` folgenden Werte an:

- Statischer privater Schlüssel des Empfängers

  Sie müssen den PEM-codierten privaten Schlüssel des Empfängers (PKCS #8 PrivateKeyInfo - Strukturen) gemäß der Definition in RFC 5958 angeben.

- Spezifikation der Kurve

  Identifiziert die Spezifikation für elliptische Kurven im angegebenen privaten Schlüssel. Sowohl die Schlüsselpaare des Absenders als auch des Empfängers müssen dieselbe Kurvenspezifikation haben.

  Zulässige Werte: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

C# / .NET

Im folgenden Beispiel wird ein Raw ECDH-Schlüsselbund mit dem `PublicKeyDiscovery` Schlüsselvereinbarungsschema erstellt. Dieser Schlüsselbund kann jeden Datensatz entschlüsseln, bei dem der öffentliche Schlüssel des angegebenen privaten Schlüssels mit dem öffentlichen Schlüssel des Empfängers übereinstimmt, der im Materialbeschreibungsfeld des verschlüsselten Datensatzes gespeichert ist.

```
// Instantiate material providers
var materialProviders = new MaterialProviders(new MaterialProvidersConfig());
     var AlicePrivateKey = new MemoryStream(new byte[] { });

     // Create the Raw ECDH discovery keyring
     var discoveryConfiguration = new RawEcdhStaticConfigurations()
     {
      PublicKeyDiscovery = new PublicKeyDiscoveryInput
      {
       RecipientStaticPrivateKey = AlicePrivateKey
```

```
     }
    };

    var createKeyringInput = new CreateRawEcdhKeyringInput()
    {
     CurveSpec = ECDHCurveSpec.ECC_NIST_P256,
     KeyAgreementScheme = discoveryConfiguration
    };

    var keyring = materialProviders.CreateRawEcdhKeyring(createKeyringInput);
```

Java

Im folgenden Beispiel wird ein RAW-ECDH-Schlüsselbund mit dem `PublicKeyDiscovery` Schlüsselvereinbarungsschema erstellt. Dieser Schlüsselbund kann jeden Datensatz entschlüsseln, bei dem der öffentliche Schlüssel des angegebenen privaten Schlüssels mit dem öffentlichen Schlüssel des Empfängers übereinstimmt, der im Materialbeschreibungsfeld des verschlüsselten Datensatzes gespeichert ist.

```
private static void RawEcdhDiscovery() {
    // Instantiate material providers
    final MaterialProviders materialProviders =
      MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();

    KeyPair recipient = GetRawEccKey();

    // Create the Raw ECDH discovery keyring
    final CreateRawEcdhKeyringInput rawKeyringInput =
      CreateRawEcdhKeyringInput.builder()
        .curveSpec(ECDHCurveSpec.ECC_NIST_P256)
        .KeyAgreementScheme(
          RawEcdhStaticConfigurations.builder()
            .PublicKeyDiscovery(
              PublicKeyDiscoveryInput.builder()
                // Must be a PEM-encoded private key

 .recipientStaticPrivateKey(ByteBuffer.wrap(sender.getPrivate().getEncoded()))
                .build()
            )
            .build()
        ).build();
```

```
    final IKeyring publicKeyDiscovery  =
 materialProviders.CreateRawEcdhKeyring(rawKeyringInput);
}
```

Rust

Im folgenden Beispiel wird ein RAW-ECDH-Schlüsselbund mit dem `discovery_raw_ecdh_static_configuration` Schlüsselvereinbarungsschema erstellt. Dieser Schlüsselbund kann jede Nachricht entschlüsseln, bei der der öffentliche Schlüssel des angegebenen privaten Schlüssels mit dem öffentlichen Schlüssel des Empfängers übereinstimmt, der im Chiffretext der Nachricht gespeichert ist.

```
// Create PublicKeyDiscoveryInput
let discovery_raw_ecdh_static_configuration_input =
    PublicKeyDiscoveryInput::builder()
        // Must be a UTF8 PEM-encoded private key
        .recipient_static_private_key(private_key_recipient_utf8_bytes)
        .build()?;

let discovery_raw_ecdh_static_configuration =

 RawEcdhStaticConfigurations::PublicKeyDiscovery(discovery_raw_ecdh_static_configuration_inp

// Create raw ECDH discovery private key keyring
let discovery_raw_ecdh_keyring = mpl
    .create_raw_ecdh_keyring()
    .curve_spec(ecdh_curve_spec)
    .key_agreement_scheme(discovery_raw_ecdh_static_configuration)
    .send()
    .await?;
```

Multi-Schlüsselbunde

Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Sie können Schlüsselbunde zu einem Multi-Schlüsselbund kombinieren. Ein Multi-Schlüsselbund ist ein Schlüsselbund, der aus einem oder mehreren einzelnen Schlüsselbunden desselben oder eines anderen Typs besteht. Das hat den gleichen Effekt wie die Verwendung von mehreren Schlüsselbunden in einer Reihe. Wenn Sie einen Multi-Schlüsselbund verwenden, um Daten zu verschlüsseln, können alle Umhüllungsschlüssel in einem seiner Schlüsselbunde diese Daten entschlüsseln.

Wenn Sie einen Multi-Schlüsselbund erstellen, um Daten zu verschlüsseln, geben Sie einen der Schlüsselbunde als Generator-Schlüsselbund an. Alle anderen Schlüsselbunde werden als untergeordnete Schlüsselbunde bezeichnet. Der Generator-Schlüsselbund generiert und verschlüsselt den Klartext-Datenschlüssel. Anschließend verschlüsseln alle Umhüllungsschlüssel in den untergeordneten Schlüsselbunden den gleichen Klartext-Datenschlüssel. Der Multi-Schlüsselbund gibt den Klartext-Datenschlüssel und einen verschlüsselten Datenschlüssel für jeden Umhüllungsschlüssel im Multi-Schlüsselbund zurück. Wenn der Generator-Schlüsselbund ein KMS-Schlüsselbund ist, generiert und verschlüsselt der Generatorschlüssel im AWS KMS Schlüsselbund den Klartext-Schlüssel. Dann verschlüsseln alle zusätzlichen Schlüssel AWS KMS keys im AWS KMS Schlüsselbund und alle Schlüssel in allen untergeordneten Schlüsselbunden im Mehrfachschlüsselbund denselben Klartext-Schlüssel.

Beim Entschlüsseln versucht das AWS Database Encryption SDK anhand der Schlüsselbunde, einen der verschlüsselten Datenschlüssel zu entschlüsseln. Die Schlüsselbunde werden in der Reihenfolge aufgerufen, in der sie im Multi-Schlüsselbund angegeben sind. Die Verarbeitung stoppt, sobald ein Schlüssel in einem Schlüsselbund einen verschlüsselten Datenschlüssel entschlüsseln kann.

Zum Erstellen eines Multi-Schlüsselbunds müssen Sie zuerst die untergeordneten Schlüsselbunde instanziieren. In diesem Beispiel verwenden wir einen Schlüsselbund und einen AWS KMS Raw-AES-Schlüsselbund, aber Sie können jeden unterstützten Schlüsselbund zu einem Mehrfachschlüsselbund kombinieren.

Java

```
// 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);
```

C# / .NET

```
// 1. Create the raw AES keyring.
var keyNamespace = "HSM_01";
var keyName = "AES_256_012";

var matProv = new MaterialProviders(new MaterialProvidersConfig());
var createRawAesKeyringInput = new CreateRawAesKeyringInput
{
    KeyName = "keyName",
    KeyNamespace = "myNamespaces",
    WrappingKey = AESWrappingKey,
    WrappingAlg = AesWrappingAlg.ALG_AES256_GCM_IV12_TAG16
};
var rawAesKeyring = matProv.CreateRawAesKeyring(createRawAesKeyringInput);

// 2. Create the AWS KMS keyring.
//    We create a MRK multi keyring, as this interface also supports
//    single-region KMS keys,
//    and creates the KMS client for us automatically.
var createAwsKmsMrkMultiKeyringInput = new CreateAwsKmsMrkMultiKeyringInput
{
    Generator = keyArn
};
var awsKmsMrkMultiKeyring =
 matProv.CreateAwsKmsMrkMultiKeyring(createAwsKmsMrkMultiKeyringInput);
```

Rust

```
// 1. Create the raw AES keyring
let mpl_config = MaterialProvidersConfig::builder().build()?;
```

```
let mpl = mpl_client::Client::from_conf(mpl_config)?;

let raw_aes_keyring = mpl
    .create_raw_aes_keyring()
    .key_name("AES_256_012")
    .key_namespace("HSM_01")
    .wrapping_key(aes_key_bytes)
    .wrapping_alg(AesWrappingAlg::AlgAes256GcmIv12Tag16)
    .send()
    .await?;

// 2. Create the AWS KMS keyring
let aws_kms_mrk_multi_keyring = mpl
    .create_aws_kms_mrk_multi_keyring()
    .generator(key_arn)
    .send()
    .await?;
```

Erstellen Sie als Nächstes den Multi-Schlüsselbund und geben Sie seinen Generator-Schlüsselbund an, falls vorhanden. In diesem Beispiel erstellen wir einen Mehrfachschlüsselbund, bei dem der Schlüsselbund der Generatorschlüsselbund und der AWS KMS AES-Schlüsselbund der untergeordnete Schlüsselbund ist.

Java

Mit dem `CreateMultiKeyringInput` Java-Konstruktor können Sie einen Generator-Schlüsselbund und untergeordnete Schlüsselanhänger definieren. Das resultierende `createMultiKeyringInput` Objekt ist unveränderlich.

```
final CreateMultiKeyringInput createMultiKeyringInput =
 CreateMultiKeyringInput.builder()
        .generator(awsKmsMrkMultiKeyring)
        .childKeyrings(Collections.singletonList(rawAesKeyring))
        .build();
IKeyring multiKeyring = matProv.CreateMultiKeyring(createMultiKeyringInput);
```

C# / .NET

Mit `CreateMultiKeyringInput` dem.NET-Konstruktor können Sie einen Generator-Schlüsselbund und untergeordnete Schlüsselringe definieren. Das resultierende `CreateMultiKeyringInput` Objekt ist unveränderlich.

```
var createMultiKeyringInput = new CreateMultiKeyringInput
{
    Generator = awsKmsMrkMultiKeyring,
    ChildKeyrings = new List<IKeyring> { rawAesKeyring }
};
var multiKeyring = matProv.CreateMultiKeyring(createMultiKeyringInput);
```

Rust

```
let multi_keyring = mpl
    .create_multi_keyring()
    .generator(aws_kms_mrk_multi_keyring)
    .child_keyrings(vec![raw_aes_keyring.clone()])
    .send()
    .await?;
```

Jetzt können Sie mit dem Multi-Schlüsselbund Daten ver- und entschlüsseln.

Durchsuchbare Verschlüsselung

Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Mit der durchsuchbaren Verschlüsselung können Sie verschlüsselte Datensätze durchsuchen, ohne die gesamte Datenbank entschlüsseln zu müssen. Dies wird mithilfe von Beacons erreicht, die eine Zuordnung zwischen dem Klartextwert, der in ein Feld geschrieben wird, und dem verschlüsselten Wert, der tatsächlich in Ihrer Datenbank gespeichert ist, erstellen. Das AWS Database Encryption SDK speichert den Beacon in einem neuen Feld, das es dem Datensatz hinzufügt. Je nach verwendetem Beacontyp können Sie nach Ihren verschlüsselten Daten nach exakten Übereinstimmungen suchen oder individuellere komplexe Abfragen durchführen.

> **Note**
>
> Die durchsuchbare Verschlüsselung im AWS Database Encryption SDK unterscheidet sich von der durchsuchbaren symmetrischen Verschlüsselung, die in der akademischen Forschung definiert wurde, z. B. durchsuchbare symmetrische Verschlüsselung.

Ein Beacon ist ein gekürztes HMAC-Tag (Hash-Based Message Authentication Code), das eine Zuordnung zwischen Klartext- und verschlüsselten Werten eines Felds erstellt. Wenn Sie einen neuen Wert in ein verschlüsseltes Feld schreiben, das für durchsuchbare Verschlüsselung konfiguriert ist, berechnet das AWS Database Encryption SDK einen HMAC-Wert über dem Klartext-Wert. Bei dieser HMAC-Ausgabe handelt es sich um eine 1:1 -Übereinstimmung mit dem Klartextwert dieses Felds. Die HMAC-Ausgabe wird gekürzt, sodass mehrere unterschiedliche Klartextwerte demselben gekürzten HMAC-Tag zugeordnet werden. Diese Fehlalarme schränken die Fähigkeit eines nicht autorisierten Benutzers ein, charakteristische Informationen über den Klartext-Wert zu identifizieren. Wenn Sie einen Beacon abfragen, filtert das AWS Database Encryption SDK diese Fehlalarme automatisch heraus und gibt das Klartextergebnis Ihrer Abfrage zurück.

Die durchschnittliche Anzahl der für jeden Beacon generierten Fehlalarme wird durch die Länge des Beacons bestimmt, die nach der Kürzung noch übrig ist. Hilfe zur Bestimmung der geeigneten Beacon-Länge für Ihre Implementierung finden Sie unter Bestimmung der Beacon-Länge.

> **Note**
>
> Die durchsuchbare Verschlüsselung ist so konzipiert, dass sie in neuen, nicht aufgefüllten Datenbanken implementiert werden kann. Jedes Beacon, das in einer vorhandenen Datenbank konfiguriert ist, ordnet nur neue Datensätze zu, die in die Datenbank hochgeladen wurden. Es gibt keine Möglichkeit für ein Beacon, bestehende Daten zuzuordnen.

Topics

Sind Beacons das Richtige für meinen Datensatz?

Die Verwendung von Beacons zur Durchführung von Abfragen verschlüsselter Daten reduziert die Leistungskosten, die mit clientseitig verschlüsselten Datenbanken verbunden sind. Wenn Sie Beacons verwenden, gibt es einen inhärenten Kompromiss zwischen der Effizienz Ihrer Abfragen und der Menge an Informationen, die über die Verteilung Ihrer Daten preisgegeben werden. Der Beacon verändert den verschlüsselten Zustand des Feldes nicht. Wenn Sie ein Feld mit dem AWS Database Encryption SDK verschlüsseln und signieren, wird der Klartextwert des Felds niemals der Datenbank zugänglich gemacht. Die Datenbank speichert den zufälligen, verschlüsselten Wert des Felds.

Beacons werden zusammen mit den verschlüsselten Feldern gespeichert, aus denen sie berechnet werden. Das heißt, selbst wenn ein nicht autorisierter Benutzer die Klartextwerte eines verschlüsselten Felds nicht einsehen kann, kann er möglicherweise statistische Analysen der Beacons durchführen, um mehr über die Verteilung Ihres Datensatzes zu erfahren und im Extremfall die Klartextwerte zu identifizieren, denen ein Beacon zugeordnet ist. Die Art und Weise, wie Sie Ihre Beacons konfigurieren, kann diese Risiken mindern. Insbesondere die Wahl der richtigen Beacon-Länge kann Ihnen helfen, die Vertraulichkeit Ihres Datensatzes zu wahren.

Sicherheit versus Leistung

- Je kürzer die Länge des Beacons, desto besser ist die Sicherheit gewährleistet.

- Je länger die Länge des Beacons ist, desto mehr Leistung bleibt erhalten.

Eine durchsuchbare Verschlüsselung kann möglicherweise nicht das gewünschte Leistungs- und Sicherheitsniveau für alle Datensätze bieten. Überprüfen Sie Ihr Bedrohungsmodell, Ihre Sicherheits- und Leistungsanforderungen, bevor Sie Beacons konfigurieren.

Beachten Sie die folgenden Anforderungen an die Eindeutigkeit von Datensätzen, wenn Sie entscheiden, ob eine durchsuchbare Verschlüsselung für Ihren Datensatz geeignet ist.

Verteilung

Wie viel Sicherheit durch ein Beacon gewährleistet wird, hängt von der Verteilung Ihres Datensatzes ab. Wenn Sie ein verschlüsseltes Feld für durchsuchbare Verschlüsselung konfigurieren, berechnet das AWS Database Encryption SDK anhand der in dieses Feld geschriebenen Klartextwerte einen HMAC. Alle für ein bestimmtes Feld berechneten Beacons werden mit demselben Schlüssel berechnet, mit Ausnahme von Multitenant-Datenbanken, die für jeden Mandanten einen eigenen Schlüssel verwenden. Das heißt, wenn derselbe Klartextwert mehrmals in das Feld geschrieben wird, wird für jede Instanz dieses Klartextwerts derselbe HMAC-Tag erstellt.

Sie sollten vermeiden, Beacons aus Feldern zu erstellen, die sehr häufig vorkommende Werte enthalten. Stellen Sie sich zum Beispiel eine Datenbank vor, in der die Adressen aller Einwohner des Bundesstaates Illinois gespeichert sind. Wenn Sie aus dem verschlüsselten `City` Feld einen Beacon konstruieren, wird der über „Chicago“ berechnete Beacon aufgrund des hohen Prozentsatzes der Bevölkerung von Illinois, der in Chicago lebt, überrepräsentiert sein. Selbst wenn ein nicht autorisierter Benutzer nur die verschlüsselten Werte und Beacon-Werte lesen kann, kann er möglicherweise feststellen, welche Datensätze Daten für Einwohner von Chicago enthalten, wenn das Beacon diese Verteilung beibehält. Um die Menge an identifizierenden Informationen, die über Ihre Verteilung preisgegeben werden, zu minimieren, müssen Sie Ihr Beacon ausreichend kürzen. Die Länge des Beacons, die erforderlich ist, um diese ungleichmäßige Verteilung zu verbergen, ist mit erheblichen Leistungseinbußen verbunden, die möglicherweise nicht den Anforderungen Ihrer Anwendung entsprechen.

Sie müssen die Verteilung Ihres Datensatzes sorgfältig analysieren, um festzustellen, wie stark Ihre Beacons gekürzt werden müssen. Die Länge der Beacons, die nach der Kürzung noch übrig ist, steht in direktem Zusammenhang mit der Menge an statistischen Informationen, die über Ihre Verteilung ermittelt werden können. Möglicherweise müssen Sie kürzere Beacon-Längen wählen,

um die Menge an Unterscheidungsinformationen, die über Ihren Datensatz preisgegeben werden, ausreichend zu minimieren.

In extremen Fällen können Sie keine Beacon-Länge für einen ungleichmäßig verteilten Datensatz berechnen, der ein effektives Gleichgewicht zwischen Leistung und Sicherheit gewährleistet. Sie sollten beispielsweise keinen Beacon aus einem Feld erstellen, in dem das Ergebnis eines medizinischen Tests für eine seltene Krankheit gespeichert ist. Da davon ausgegangen wird, dass `NEGATIVE` Ergebnisse innerhalb des Datensatzes deutlich häufiger vorkommen, können `POSITIVE` Ergebnisse leicht anhand ihrer Seltenheit identifiziert werden. Es ist sehr schwierig, die Verteilung zu verbergen, wenn das Feld nur zwei mögliche Werte hat. Wenn Sie eine Beacon-Länge verwenden, die kurz genug ist, um die Verteilung zu verbergen, werden alle Klartextwerte demselben HMAC-Tag zugeordnet. Wenn Sie eine längere Beacon-Länge verwenden, ist es offensichtlich, welche Beacons den Klartext-Werten zugeordnet werden. `POSITIVE`

Korrelation

Wir empfehlen dringend, dass Sie vermeiden, unterschiedliche Beacons aus Feldern mit korrelierten Werten zu erstellen. Beacons, die aus korrelierten Feldern erstellt wurden, erfordern kürzere Beacon-Längen, um die Menge an Informationen, die über die Verteilung der einzelnen Datensätze an einen nicht autorisierten Benutzer preisgegeben werden, ausreichend zu minimieren. Sie müssen Ihren Datensatz sorgfältig analysieren, einschließlich seiner Entropie und der gemeinsamen Verteilung der korrelierten Werte, um festzustellen, wie stark Ihre Beacons gekürzt werden müssen. Wenn die resultierende Beacon-Länge nicht Ihren Leistungsanforderungen entspricht, sind Beacons möglicherweise nicht für Ihren Datensatz geeignet.

Sie sollten beispielsweise nicht zwei separate Beacons aus `City` und `ZIPCode` -Feldern erstellen, da die Postleitzahl wahrscheinlich nur einer Stadt zugeordnet ist. In der Regel schränken die von einem Beacon generierten Fehlalarme die Fähigkeit eines nicht autorisierten Benutzers ein, charakteristische Informationen über Ihren Datensatz zu identifizieren. Die Korrelation zwischen den `ZIPCode` Feldern `City` und bedeutet jedoch, dass ein nicht autorisierter Benutzer leicht erkennen kann, welche Ergebnisse falsch positive Ergebnisse sind, und die verschiedenen Postleitzahlen unterscheiden kann.

Sie sollten auch vermeiden, Beacons aus Feldern zu erstellen, die dieselben Klartextwerte enthalten. Beispielsweise sollten Sie kein Beacon aus `preferredPhone` Feldern `mobilePhone` und erstellen, da diese wahrscheinlich dieselben Werte enthalten. Wenn Sie aus beiden Feldern unterschiedliche Beacons erstellen, erstellt das AWS Database Encryption SDK die Beacons für jedes Feld unter unterschiedlichen Schlüsseln. Dies führt zu zwei verschiedenen HMAC-

Tags für denselben Klartext-Wert. Es ist unwahrscheinlich, dass die beiden unterschiedlichen Beacons dieselben Fehlalarme haben, und ein nicht autorisierter Benutzer kann möglicherweise verschiedene Telefonnummern unterscheiden.

Selbst wenn Ihr Datensatz korrelierte Felder enthält oder eine ungleichmäßige Verteilung aufweist, können Sie möglicherweise Beacons erstellen, die die Vertraulichkeit Ihres Datensatzes wahren, indem Sie kürzere Beacon-Längen verwenden. Die Beacon-Länge garantiert jedoch nicht, dass jeder eindeutige Wert in Ihrem Datensatz zu einer Reihe von Fehlalarmen führt, wodurch die Menge an Unterscheidungsinformationen, die über Ihren Datensatz preisgegeben werden, effektiv minimiert wird. Mit der Beacon-Länge wird lediglich die durchschnittliche Anzahl der generierten Fehlalarme geschätzt. Je ungleichmäßiger Ihr Datensatz verteilt ist, desto weniger effektiv ist die Beacon-Länge bei der Bestimmung der durchschnittlichen Anzahl der erzeugten Fehlalarme.

Berücksichtigen Sie sorgfältig die Verteilung der Felder, aus denen Sie Beacons erstellen, und überlegen Sie, um wie viel Sie die Beacon-Länge kürzen müssen, um Ihre Sicherheitsanforderungen zu erfüllen. Bei den folgenden Themen in diesem Kapitel wird davon ausgegangen, dass Ihre Beacons gleichmäßig verteilt sind und keine korrelierten Daten enthalten.

Durchsuchbares Verschlüsselungsszenario

Das folgende Beispiel zeigt eine einfache durchsuchbare Verschlüsselungslösung. In der Anwendung entsprechen die in diesem Beispiel verwendeten Beispielfelder möglicherweise nicht den Empfehlungen zur Verteilung und Korrelation zur Eindeutigkeit von Beacons. Sie können dieses Beispiel als Referenz verwenden, wenn Sie in diesem Kapitel mehr über die Konzepte der durchsuchbaren Verschlüsselung lesen.

Stellen Sie sich eine Datenbank mit dem Namen vor`Employees`, die Mitarbeiterdaten eines Unternehmens verfolgt. Jeder Datensatz in der Datenbank enthält Felder namens EmployeeID, LastNameFirstName, und Address. Jedes Feld in der `Employees` Datenbank wird durch den Primärschlüssel `EmployeeID` identifiziert.

Im Folgenden finden Sie ein Beispiel für einen Klartext-Datensatz in der Datenbank.

```
{
    "EmployeeID": 101,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
                "Street": "123 Main",
```

```
                "City": "Anytown",
                "State": "OH",
                "ZIPCode": 12345
    }
}
```

Wenn Sie `ENCRYPT_AND_SIGN` in Ihren kryptografischen Aktionen die `FirstName` Felder `LastName` und als markiert haben, werden die Werte in diesen Feldern lokal verschlüsselt, bevor sie in die Datenbank hochgeladen werden. Die verschlüsselten Daten, die hochgeladen werden, sind vollständig randomisiert. Die Datenbank erkennt diese Daten nicht als geschützt. Sie erkennt nur typische Dateneinträge. Das bedeutet, dass der Datensatz, der tatsächlich in der Datenbank gespeichert ist, wie folgt aussehen könnte.

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

Wenn Sie die Datenbank nach exakten Übereinstimmungen im `LastName` Feld abfragen müssen, konfigurieren Sie einen Standard-Beacon, der so benannt ist, LastNamedass er die in das `LastName` Feld geschriebenen Klartextwerte den in der Datenbank gespeicherten verschlüsselten Werten zuordnet.

Dieser Beacon berechnet anhand HMACs der Klartextwerte im Feld. `LastName` Jede HMAC-Ausgabe wird gekürzt, sodass sie nicht mehr exakt dem Klartext-Wert entspricht. Der vollständige Hash und der gekürzte Hash für `Jones` könnten beispielsweise wie folgt aussehen.

Vollständiger Hash

```
2aa4e9b404c68182562b6ec761fcca5306de527826a69468885e59dc36d0c3f824bdd44cab45526f
```

Verkürzter Hash

```
b35099d408c833
```

Nachdem der Standard-Beacon konfiguriert wurde, können Sie Gleichheitssuchen für das `LastName` Feld durchführen. Wenn Sie beispielsweise nach suchen möchten`Jones`, verwenden Sie den LastNameBeacon, um die folgende Abfrage durchzuführen.

```
LastName = Jones
```

Das AWS Database Encryption SDK filtert automatisch die Fehlalarme heraus und gibt das Klartextergebnis Ihrer Abfrage zurück.

Leuchtfeuer

> Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Ein Beacon ist ein gekürztes HMAC-Tag (Hash-Based Message Authentication Code), das eine Zuordnung zwischen dem Klartextwert, der in ein Feld geschrieben wird, und dem verschlüsselten Wert, der tatsächlich in Ihrer Datenbank gespeichert ist, erstellt. Der Beacon ändert den verschlüsselten Status des Feldes nicht. Der Beacon berechnet anhand des Klartextwerts des Felds einen HMAC und speichert ihn zusammen mit dem verschlüsselten Wert. Bei dieser HMAC-Ausgabe handelt es sich um eine 1:1 -Übereinstimmung mit dem Klartextwert dieses Felds. Die HMAC-Ausgabe wird gekürzt, sodass mehrere unterschiedliche Klartextwerte demselben gekürzten HMAC-Tag zugeordnet werden. Diese Fehlalarme schränken die Fähigkeit eines nicht autorisierten Benutzers ein, charakteristische Informationen über den Klartext-Wert zu identifizieren.

Beacons können nur aus Feldern erstellt werden, die markiert sind `ENCRYPT_AND_SIGNSIGN_ONLY`, oder `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` in Ihren kryptografischen Aktionen enthalten sind. Der Beacon selbst ist weder signiert noch verschlüsselt. Sie können kein Beacon mit markierten Feldern erstellen. `DO_NOTHING`

Der Typ des Beacons, den Sie konfigurieren, bestimmt die Art der Abfragen, die Sie ausführen können. Es gibt zwei Arten von Beacons, die durchsuchbare Verschlüsselung unterstützen. Standard-Beacons führen Gleichheitssuchen durch. Zusammengesetzte Beacons kombinieren wörtliche Klartext-Zeichenketten und Standard-Beacons, um komplexe Datenbankoperationen durchzuführen. Nachdem Sie Ihre Beacons konfiguriert haben, müssen Sie für jedes Beacon einen sekundären Index konfigurieren, bevor Sie in den verschlüsselten Feldern suchen können. Weitere Informationen finden Sie unter Konfiguration sekundärer Indizes mit Beacons.

Themen

Standard-Beacons

Standard-Beacons sind die einfachste Methode, eine durchsuchbare Verschlüsselung in Ihrer Datenbank zu implementieren. Sie können nur Gleichheitssuchen für ein einzelnes verschlüsseltes oder virtuelles Feld durchführen. Informationen zur Konfiguration von Standard-Beacons finden Sie unter Konfiguration von Standard-Beacons.

Das Feld, aus dem ein Standard-Beacon erstellt wird, wird als Beacon-Quelle bezeichnet. Es identifiziert den Standort der Daten, die der Beacon für die Kartierung benötigt. Die Beacon-Quelle kann entweder ein verschlüsseltes Feld oder ein virtuelles Feld sein. Die Beacon-Quelle in jedem Standard-Beacon muss eindeutig sein. Sie können nicht zwei Beacons mit derselben Beacon-Quelle konfigurieren.

Standard-Beacons können verwendet werden, um Gleichheitssuchen für ein verschlüsseltes oder virtuelles Feld durchzuführen. Sie können auch verwendet werden, um zusammengesetzte Beacons für komplexere Datenbankoperationen zu erstellen. Um Ihnen bei der Organisation und Verwaltung von Standard-Beacons zu helfen, bietet das AWS Database Encryption SDK die folgenden optionalen Beacon-Stile, die den Verwendungszweck eines Standard-Beacons definieren. Weitere Informationen finden Sie unter Beacon-Stile definieren.

Sie können ein Standard-Beacon erstellen, das Gleichheitssuchen für ein einzelnes verschlüsseltes Feld durchführt, oder Sie können ein Standard-Beacon erstellen, das Gleichheitssuchen bei der Verkettung mehrererENCRYPT_AND_SIGN, und SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT - Felder durchführtSIGN_ONLY, indem Sie ein virtuelles Feld erstellen.

Virtuelle Felder

Ein virtuelles Feld ist ein konzeptionelles Feld, das aus einem oder mehreren Quellfeldern besteht. Beim Erstellen eines virtuellen Felds wird kein neues Feld in Ihren Datensatz geschrieben. Das virtuelle Feld wird nicht explizit in Ihrer Datenbank gespeichert. Es wird in der

Standard-Beacon-Konfiguration verwendet, um dem Beacon Anweisungen zu geben, wie ein bestimmtes Segment eines Feldes identifiziert oder mehrere Felder innerhalb eines Datensatzes verkettet werden kann, um eine bestimmte Abfrage durchzuführen. Ein virtuelles Feld erfordert mindestens ein verschlüsseltes Feld.

> **Note**
>
> Das folgende Beispiel zeigt, welche Arten von Transformationen und Abfragen Sie mit einem virtuellen Feld durchführen können. In der Anwendung entsprechen die in diesem Beispiel verwendeten Beispielfelder möglicherweise nicht den Empfehlungen zur Verteilung und Korrelationseindeutigkeit für Beacons.

Wenn Sie beispielsweise Gleichheitssuchen für die Verkettung von `FirstName` und `LastName` -Feldern durchführen möchten, können Sie eines der folgenden virtuellen Felder erstellen.

- Ein virtuelles `NameTag` Feld, das aus dem ersten Buchstaben des `FirstName` Felds, gefolgt vom Feld, gebildet wird, alles in Kleinbuchstaben. `LastName` Mit diesem virtuellen Feld können Sie Abfragen `NameTag=mjones` durchführen.
- Ein virtuelles `LastFirst` Feld, das aus dem `LastName` Feld, gefolgt vom `FirstName` Feld, aufgebaut wird. Mit diesem virtuellen Feld können Sie Abfragen durchführen`LastFirst=JonesMary`.

Oder, wenn Sie Gleichheitssuchen für ein bestimmtes Segment eines verschlüsselten Feldes durchführen möchten, erstellen Sie ein virtuelles Feld, das das Segment identifiziert, das Sie abfragen möchten.

Wenn Sie beispielsweise ein verschlüsseltes `IPAddress` Feld anhand der ersten drei Segmente der IP-Adresse abfragen möchten, erstellen Sie das folgende virtuelle Feld.

- Ein virtuelles `IPSegment` Feld, aufgebaut aus`Segments('.', 0, 3)`. Mit diesem virtuellen Feld können Sie Abfragen durchführen`IPSegment=192.0.2`. Die Abfrage gibt alle Datensätze zurück, deren `IPAddress` Wert mit „192.0.2" beginnt.

Virtuelle Felder müssen eindeutig sein. Zwei virtuelle Felder können nicht aus exakt denselben Quellfeldern erstellt werden.

Hilfe zur Konfiguration virtueller Felder und der Beacons, die sie verwenden, finden Sie unter Virtuelles Feld erstellen.

Zusammengesetzte Beacons

Zusammengesetzte Beacons erstellen Indizes, die die Abfrageleistung verbessern und es Ihnen ermöglichen, komplexere Datenbankoperationen durchzuführen. Sie können zusammengesetzte Beacons verwenden, um literale Klartextzeichenfolgen und Standardbeacons zu kombinieren, um komplexe Abfragen an verschlüsselten Datensätzen durchzuführen, z. B. um zwei verschiedene Datensatztypen aus einem einzigen Index abzufragen oder um eine Kombination von Feldern mit einem Sortierschlüssel abzufragen. Weitere Lösungsbeispiele für zusammengesetzte Beacons finden Sie unter Wählen Sie einen Beacon-Typ.

Zusammengesetzte Beacons können aus Standardbeacons oder einer Kombination aus Standardbeacons und signierten Feldern erstellt werden. Sie bestehen aus einer Liste von Teilen. Alle zusammengesetzten Beacons sollten eine Liste verschlüsselter Teile enthalten, die die im Beacon enthaltenen `ENCRYPT_AND_SIGN` Felder identifiziert. Jedes `ENCRYPT_AND_SIGN` Feld muss durch einen Standard-Beacon identifiziert werden. Komplexere zusammengesetzte Beacons können auch eine Liste von signierten Teilen enthalten, die den Klartext `SIGN_ONLY` oder die `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Felder identifizieren, die im Beacon enthalten sind, und eine Liste von Konstruktorteilen, die alle Möglichkeiten angeben, wie der Compound Beacon die Felder zusammenstellen kann.

> **Note**
>
> Das AWS Database Encryption SDK unterstützt auch signierte Beacons, die vollständig aus Klartext und Feldern konfiguriert werden können. `SIGN_ONLY` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Signierte Beacons sind eine Art von zusammengesetzten Beacons, die signierte, aber nicht verschlüsselte Felder indexieren und komplexe Abfragen in diesen ausführen. Weitere Informationen finden Sie unter Signierte Beacons erstellen.

Hilfe zur Konfiguration von zusammengesetzten Beacons finden Sie unter Konfiguration von zusammengesetzten Beacons.

Die Art und Weise, wie Sie Ihren Compound Beacon konfigurieren, bestimmt, welche Arten von Abfragen er ausführen kann. Sie können beispielsweise einige verschlüsselte und signierte Teile optional machen, um mehr Flexibilität bei Ihren Abfragen zu gewährleisten. Weitere Informationen zu den Abfragetypen, die Compound Beacons ausführen können, finden Sie unterBeacons abfragen.

Leuchtfeuer planen

Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Beacons sind so konzipiert, dass sie in neuen, nicht aufgefüllten Datenbanken implementiert werden können. Jedes Beacon, das in einer vorhandenen Datenbank konfiguriert ist, ordnet nur neue Datensätze zu, die in die Datenbank geschrieben wurden. Beacons werden anhand des Klartextwerts eines Felds berechnet. Sobald das Feld verschlüsselt ist, kann das Beacon keine vorhandenen Daten zuordnen. Nachdem Sie neue Datensätze mit einem Beacon geschrieben haben, können Sie die Konfiguration des Beacons nicht mehr aktualisieren. Sie können jedoch neue Beacons für neue Felder hinzufügen, die Sie Ihrem Datensatz hinzufügen.

Um eine durchsuchbare Verschlüsselung zu implementieren, müssen Sie den AWS KMS hierarchischen Schlüsselbund verwenden, um die Datenschlüssel zu generieren, zu verschlüsseln und zu entschlüsseln, die zum Schutz Ihrer Datensätze verwendet werden. Weitere Informationen finden Sie unter Verwendung des hierarchischen Schlüsselbunds für durchsuchbare Verschlüsselung.

Bevor Sie Beacons für durchsuchbare Verschlüsselung konfigurieren können, müssen Sie Ihre Verschlüsselungsanforderungen, Datenbankzugriffsmuster und Ihr Bedrohungsmodell überprüfen, um die beste Lösung für Ihre Datenbank zu finden.

Der Typ des Beacons, den Sie konfigurieren, bestimmt die Art der Abfragen, die Sie ausführen können. Die Beacon-Länge, die Sie in der Standard-Beacon-Konfiguration angeben, bestimmt die erwartete Anzahl von Fehlalarmen, die für einen bestimmten Beacon erzeugt werden. Wir empfehlen dringend, die Arten von Abfragen, die Sie durchführen müssen, zu identifizieren und zu planen, bevor Sie Ihre Beacons konfigurieren. Sobald Sie ein Beacon verwendet haben, kann die Konfiguration nicht mehr aktualisiert werden.

Wir empfehlen dringend, dass Sie die folgenden Aufgaben überprüfen und ausführen, bevor Sie Beacons konfigurieren.

- Stellen Sie fest, ob Beacons für Ihren Datensatz geeignet sind
- Wählen Sie einen Beacon-Typ
- Wählen Sie eine Beacon-Länge

- Wählen Sie einen Beacon-Namen

Beachten Sie bei der Planung der durchsuchbaren Verschlüsselungslösung für Ihre Datenbank die folgenden Anforderungen an die Eindeutigkeit von Beacons.

- Jeder Standard-Beacon muss über eine eindeutige Beacon-Quelle verfügen

  Es können nicht mehrere Standard-Beacons aus demselben verschlüsselten oder virtuellen Feld erstellt werden.

  Ein einziger Standard-Beacon kann jedoch verwendet werden, um mehrere zusammengesetzte Beacons zu erstellen.
- Vermeiden Sie es, ein virtuelles Feld mit Quellfeldern zu erstellen, die sich mit vorhandenen Standard-Beacons überschneiden

  Die Konstruktion eines Standard-Beacons aus einem virtuellen Feld, das ein Quellfeld enthält, das zur Erstellung eines anderen Standard-Beacons verwendet wurde, kann die Sicherheit beider Beacons verringern.

  Weitere Informationen finden Sie unter Sicherheitsüberlegungen für virtuelle Felder.

Überlegungen zu Mehrmandantendatenbanken

Um Beacons abzufragen, die in einer Mehrmandantendatenbank konfiguriert sind, müssen Sie das Feld, das die Daten speichert, die dem Mandanten `branch-key-id` zugeordnet sind, der den Datensatz verschlüsselt hat, in Ihre Abfrage aufnehmen. Sie definieren dieses Feld, wenn Sie die Beacon-Schlüsselquelle definieren. Damit die Abfrage erfolgreich ist, muss der Wert in diesem Feld die entsprechenden Beacon-Schlüsselmaterialien identifizieren, die für die Neuberechnung des Beacons erforderlich sind.

Bevor Sie Ihre Beacons konfigurieren, müssen Sie entscheiden, wie Sie sie in Ihre Abfragen einbeziehen möchten. `branch-key-id` Weitere Informationen zu den verschiedenen Möglichkeiten, wie Sie die `branch-key-id` in Ihre Abfragen einbeziehen können, finden Sie unterAbfragen von Beacons in einer mandantenfähigen Datenbank.

Auswahl eines Beacon-Typs

Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Mit durchsuchbarer Verschlüsselung können Sie verschlüsselte Datensätze durchsuchen, indem Sie die Klartextwerte in einem verschlüsselten Feld einem Beacon zuordnen. Der Typ des Beacons, den Sie konfigurieren, bestimmt die Art der Abfragen, die Sie ausführen können.

Wir empfehlen dringend, die Arten von Abfragen, die Sie ausführen müssen, zu identifizieren und zu planen, bevor Sie Ihre Beacons konfigurieren. Nachdem Sie Ihre Beacons konfiguriert haben, müssen Sie für jedes Beacon einen sekundären Index konfigurieren, bevor Sie in den verschlüsselten Feldern suchen können. Weitere Informationen finden Sie unter Konfiguration sekundärer Indizes mit Beacons.

Beacons erstellen eine Zuordnung zwischen dem Klartextwert, der in ein Feld geschrieben wird, und dem verschlüsselten Wert, der tatsächlich in Ihrer Datenbank gespeichert ist. Sie können die Werte von zwei Standard-Beacons nicht vergleichen, selbst wenn sie denselben zugrunde liegenden Klartext enthalten. Die beiden Standard-Beacons erzeugen zwei verschiedene HMAC-Tags für dieselben Klartext-Werte. Daher können Standard-Beacons die folgenden Abfragen nicht ausführen.

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

Sie können die obigen Abfragen nur durchführen, wenn Sie die signierten Teile von zusammengesetzten Beacons vergleichen, mit Ausnahme des `CONTAINS` Operators, den Sie mit zusammengesetzten Beacons verwenden können, um den gesamten Wert eines verschlüsselten oder signierten Felds zu identifizieren, das der zusammengestellte Beacon enthält. Wenn Sie signierte Teile vergleichen, können Sie optional das Präfix eines verschlüsselten Teils angeben, nicht jedoch den verschlüsselten Wert eines Felds. Weitere Informationen zu den Abfragetypen, die Standard- und Verbundbeacons ausführen können, finden Sie unter Abfragen von Beacons.

Ziehen Sie bei der Überprüfung Ihrer Datenbankzugriffsmuster die folgenden durchsuchbaren Verschlüsselungslösungen in Betracht. In den folgenden Beispielen wird definiert, welcher Beacon konfiguriert werden muss, um unterschiedliche Verschlüsselungs- und Abfrageanforderungen zu erfüllen.

Standard-Beacons

Standard-Beacons können nur Gleichheitssuchen durchführen. Sie können Standard-Beacons verwenden, um die folgenden Abfragen durchzuführen.

Fragen Sie ein einzelnes verschlüsseltes Feld ab

Wenn Sie Datensätze identifizieren möchten, die einen bestimmten Wert für ein verschlüsseltes Feld enthalten, erstellen Sie einen Standard-Beacon.

Beispiele

Stellen Sie sich für das folgende Beispiel eine Datenbank mit dem Namen vor`UnitInspection`, die Inspektionsdaten für eine Produktionsanlage verfolgt. Jeder Datensatz in der Datenbank enthält Felder mit den Namen `work_id``inspection_date`,`inspector_id_last4`, und`unit`. Die vollständige Inspektor-ID ist eine Zahl zwischen 0 und 99.999.999. Um jedoch sicherzustellen, dass der Datensatz gleichmäßig verteilt ist, speichert The `inspector_id_last4` nur die letzten vier Ziffern der Inspektor-ID. Jedes Feld in der Datenbank wird durch den Primärschlüssel `work_id` identifiziert. Die `unit` Felder `inspector_id_last4` und sind `ENCRYPT_AND_SIGN` in den kryptografischen Aktionen markiert.

Das Folgende ist ein Beispiel für einen Klartext-Eintrag in der `UnitInspection` Datenbank.

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

Fragen Sie ein einzelnes verschlüsseltes Feld in einem Datensatz ab

> Wenn das `inspector_id_last4` Feld verschlüsselt werden muss, Sie es aber trotzdem nach exakten Übereinstimmungen abfragen müssen, erstellen Sie aus dem `inspector_id_last4` Feld ein Standard-Beacon. Verwenden Sie dann den Standard-Beacon, um einen sekundären

Index zu erstellen. Sie können diesen sekundären Index verwenden, um das verschlüsselte `inspector_id_last4` Feld abzufragen.

Hilfe zur Konfiguration von Standard-Beacons finden Sie unter Konfiguration von Standard-Beacons.

Fragen Sie ein virtuelles Feld ab

Ein virtuelles Feld ist ein konzeptionelles Feld, das aus einem oder mehreren Quellfeldern besteht. Wenn Sie Gleichheitssuchen für ein bestimmtes Segment eines verschlüsselten Felds oder Gleichheitssuchen für die Verkettung mehrerer Felder durchführen möchten, konstruieren Sie ein Standard-Beacon aus einem virtuellen Feld. Alle virtuellen Felder müssen mindestens ein verschlüsseltes Quellfeld enthalten.

Beispiele

In den folgenden Beispielen werden virtuelle Felder für die `Employees` Datenbank erstellt. Im Folgenden finden Sie ein Beispiel für einen Klartext-Datensatz in der `Employees` Datenbank.

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

Fragen Sie ein Segment eines verschlüsselten Felds ab

In diesem Beispiel ist das `SSN` Feld verschlüsselt.

Wenn Sie das `SSN` Feld mit den letzten vier Ziffern einer Sozialversicherungsnummer abfragen möchten, erstellen Sie ein virtuelles Feld, das das Segment identifiziert, das Sie abfragen möchten.

Ein virtuelles `Last4SSN` Feld, das aus erstellt wurde, `Suffix(4)` ermöglicht es Ihnen, Abfragen durchzuführen`Last4SSN=0000`. Verwenden Sie dieses virtuelle Feld, um einen Standard-Beacon

zu erstellen. Verwenden Sie dann den Standard-Beacon, um einen sekundären Index zu erstellen. Sie können diesen sekundären Index verwenden, um das virtuelle Feld abzufragen. Diese Abfrage gibt alle Datensätze zurück, `SSN` deren Wert mit den letzten vier von Ihnen angegebenen Ziffern endet.

Fragen Sie die Verkettung mehrerer Felder ab

> **Note**
>
> Das folgende Beispiel zeigt, welche Arten von Transformationen und Abfragen Sie mit einem virtuellen Feld ausführen können. In der Anwendung entsprechen die in diesem Beispiel verwendeten Beispielfelder möglicherweise nicht den Empfehlungen zur Verteilung und Korrelationseindeutigkeit für Beacons.

Wenn Sie Gleichheitssuchen für eine Verkettung von `FirstName` und `LastName` -Feldern durchführen möchten, können Sie ein virtuelles `NameTag` Feld erstellen, das aus dem ersten Buchstaben des Felds, gefolgt von dem `FirstName` Feld, gebildet wird, alles in Kleinbuchstaben. `LastName` Verwenden Sie dieses virtuelle Feld, um einen Standard-Beacon zu erstellen. Verwenden Sie dann den Standard-Beacon, um einen sekundären Index zu erstellen. Sie können diesen sekundären Index verwenden, um das virtuelle Feld abzufragen`NameTag=mjones`.

Mindestens eines der Quellfelder muss verschlüsselt sein. Entweder `FirstName` oder `LastName` könnte verschlüsselt werden, oder beide könnten verschlüsselt sein. Alle Klartext-Quellfelder müssen `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` in Ihren kryptografischen `SIGN_ONLY` Aktionen als oder gekennzeichnet sein.

Hilfe zur Konfiguration virtueller Felder und der Beacons, die sie verwenden, finden Sie unter Virtuelles Feld erstellen.

Zusammengesetzte Beacons

Zusammengesetzte Beacons erstellen einen Index aus wörtlichen Klartext-Zeichenketten und Standard-Beacons, um komplexe Datenbankoperationen durchzuführen. Sie können zusammengesetzte Beacons verwenden, um die folgenden Abfragen durchzuführen.

Fragen Sie eine Kombination verschlüsselter Felder in einem einzelnen Index ab

Wenn Sie eine Kombination von verschlüsselten Feldern in einem einzelnen Index abfragen müssen, erstellen Sie einen Verbundbeacon, der die einzelnen Standard-Beacons, die für jedes verschlüsselte Feld erstellt wurden, zu einem einzigen Index kombiniert.

Nachdem Sie den Compound Beacon konfiguriert haben, können Sie einen sekundären Index erstellen, der den Compound Beacon als Partitionsschlüssel für Abfragen mit exakter Übereinstimmung oder mit einem Sortierschlüssel für komplexere Abfragen angibt. Sekundäre Indizes, die den Compound Beacon als Sortierschlüssel angeben, können Abfragen mit exakter Übereinstimmung und individuellere komplexe Abfragen ausführen.

Beispiele

Stellen Sie sich für die folgenden Beispiele eine Datenbank mit dem Namen vor`UnitInspection`, die Inspektionsdaten für eine Produktionsanlage verfolgt. Jeder Datensatz in der Datenbank enthält Felder mit den Namen `work_id`,`inspection_date`,`inspector_id_last4`, und`unit`. Die vollständige Inspektor-ID ist eine Zahl zwischen 0 und 99.999.999. Um jedoch sicherzustellen, dass der Datensatz gleichmäßig verteilt ist, speichert The `inspector_id_last4` nur die letzten vier Ziffern der Inspektor-ID. Jedes Feld in der Datenbank wird durch den Primärschlüssel `work_id` identifiziert. Die `unit` Felder `inspector_id_last4` und sind `ENCRYPT_AND_SIGN` in den kryptografischen Aktionen markiert.

Das Folgende ist ein Beispiel für einen Klartext-Eintrag in der `UnitInspection` Datenbank.

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

Führen Sie Gleichheitssuchen in einer Kombination von verschlüsselten Feldern durch

Wenn Sie die `UnitInspection` Datenbank nach exakten Übereinstimmungen abfragen möchten`inspector_id_last4.unit`, erstellen Sie zunächst unterschiedliche Standard-Beacons für die `unit` Felder `inspector_id_last4` und. Erstellen Sie dann aus den beiden Standard-Beacons ein zusammengesetztes Beacon.

Nachdem Sie den Compound Beacon konfiguriert haben, erstellen Sie einen sekundären Index, der den Compound Beacon als Partitionsschlüssel angibt. Verwenden Sie diesen sekundären

Index, um nach exakten Übereinstimmungen zu suchen. `inspector_id_last4.unit` Sie könnten diesen Beacon beispielsweise abfragen, um eine Liste von Inspektionen zu finden, die ein Inspektor für eine bestimmte Einheit durchgeführt hat.

Führen Sie komplexe Abfragen für eine Kombination von verschlüsselten Feldern durch

Wenn Sie die `UnitInspection` Datenbank für `inspector_id_last4` und abfragen möchten`inspector_id_last4.unit`, erstellen Sie zunächst unterschiedliche Standardbeacons für die `unit` Felder `inspector_id_last4` und. Erstellen Sie dann ein zusammengesetztes Beacon aus den beiden Standard-Beacons.

Nachdem Sie den zusammengesetzten Beacon konfiguriert haben, erstellen Sie einen sekundären Index, der den zusammengesetzten Beacon als Sortierschlüssel angibt. Verwenden Sie diesen sekundären Index, um die `UnitInspection` Datenbank nach Einträgen abzufragen, die mit einem bestimmten Inspektor beginnen, oder fragen Sie die Datenbank nach einer Liste aller Einheiten innerhalb eines bestimmten Einheiten-ID-Bereichs ab, die von einem bestimmten Inspektor geprüft wurden. Sie können auch nach exakten Übereinstimmungen suchen für`inspector_id_last4.unit`.

Hilfe zur Konfiguration von zusammengesetzten Beacons finden Sie unter Konfiguration von zusammengesetzten Beacons.

Fragen Sie eine Kombination aus verschlüsselten Feldern und Klartextfeldern in einem einzigen Index ab

Wenn Sie eine Kombination aus verschlüsselten Feldern und Klartextfeldern in einem einzigen Index abfragen müssen, erstellen Sie einen zusammengesetzten Beacon, der einzelne Standard-Beacons und Klartextfelder zu einem einzigen Index kombiniert. Die Klartextfelder, die zur Erstellung des Verbund-Beacons verwendet werden, müssen markiert `SIGN_ONLY` oder in Ihren kryptografischen Aktionen enthalten sein. `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`

Nachdem Sie den Compound Beacon konfiguriert haben, können Sie einen sekundären Index erstellen, der den Compound Beacon als Partitionsschlüssel für exakt passende Abfragen oder mit einem Sortierschlüssel für komplexere Abfragen angibt. Sekundäre Indizes, die den Compound Beacon als Sortierschlüssel angeben, können Abfragen mit exakter Übereinstimmung und individuellere komplexe Abfragen ausführen.

Beispiele

Stellen Sie sich für die folgenden Beispiele eine Datenbank mit dem Namen vor`UnitInspection`, die Inspektionsdaten für eine Produktionsanlage verfolgt. Jeder Datensatz in der Datenbank enthält Felder mit den Namen `work_id`,`inspection_date`,`inspector_id_last4`, und`unit`. Die vollständige Inspektor-ID ist eine Zahl zwischen 0 und 99.999.999. Um jedoch sicherzustellen, dass der Datensatz gleichmäßig verteilt ist, speichert The `inspector_id_last4` nur die letzten vier Ziffern der Inspektor-ID. Jedes Feld in der Datenbank wird durch den Primärschlüssel `work_id` identifiziert. Die `unit` Felder `inspector_id_last4` und sind `ENCRYPT_AND_SIGN` in den kryptografischen Aktionen markiert.

Das Folgende ist ein Beispiel für einen Klartext-Eintrag in der `UnitInspection` Datenbank.

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

Führen Sie Gleichheitssuchen in einer Kombination von Feldern durch

Wenn Sie die `UnitInspection` Datenbank nach Inspektionen abfragen möchten, die von einem bestimmten Inspektor an einem bestimmten Datum durchgeführt wurden, erstellen Sie zunächst einen Standard-Beacon für das `inspector_id_last4` Feld. Das `inspector_id_last4` Feld ist `ENCRYPT_AND_SIGN` in den kryptografischen Aktionen markiert. Alle verschlüsselten Teile benötigen einen eigenen Standard-Beacon. Das `inspection_date` Feld ist markiert `SIGN_ONLY` und benötigt keinen Standard-Beacon. Erstellen Sie als Nächstes ein Verbundsignal aus dem `inspection_date` Feld und dem `inspector_id_last4` Standardbeacon.

Nachdem Sie den Compound Beacon konfiguriert haben, erstellen Sie einen sekundären Index, der den Compound Beacon als Partitionsschlüssel angibt. Verwenden Sie diesen sekundären Index, um die Datenbanken nach Datensätzen abzufragen, die exakt mit einem bestimmten Inspektor und einem bestimmten Inspektionsdatum übereinstimmen. Beispielsweise können Sie die Datenbank nach einer Liste aller Inspektionen abfragen, die der Inspektor, dessen ID auf `8744` endet, an einem bestimmten Datum durchgeführt hat.

Führen Sie komplexe Abfragen für eine Kombination von Feldern durch

Wenn Sie die Datenbank nach Inspektionen abfragen möchten, die innerhalb eines bestimmten `inspection_date` Bereichs durchgeführt wurden, oder die Datenbank nach Inspektionen abfragen möchten, die für einen bestimmten `inspection_date` eingeschränkten Wert von `inspector_id_last4` oder durchgeführt wurden`inspector_id_last4.unit`, erstellen Sie zunächst separate Standard-Beacons für die Felder `inspector_id_last4` und`unit`. Erstellen Sie dann einen Verbundbeacon aus dem `inspection_date` Klartextfeld und den beiden Standard-Beacons.

Nachdem Sie den zusammengesetzten Beacon konfiguriert haben, erstellen Sie einen sekundären Index, der den zusammengesetzten Beacon als Sortierschlüssel angibt. Verwenden Sie diesen sekundären Index, um Abfragen für Inspektionen durchzuführen, die an bestimmten Terminen von einem bestimmten Inspektor durchgeführt wurden. Sie können beispielsweise die Datenbank nach einer Liste aller am selben Tag inspizierten Einheiten abfragen. Oder Sie können die Datenbank nach einer Liste aller Inspektionen abfragen, die an einer bestimmten Einheit zwischen einem bestimmten Zeitraum von Inspektionsterminen durchgeführt wurden.

Hilfe zur Konfiguration von zusammengesetzten Beacons finden Sie unter Konfiguration von zusammengesetzten Beacons.

Wahl einer Beacon-Länge

> Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Wenn Sie einen neuen Wert in ein verschlüsseltes Feld schreiben, das für durchsuchbare Verschlüsselung konfiguriert ist, berechnet das AWS Database Encryption SDK einen HMAC-Wert über dem Klartext-Wert. Bei dieser HMAC-Ausgabe handelt es sich um eine 1:1 - Übereinstimmung mit dem Klartextwert dieses Felds. Die HMAC-Ausgabe wird gekürzt, sodass mehrere unterschiedliche Klartextwerte demselben gekürzten HMAC-Tag zugeordnet werden. Diese Kollisionen oder Fehlalarme schränken die Fähigkeit eines nicht autorisierten Benutzers ein, charakteristische Informationen über den Klartext-Wert zu identifizieren.

Die durchschnittliche Anzahl der für jeden Beacon generierten Fehlalarme wird durch die Länge des Beacons bestimmt, die nach der Kürzung noch übrig ist. Sie müssen die Beacon-Länge nur

definieren, wenn Sie Standard-Beacons konfigurieren. Verbund-Beacons verwenden die Beacon-Längen der Standard-Beacons, aus denen sie aufgebaut sind.

Der Beacon ändert den verschlüsselten Zustand des Feldes nicht. Bei der Verwendung von Beacons besteht jedoch ein inhärenter Kompromiss zwischen der Effizienz Ihrer Abfragen und der Menge an Informationen, die über die Verteilung Ihrer Daten preisgegeben werden.

Das Ziel der durchsuchbaren Verschlüsselung besteht darin, die mit clientseitig verschlüsselten Datenbanken verbundenen Leistungskosten zu reduzieren, indem Beacons zur Durchführung von Abfragen verschlüsselter Daten verwendet werden. Beacons werden zusammen mit den verschlüsselten Feldern gespeichert, aus denen sie berechnet werden. Das bedeutet, dass sie aussagekräftige Informationen über die Verteilung Ihres Datensatzes preisgeben können. In extremen Fällen kann ein nicht autorisierter Benutzer die Informationen über Ihre Verteilung analysieren und anhand dieser Informationen den Klartextwert eines Felds ermitteln. Die Wahl der richtigen Beacon-Länge kann dazu beitragen, diese Risiken zu minimieren und die Vertraulichkeit Ihrer Verteilung zu wahren.

Überprüfen Sie Ihr Bedrohungsmodell, um das Sicherheitsniveau zu ermitteln, das Sie benötigen. Je mehr Personen beispielsweise Zugriff auf Ihre Datenbank haben, aber keinen Zugriff auf die Klartextdaten haben sollten, desto mehr möchten Sie möglicherweise die Vertraulichkeit Ihrer Datensatzverteilung schützen. Um die Vertraulichkeit zu erhöhen, muss ein Beacon mehr Fehlalarme generieren. Eine erhöhte Vertraulichkeit führt zu einer verringerten Abfrageleistung.

Sicherheit versus Leistung

- Eine zu lange Beacon-Länge erzeugt zu wenige Fehlalarme und kann aussagekräftige Informationen über die Verteilung Ihres Datensatzes preisgeben.
- Eine zu kurze Beacon-Länge erzeugt zu viele Fehlalarme und erhöht die Leistungseinbußen bei Abfragen, da dafür ein umfassenderer Scan der Datenbank erforderlich ist.

Bei der Bestimmung der geeigneten Beacon-Länge für Ihre Lösung müssen Sie eine Länge wählen, die die Sicherheit Ihrer Daten angemessen gewährleistet, ohne die Leistung Ihrer Abfragen mehr als unbedingt erforderlich zu beeinträchtigen. Der Sicherheitsgrad, den ein Beacon gewährleistet, hängt von der Verteilung Ihres Datensatzes und der Korrelation der Felder ab, aus denen Ihre Beacons aufgebaut sind. In den folgenden Themen wird davon ausgegangen, dass Ihre Beacons gleichmäßig verteilt sind und keine korrelierten Daten enthalten.

Themen

Berechnung der Beacon-Länge

Die Länge des Beacons wird in Bits definiert und bezieht sich auf die Anzahl der Bits des HMAC-Tags, die nach der Kürzung beibehalten werden. Die empfohlene Beacon-Länge hängt von der Verteilung des Datensatzes, dem Vorhandensein korrelierter Werte und Ihren spezifischen Sicherheits- und Leistungsanforderungen ab. Wenn Ihr Datensatz gleichmäßig verteilt ist, können Sie die folgenden Gleichungen und Verfahren verwenden, um die beste Beacon-Länge für Ihre Implementierung zu ermitteln. Mit diesen Gleichungen wird nur die durchschnittliche Anzahl falsch positiver Ergebnisse geschätzt, die der Beacon erzeugt. Sie garantieren nicht, dass jeder einzelne Wert in Ihrem Datensatz eine bestimmte Anzahl falsch positiver Ergebnisse erzeugt.

> **Note**
>
> Die Wirksamkeit dieser Gleichungen hängt von der Verteilung Ihres Datensatzes ab. Wenn Ihr Datensatz nicht gleichmäßig verteilt ist, finden Sie weitere Informationen unterSind Beacons das Richtige für meinen Datensatz?.
> Generell gilt: Je weiter Ihr Datensatz von einer gleichmäßigen Verteilung entfernt ist, desto mehr müssen Sie Ihre Beacon-Länge verkürzen.

1.

   Schätzen Sie die Population

   Bei der Grundgesamtheit handelt es sich um die erwartete Anzahl von Einzelwerten in dem Feld, aus dem Ihr Standard-Beacon erstellt wurde, nicht um die erwartete Gesamtanzahl der im Feld gespeicherten Werte. Stellen Sie sich zum Beispiel ein verschlüsseltes `Room` Feld vor, das den Ort von Mitarbeiterversammlungen identifiziert. Es wird erwartet, dass das `Room` Feld insgesamt 100.000 Werte speichert, aber es gibt nur 50 verschiedene Räume, die Mitarbeiter für Besprechungen reservieren können. Das bedeutet, dass die Population 50 beträgt, weil es nur 50 mögliche Einzelwerte gibt, die in dem `Room` Feld gespeichert werden können.

> **Note**
>
> Wenn Ihr Standard-Beacon aus einem virtuellen Feld besteht, entspricht die zur Berechnung der Beacon-Länge verwendete Population der Anzahl der eindeutigen Kombinationen, die durch das virtuelle Feld erzeugt werden.

Achten Sie bei der Schätzung Ihrer Population darauf, das prognostizierte Wachstum des Datensatzes zu berücksichtigen. Nachdem Sie mit dem Beacon neue Datensätze geschrieben haben, können Sie die Länge des Beacons nicht mehr aktualisieren. Überprüfen Sie Ihr Bedrohungsmodell und alle vorhandenen Datenbanklösungen, um eine Schätzung der Anzahl der Einzelwerte zu erstellen, die dieses Feld voraussichtlich in den nächsten fünf Jahren speichern wird.

Ihre Population muss nicht genau sein. Identifizieren Sie zunächst die Anzahl der Einzelwerte in Ihrer aktuellen Datenbank, oder schätzen Sie die Anzahl der Einzelwerte, die Sie voraussichtlich im ersten Jahr speichern werden. Verwenden Sie als Nächstes die folgenden Fragen, um das prognostizierte Wachstum der Einzelwerte in den nächsten fünf Jahren zu ermitteln.

- Erwarten Sie, dass sich die Einzelwerte mit 10 multiplizieren werden?
- Erwarten Sie, dass sich die Einzelwerte mit 100 multiplizieren?
- Erwarten Sie, dass sich die Einzelwerte mit 1000 multiplizieren?

Der Unterschied zwischen 50.000 und 60.000 Einzelwerten ist nicht signifikant und beide führen zu derselben empfohlenen Beacon-Länge. Der Unterschied zwischen 50.000 und 500.000 Einzelwerten wirkt sich jedoch erheblich auf die empfohlene Beacon-Länge aus.

Erwägen Sie, öffentliche Daten zur Häufigkeit gängiger Datentypen wie Postleitzahlen oder Nachnamen zu überprüfen. In den Vereinigte Staaten gibt es beispielsweise 41.707 Postleitzahlen. Die von Ihnen verwendete Population sollte proportional zu Ihrer eigenen Datenbank sein. Wenn das `ZIPCode` Feld in Ihrer Datenbank Daten aus den gesamten Vereinigte Staaten enthält, können Sie Ihre Bevölkerung als 41.707 definieren, auch wenn das `ZIPCode` Feld derzeit keine 41.707 Einzelwerte enthält. Wenn das `ZIPCode` Feld in Ihrer Datenbank nur Daten aus einem einzigen Bundesstaat enthält und immer nur Daten aus einem einzigen Bundesstaat enthalten wird, können Sie Ihre Bevölkerung als die Gesamtzahl der Postleitzahlen in diesem Bundesstaat statt als 41.704 definieren.

2. Berechnen Sie den empfohlenen Bereich für die erwartete Anzahl von Kollisionen

   Um die geeignete Beacon-Länge für ein bestimmtes Feld zu bestimmen, müssen Sie zunächst einen geeigneten Bereich für die erwartete Anzahl von Kollisionen ermitteln. Die erwartete Anzahl von Kollisionen stellt die durchschnittliche erwartete Anzahl eindeutiger Klartextwerte dar, die einem bestimmten HMAC-Tag zugeordnet sind. Die erwartete Anzahl falsch positiver Ergebnisse für einen eindeutigen Klartextwert liegt um eins unter der erwarteten Anzahl von Kollisionen.

   Wir empfehlen, dass die erwartete Anzahl von Kollisionen größer oder gleich zwei und kleiner als die Quadratwurzel Ihrer Grundgesamtheit ist. Die folgenden Gleichungen funktionieren nur, wenn Ihre Grundgesamtheit 16 oder mehr Einzelwerte hat.

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

   Wenn die Anzahl der Kollisionen weniger als zwei beträgt, erzeugt der Beacon zu wenige Fehlalarme. Wir empfehlen zwei als Mindestanzahl erwarteter Kollisionen, da dies bedeutet, dass im Durchschnitt jeder Einzelwert im Feld mindestens ein falsches Positiv generiert, wenn er einem anderen Einzelwert zugeordnet wird.

3. Berechnen Sie den empfohlenen Bereich für die Länge der Beacons

   Nachdem Sie die minimale und maximale Anzahl erwarteter Kollisionen ermittelt haben, verwenden Sie die folgende Gleichung, um einen Bereich geeigneter Beacon-Längen zu ermitteln.

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

   Ermitteln Sie zunächst die Beacon-Länge, bei der die Anzahl der erwarteten Kollisionen gleich zwei ist (die empfohlene Mindestanzahl erwarteter Kollisionen).

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

   Berechne dann nach der Länge des Beacons, wobei die erwartete Anzahl von Kollisionen der Quadratwurzel deiner Grundgesamtheit entspricht (der empfohlenen maximalen Anzahl erwarteter Kollisionen).

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

Wir empfehlen, die mit dieser Gleichung erzeugte Ausgabe auf die kürzere Beacon-Länge abzurunden. Ergibt die Gleichung beispielsweise eine Beacon-Länge von 15,6, empfehlen wir, diesen Wert auf 15 Bit abzurunden, anstatt ihn auf 16 Bit aufzurunden.

4. Wählen Sie eine Beacon-Länge

   Diese Gleichungen geben nur einen empfohlenen Bereich von Beacon-Längen für Ihr Fachgebiet an. Wir empfehlen, eine kürzere Beacon-Länge zu verwenden, um die Sicherheit Ihres Datensatzes zu gewährleisten, wann immer dies möglich ist. Die Länge des Beacons, das Sie tatsächlich verwenden, hängt jedoch von Ihrem Bedrohungsmodell ab. Berücksichtigen Sie bei der Überprüfung Ihres Bedrohungsmodells Ihre Leistungsanforderungen, um die beste Beacon-Länge für Ihr Einsatzgebiet zu ermitteln.

   Die Verwendung einer kürzeren Beacon-Länge verringert die Abfrageleistung, während die Verwendung einer längeren Beacon-Länge die Sicherheit verringert. Wenn Ihr Datensatz ungleichmäßig verteilt ist oder Sie unterschiedliche Beacons aus korrelierten Feldern erstellen, müssen Sie im Allgemeinen kürzere Beacon-Längen verwenden, um die Menge an Informationen zu minimieren, die über die Verteilung Ihrer Datensätze preisgegeben werden.

   Wenn Sie Ihr Bedrohungsmodell überprüfen und zu dem Schluss kommen, dass alle offengelegten Unterscheidungsinformationen über die Verteilung eines Feldes keine Gefahr für Ihre allgemeine Sicherheit darstellen, können Sie eine Beacon-Länge verwenden, die länger ist als der von Ihnen berechnete empfohlene Bereich. Wenn Sie beispielsweise den empfohlenen Bereich der Beacon-Längen für ein Feld mit 9—16 Bit berechnet haben, könnten Sie sich für eine Beacon-Länge von 24 Bit entscheiden, um Leistungseinbußen zu vermeiden.

   Wählen Sie Ihre Beacon-Länge sorgfältig aus. Nachdem Sie mit dem Beacon neue Datensätze geschrieben haben, können Sie die Länge des Beacons nicht mehr aktualisieren.

Beispiel

Stellen Sie sich eine Datenbank vor, die das `unit` Feld als `ENCRYPT_AND_SIGN` in den kryptografischen Aktionen markiert hat. Um einen Standard-Beacon für das `unit` Feld zu konfigurieren, müssen wir die erwartete Anzahl von Fehlalarmen und die Länge des Beacons für das Feld ermitteln. `unit`

1. Schätzen Sie die Bevölkerung

Nach der Überprüfung unseres Bedrohungsmodells und unserer aktuellen Datenbanklösung gehen wir davon aus, dass das `unit` Feld irgendwann 100.000 eindeutige Werte haben wird.

Das bedeutet, dass Bevölkerung = 100.000 ist.

2. Berechnet den empfohlenen Bereich für die erwartete Anzahl von Kollisionen.

   In diesem Beispiel sollte die erwartete Anzahl von Kollisionen zwischen 2 und 316 liegen.

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

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

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

3. Berechnen Sie den empfohlenen Bereich für die Länge des Beacons.

   In diesem Beispiel sollte die Länge des Beacons zwischen 9 und 16 Bit liegen.

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

   a. Berechnen Sie die Länge des Beacons, bei der die erwartete Anzahl von Kollisionen dem in Schritt 2 ermittelten Minimum entspricht.

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

      Länge des Beacons = 15,6 oder 15 Bit

   b. Berechnen Sie die Länge des Beacons, wobei die erwartete Anzahl von Kollisionen dem in Schritt 2 ermittelten Maximum entspricht.

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

      Länge des Beacons = 8,3 oder 8 Bit

4. Ermitteln Sie die Beacon-Länge, die Ihren Sicherheits- und Leistungsanforderungen entspricht.

   Für jedes Bit unter 15 verdoppeln sich die Kosten für Leistung und Sicherheit.

   - 16 Bit

    - Im Durchschnitt wird jeder Einzelwert 1,5 anderen Einheiten zugeordnet.
    - Sicherheit: Bei zwei Datensätzen mit demselben gekürzten HMAC-Tag besteht eine Wahrscheinlichkeit von 66%, dass sie denselben Klartextwert haben.
    - Leistung: Eine Abfrage ruft 15 Datensätze für jeweils 10 Datensätze ab, die Sie tatsächlich angefordert haben.
- 14 Bit
    - Im Durchschnitt wird jeder Einzelwert 6,1 anderen Einheiten zugeordnet.
    - Sicherheit: Bei zwei Datensätzen mit demselben gekürzten HMAC-Tag besteht eine Wahrscheinlichkeit von 33%, dass sie denselben Klartextwert haben.
    - Leistung: Eine Abfrage ruft 30 Datensätze für jeweils 10 Datensätze ab, die Sie tatsächlich angefordert haben.

Einen Beacon-Namen wählen

> Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Jeder Beacon wird durch einen eindeutigen Beacon-Namen identifiziert. Sobald ein Beacon konfiguriert ist, ist der Beacon-Name der Name, den Sie bei der Abfrage eines verschlüsselten Felds verwenden. Ein Beacon-Name kann derselbe Name wie ein verschlüsseltes Feld oder virtuelles Feld sein, er kann jedoch nicht derselbe Name wie ein unverschlüsseltes Feld sein. Zwei verschiedene Beacons können nicht denselben Beacon-Namen haben.

Beispiele, die zeigen, wie Beacons benannt und konfiguriert werden, finden Sie unter Konfiguration von Beacons.

Benennen von Standard-Beacons

Bei der Benennung von Standardbeacons empfehlen wir dringend, dass Ihr Beacon-Name nach Möglichkeit in die Beacon-Quelle aufgelöst wird. Das bedeutet, dass der Beacon-Name und der Name des verschlüsselten oder virtuellen Feldes, aus dem Ihr Standard-Beacon aufgebaut ist,

identisch sind. Wenn Sie beispielsweise einen Standard-Beacon für ein verschlüsseltes Feld mit dem Namen erstellen`LastName`, sollte Ihr Beacon-Name ebenfalls lauten. `LastName`

Wenn Ihr Beacon-Name mit der Beacon-Quelle identisch ist, können Sie die Beacon-Quelle aus Ihrer Konfiguration weglassen und das AWS Database Encryption SDK verwendet den Beacon-Namen automatisch als Beacon-Quelle.

Konfiguration von Beacons

> Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Es gibt zwei Arten von Beacons, die durchsuchbare Verschlüsselung unterstützen. Standard-Beacons führen Gleichheitssuchen durch. Sie sind der einfachste Weg, eine durchsuchbare Verschlüsselung in Ihrer Datenbank zu implementieren. Compound Beacons kombinieren wörtliche Klartext-Zeichenketten und Standard-Beacons, um komplexere Abfragen durchzuführen.

Beacons sind so konzipiert, dass sie in neuen, nicht aufgefüllten Datenbanken implementiert werden können. Jedes in einer vorhandenen Datenbank konfigurierte Beacon ordnet nur neue Datensätze zu, die in die Datenbank geschrieben wurden. Beacons werden anhand des Klartextwerts eines Felds berechnet. Sobald das Feld verschlüsselt ist, kann das Beacon keine vorhandenen Daten zuordnen. Nachdem Sie neue Datensätze mit einem Beacon geschrieben haben, können Sie die Konfiguration des Beacons nicht mehr aktualisieren. Sie können jedoch neue Beacons für neue Felder hinzufügen, die Sie Ihrem Datensatz hinzufügen.

Nachdem Sie Ihre Zugriffsmuster bestimmt haben, sollte die Konfiguration von Beacons der zweite Schritt in Ihrer Datenbankimplementierung sein. Nachdem Sie alle Ihre Beacons konfiguriert haben, müssen Sie einen AWS KMS hierarchischen Schlüsselbund erstellen, die Beacon-Version definieren, einen sekundären Index für jedes Beacon konfigurieren, Ihre kryptografischen Aktionen definieren und Ihre Datenbank und den Database Encryption SDK-Client konfigurieren. AWS Weitere Informationen finden Sie unter Verwenden von Beacons.

Um die Definition der Beacon-Version zu vereinfachen, empfehlen wir, Listen für Standard- und Verbund-Beacons zu erstellen. Fügen Sie jedes Beacon, das Sie erstellen, bei der Konfiguration der jeweiligen Standard- oder Verbund-Beacons hinzu.

Themen

Konfiguration von Standard-Beacons

Standard-Beacons sind die einfachste Methode, eine durchsuchbare Verschlüsselung in Ihrer Datenbank zu implementieren. Sie können nur Gleichheitssuchen für ein einzelnes verschlüsseltes oder virtuelles Feld durchführen.

Beispiel für eine Konfigurationssyntax

Java

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

C# / .NET

```
var standardBeaconList = new List<StandardBeacon>();
StandardBeacon exampleStandardBeacon = new StandardBeacon
  {
    Name = "beaconName",
    Length = 10
  };
standardBeaconList.Add(exampleStandardBeacon);
```

Rust

```
let standard_beacon_list = vec![

 StandardBeacon::builder().name("beacon_name").length(beacon_length_in_bits).build()?,
```

Um ein Standard-Beacon zu konfigurieren, geben Sie die folgenden Werte an.

Name des Beacons

Der Name, den Sie bei der Abfrage eines verschlüsselten Felds verwenden.

Ein Beacon-Name kann derselbe Name wie ein verschlüsseltes Feld oder virtuelles Feld sein, er kann jedoch nicht derselbe Name wie ein unverschlüsseltes Feld sein. Wir empfehlen dringend, wann immer möglich den Namen des verschlüsselten Felds oder virtuellen Feldes zu verwenden, aus dem Ihr Standard-Beacon erstellt wird. Zwei verschiedene Beacons können nicht denselben Beacon-Namen haben. Hilfe bei der Bestimmung des besten Beacon-Namens für Ihre Implementierung finden Sie unter Auswahl eines Beacon-Namens.

Länge des Beacons

Die Anzahl der Bits des Beacon-Hashwerts, die nach der Kürzung beibehalten werden.

Die Länge des Beacons bestimmt die durchschnittliche Anzahl von Fehlalarmen, die von einem bestimmten Beacon erzeugt werden. Weitere Informationen und Hilfe bei der Bestimmung der geeigneten Beacon-Länge für Ihre Implementierung finden Sie unter Bestimmung der Beacon-Länge.

Beacon-Quelle (optional)

Das Feld, aus dem ein Standard-Beacon erstellt wird.

Die Beacon-Quelle muss ein Feldname oder ein Index sein, der auf den Wert eines verschachtelten Felds verweist. Wenn Ihr Beacon-Name mit der Beacon-Quelle identisch ist, können Sie die Beacon-Quelle aus Ihrer Konfiguration weglassen und das AWS Database Encryption SDK verwendet den Beacon-Namen automatisch als Beacon-Quelle.

Ein virtuelles Feld erstellen

Um ein virtuelles Feld zu erstellen, müssen Sie einen Namen für das virtuelle Feld und eine Liste der Quellfelder angeben. Die Reihenfolge, in der Sie der virtuellen Komponentenliste Quellfelder hinzufügen, bestimmt die Reihenfolge, in der sie zum Aufbau des virtuellen Felds verkettet werden. Im folgenden Beispiel werden zwei Quellfelder in ihrer Gesamtheit verkettet, um ein virtuelles Feld zu erstellen.

> **Note**
>
> Wir empfehlen, zu überprüfen, ob Ihre virtuellen Felder das erwartete Ergebnis liefern, bevor Sie Ihre Datenbank auffüllen. Weitere Informationen finden Sie unter Beacon-Ausgaben testen.

Java

Sehen Sie sich das vollständige Codebeispiel an: .java VirtualBeaconSearchableEncryptionExample

```
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);
```

C# / .NET

Sehen Sie sich das vollständige Codebeispiel an: .cs VirtualBeaconSearchableEncryptionExample

```
var virtualPartList = new List<VirtualPart> { sourceField1, sourceField2 };

var virtualFieldName = new VirtualField
{
    Name = "virtualFieldName",
    Parts = virtualPartList
};

var virtualFieldList = new List<VirtualField> { virtualFieldName };
```

Rust

Sehen Sie sich das vollständige Codebeispiel an: virtual_beacon_searchable_encryption.rs

```
let virtual_part_list = vec![source_field_one, source_field_two];

let state_and_has_test_result_field = VirtualField::builder()
    .name("virtual_field_name")
    .parts(virtual_part_list)
    .build()?;

let virtual_field_list = vec![virtual_field_name];
```

Um ein virtuelles Feld mit einem bestimmten Segment eines Quellfeldes zu erstellen, müssen Sie diese Transformation definieren, bevor Sie das Quellfeld zu Ihrer virtuellen Teileliste hinzufügen.

Sicherheitsüberlegungen für virtuelle Felder

Beacons ändern den verschlüsselten Zustand des Feldes nicht. Bei der Verwendung von Beacons besteht jedoch ein inhärenter Kompromiss zwischen der Effizienz Ihrer Abfragen und der Menge an Informationen, die über die Verteilung Ihrer Daten preisgegeben werden. Die Art und Weise, wie Sie Ihr Beacon konfigurieren, bestimmt das Sicherheitsniveau, das durch dieses Beacon gewährleistet wird.

Vermeiden Sie es, ein virtuelles Feld mit Quellfeldern zu erstellen, die sich mit vorhandenen Standard-Beacons überschneiden. Das Erstellen virtueller Felder, die ein Quellfeld enthalten, das bereits zur Erstellung eines Standard-Beacons verwendet wurde, kann das Sicherheitsniveau für beide Beacons verringern. Das Ausmaß, in dem die Sicherheit reduziert wird, hängt von der Entropiestufe ab, die durch die zusätzlichen Quellfelder hinzugefügt wird. Der Grad der Entropie wird durch die Verteilung der Einzelwerte im zusätzlichen Quellfeld und die Anzahl der Bits bestimmt, die das zusätzliche Quellfeld zur Gesamtgröße des virtuellen Feldes beiträgt.

Sie können anhand der Population und der Beacon-Länge ermitteln, ob die Quellfelder für ein virtuelles Feld die Sicherheit Ihres Datensatzes gewährleisten. Die Population ist die erwartete Anzahl von Einzelwerten in einem Feld. Ihre Population muss nicht exakt sein. Hilfe zur Schätzung der Grundgesamtheit eines Felds finden Sie unter Grundgesamtheit schätzen.

Betrachten Sie das folgende Beispiel, wenn Sie die Sicherheit Ihrer virtuellen Felder überprüfen.

- Beacon1 besteht aus. `FieldA` `FieldA`hat eine Population von mehr als $2^{\text{(Beacon1-Länge)}}$.
- Beacon2 wird aus`VirtualField`, was aus,, und aufgebaut ist`FieldA`, `FieldB` konstruiert. `FieldC` `FieldD` Zusammen **`FieldD`** haben, **`FieldBFieldC`**, und eine Population von mehr als 2 N

Beacon2 gewährleistet die Sicherheit von Beacon1 und Beacon2, wenn die folgenden Aussagen zutreffen:

```
N ≥ (Beacon1 length)/2
```

und

```
N ≥ (Beacon2 length)/2
```

Definition von Beacon-Stilen

Standard-Beacons können verwendet werden, um Gleichheitssuchen für ein verschlüsseltes oder virtuelles Feld durchzuführen. Sie können auch verwendet werden, um zusammengesetzte Beacons für komplexere Datenbankoperationen zu erstellen. Um Ihnen bei der Organisation und Verwaltung von Standard-Beacons zu helfen, bietet das AWS Database Encryption SDK die folgenden optionalen Beacon-Stile, die den Verwendungszweck eines Standard-Beacons definieren.

> **Note**
>
> Um Beacon-Stile zu definieren, müssen Sie Version 3.2 oder höher des Database Encryption SDK verwenden. AWS Stellen Sie die neue Version für alle Leser bereit, bevor Sie Beacon-Styles zu Ihren Beacon-Konfigurationen hinzufügen.

PartOnly

Ein Standard-Beacon, das als definiert ist, `PartOnly` kann nur zur Definition eines verschlüsselten Teils eines zusammengesetzten Beacons verwendet werden. Sie können ein `PartOnly` Standard-Beacon nicht direkt abfragen.

Java

```
List<StandardBeacon> standardBeaconList = new ArrayList<>();
StandardBeacon exampleStandardBeacon = StandardBeacon.builder()
    .name("beaconName")
    .length(beaconLengthInBits)
    .style(
        BeaconStyle.builder()
           .partOnly(PartOnly.builder().build())
```

```
            .build()
        )
        .build();
standardBeaconList.add(exampleStandardBeacon);
```

C#/.NET

```
new StandardBeacon
{
    Name = "beaconName",
    Length = beaconLengthInBits,
    Style = new BeaconStyle
    {
        PartOnly = new PartOnly()
    }
}
```

Rust

```
StandardBeacon::builder()
    .name("beacon_name")
    .length(beacon_length_in_bits)
    .style(BeaconStyle::PartOnly(PartOnly::builder().build()?))
    .build()?
```

Shared

Standardmäßig generiert jedes Standard-Beacon einen eindeutigen HMAC-Schlüssel für die Beacon-Berechnung. Daher können Sie keine Gleichheitssuche in den verschlüsselten Feldern von zwei separaten Standard-Beacons durchführen. Ein als definierter Standard-Beacon `Shared` verwendet für seine Berechnungen den HMAC-Schlüssel eines anderen Standard-Beacons.

Wenn Sie beispielsweise Felder mit `beacon1` Feldern vergleichen müssen, definieren Sie es `beacon2` `beacon2` als `Shared` Beacon, das den HMAC-Schlüssel von für seine Berechnungen verwendet. `beacon1`

> **Note**
>
> Berücksichtigen Sie Ihre Sicherheits- und Leistungsanforderungen, bevor Sie Beacons konfigurieren. `Shared` `Shared`Beacons können die Menge an statistischen

Informationen, die über die Verteilung Ihres Datensatzes identifiziert werden können, erhöhen. Sie könnten beispielsweise Aufschluss darüber geben, welche gemeinsam genutzten Felder denselben Klartextwert enthalten.

Java

```
List<StandardBeacon> standardBeaconList = new ArrayList<>();
StandardBeacon exampleStandardBeacon = StandardBeacon.builder()
    .name("beacon2")
    .length(beaconLengthInBits)
    .style(
        BeaconStyle.builder()
           .shared(Shared.builder().other("beacon1").build())
        .build()
    )
    .build();
standardBeaconList.add(exampleStandardBeacon);
```

C#/.NET

```
new StandardBeacon
{
    Name = "beacon2",
    Length = beaconLengthInBits,
    Style = new BeaconStyle
    {
        Shared = new Shared { Other = "beacon1" }
    }
}
```

Rust

```
StandardBeacon::builder()
    .name("beacon2")
    .length(beacon_length_in_bits)
    .style(BeaconStyle::Shared(
       Shared::builder().other("beacon1").build()?,
    ))
    .build()?
```

AsSet

Wenn es sich bei einem Feldwert um einen Satz handelt, berechnet das AWS Database Encryption SDK standardmäßig einen einzelnen Standard-Beacon für den Satz. Daher können Sie die Abfrage nicht ausführen, `CONTAINS(a, :value)` wenn sich ein verschlüsseltes `a` Feld befindet. Ein Standard-Beacon, definiert als, `AsSet` berechnet einzelne Standard-Beacon-Werte für jedes einzelne Element des Satzes und speichert den Beacon-Wert im Element als Satz. Dadurch kann das AWS Database Encryption SDK die Abfrage durchführen. `CONTAINS(a, :value)`

Um einen `AsSet` Standard-Beacon zu definieren, müssen die Elemente in der Gruppe aus derselben Population stammen, sodass sie alle dieselbe Beacon-Länge verwenden können. Das Beacon-Set enthält möglicherweise weniger Elemente als das Klartext-Set, wenn es bei der Berechnung der Beacon-Werte zu Kollisionen kommen sollte.

> **Note**
>
> Berücksichtigen Sie Ihre Sicherheits- und Leistungsanforderungen, bevor Sie Beacons konfigurieren. `AsSet` `AsSet`Beacons können die Menge an statistischen Informationen, die über die Verteilung Ihres Datensatzes identifiziert werden können, erhöhen. Sie könnten beispielsweise die Größe des Klartext-Datensatzes offenlegen.

Java

```
List<StandardBeacon> standardBeaconList = new ArrayList<>();
StandardBeacon exampleStandardBeacon = StandardBeacon.builder()
    .name("beaconName")
    .length(beaconLengthInBits)
    .style(
        BeaconStyle.builder()
           .asSet(AsSet.builder().build())
        .build()
    )
    .build();
standardBeaconList.add(exampleStandardBeacon);
```

C#/.NET

```
new StandardBeacon
```

```
{
    Name = "beaconName",
    Length = beaconLengthInBits,
    Style = new BeaconStyle
    {
        AsSet = new AsSet()
    }
}
```

Rust

```
StandardBeacon::builder()
    .name("beacon_name")
    .length(beacon_length_in_bits)
    .style(BeaconStyle::AsSet(AsSet::builder().build()?))
    .build()?
```

SharedSet

Ein Standard-Beacon, definiert als, `SharedSet` kombiniert die `AsSet` Funktionen `Shared` und, sodass Sie Gleichheitssuchen für die verschlüsselten Werte einer Menge und eines Felds durchführen können. Auf diese Weise kann das AWS Database Encryption SDK die Abfrage durchführen, `CONTAINS(a, b)` bei der *a* es sich um einen verschlüsselten Satz und um ein verschlüsseltes Feld *b* handelt.

> **Note**
>
> Berücksichtigen Sie Ihre Sicherheits- und Leistungsanforderungen, bevor Sie `Shared` Beacons konfigurieren. `SharedSet`Beacons können die Menge an statistischen Informationen, die über die Verteilung Ihres Datensatzes identifiziert werden können, erhöhen. Sie könnten beispielsweise Aufschluss darüber geben, wie groß der Klartextsatz ist oder welche gemeinsam genutzten Felder denselben Klartextwert enthalten.

Java

```
List<StandardBeacon> standardBeaconList = new ArrayList<>();
StandardBeacon exampleStandardBeacon = StandardBeacon.builder()
    .name("beacon2")
```

```
    .length(beaconLengthInBits)
    .style(
        BeaconStyle.builder()
           .sharedSet(SharedSet.builder().other("beacon1").build())
        .build()
    )
    .build();
standardBeaconList.add(exampleStandardBeacon);
```

C#/.NET

```
new StandardBeacon
{
    Name = "beacon2",
    Length = beaconLengthInBits,
    Style = new BeaconStyle
    {
        SharedSet = new SharedSet { Other = "beacon1" }
    }
}
```

Rust

```
StandardBeacon::builder()
    .name("beacon2")
    .length(beacon_length_in_bits)
    .style(BeaconStyle::SharedSet(
        SharedSet::builder().other("beacon1").build()?,
    ))
    .build()?
```

Konfiguration von Compound-Beacons

Zusammengesetzte Beacons kombinieren wörtliche Klartext-Zeichenketten und Standard-Beacons, um komplexe Datenbankoperationen durchzuführen, z. B. das Abfragen von zwei verschiedenen Datensatztypen aus einem einzigen Index oder das Abfragen einer Kombination von Feldern mit einem Sortierschlüssel. Zusammengesetzte Beacons können aus Feldern, und erstellt werden. `ENCRYPT_AND_SIGN` `SIGN_ONLY` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Sie müssen für jedes verschlüsselte Feld, das im Verbund-Beacon enthalten ist, einen Standard-Beacon erstellen.

> **Note**
>
> Wir empfehlen, zu überprüfen, ob Ihre Compound-Beacons das erwartete Ergebnis erzielen, bevor Sie Ihre Datenbank auffüllen. Weitere Informationen finden Sie unter Beacon-Ausgaben testen.

Beispiel für eine Konfigurationssyntax

Java

Konfiguration eines zusammengesetzten Beacons

Im folgenden Beispiel werden verschlüsselte und signierte Teilelisten lokal in der Konfiguration der Compound Beacons definiert.

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

Definition der Beacon-Version

Im folgenden Beispiel werden verschlüsselte und signierte Teilelisten global in der Beacon-Version definiert. Weitere Informationen zur Definition der Beacon-Version finden Sie unter Beacons verwenden.

```
 List<BeaconVersion> beaconVersions = new ArrayList<>();
beaconVersions.add(
    BeaconVersion.builder()
        .standardBeacons(standardBeaconList)
        .compoundBeacons(compoundBeaconList)
        .encryptedParts(encryptedPartList)
        .signedParts(signedPartList)
        .version(1) // MUST be 1
        .keyStore(keyStore)
```

```
        .keySource(BeaconKeySource.builder()
            .single(SingleKeyStore.builder()
                .keyId(branchKeyId)
                .cacheTTL(6000)
                .build())
            .build())
        .build()
);
```

C# / .NET

Sehen Sie sich das vollständige Codebeispiel an: .cs BeaconConfig

Konfiguration eines zusammengesetzten Beacons

Im folgenden Beispiel werden verschlüsselte und signierte Teilelisten lokal in der Konfiguration der Compound Beacons definiert.

```
var compoundBeaconList = new List<CompoundBeacon>();
var exampleCompoundBeacon = new CompoundBeacon
 {
    Name = "compoundBeaconName",
    Split = ".",
    Encrypted = encryptedPartList,
    Signed = signedPartList,
    Constructors = constructorList
 };
compoundBeaconList.Add(exampleCompoundBeacon);
```

Definition der Beacon-Version

Im folgenden Beispiel werden verschlüsselte und signierte Teilelisten global in der Beacon-Version definiert. Weitere Informationen zur Definition der Beacon-Version finden Sie unter Beacons verwenden.

```
var beaconVersions = new List<BeaconVersion>
{
    new BeaconVersion
    {
        StandardBeacons = standardBeaconList,
        CompoundBeacons = compoundBeaconList,
        EncryptedParts = encryptedPartsList,
```

```
        SignedParts = signedPartsList,
        Version = 1, // MUST be 1
        KeyStore = keyStore,
        KeySource = new BeaconKeySource
        {
            Single = new SingleKeyStore
            {
                KeyId = branchKeyId,
                CacheTTL = 6000
            }
        }
    }
};
```

Rust

Sehen Sie sich das vollständige Codebeispiel an: beacon_config.rs

Konfiguration eines zusammengesetzten Beacons

Im folgenden Beispiel werden verschlüsselte und signierte Teilelisten lokal in der Konfiguration der Compound Beacons definiert.

```
let compound_beacon_list = vec![
    CompoundBeacon::builder()
        .name("compound_beacon_name")
        .split(".")
        .encrypted(encrypted_parts_list)
        .signed(signed_parts_list)
        .constructors(constructor_list)
        .build()?
```

Definition der Beacon-Version

Im folgenden Beispiel werden verschlüsselte und signierte Teilelisten global in der Beacon-Version definiert. Weitere Informationen zur Definition der Beacon-Version finden Sie unter Beacons verwenden.

```
let beacon_versions = BeaconVersion::builder()
    .standard_beacons(standard_beacon_list)
    .compound_beacons(compound_beacon_list)
    .encrypted_parts(encrypted_parts_list)
```

```
    .signed_parts(signed_parts_list)
    .version(1) // MUST be 1
    .key_store(key_store.clone())
    .key_source(BeaconKeySource::Single(
        SingleKeyStore::builder()
            .key_id(branch_key_id)
            .cache_ttl(6000)
            .build()?,
    ))
    .build()?;
let beacon_versions = vec![beacon_versions];
```

Sie können Ihre verschlüsselten und signierten Teile in lokal oder global definierten Listen definieren. Wir empfehlen, Ihre verschlüsselten und signierten Teile wann immer möglich in einer globalen Liste in der Beacon-Version zu definieren. Indem Sie verschlüsselte und signierte Teile global definieren, können Sie jedes Teil einmal definieren und die Teile dann in mehreren Compound-Beacon-Konfigurationen wiederverwenden. Wenn Sie beabsichtigen, einen verschlüsselten oder signierten Teil nur einmal zu verwenden, können Sie ihn in einer lokalen Liste in der Compound-Beacon-Konfiguration definieren. Sie können in Ihrer Konstruktorliste sowohl auf lokale als auch auf globale Teile verweisen.

Wenn Sie Ihre verschlüsselten und signierten Teilelisten global definieren, müssen Sie eine Liste von Konstruktorteilen bereitstellen, in der alle Möglichkeiten aufgeführt sind, wie der Compound Beacon die Felder in Ihrer Compound-Beacon-Konfiguration zusammenstellen kann.

> **Note**
>
> Um verschlüsselte und signierte Teilelisten global zu definieren, müssen Sie Version 3.2 oder höher des AWS Database Encryption SDK verwenden. Stellen Sie die neue Version allen Lesern zur Verfügung, bevor Sie neue Teile global definieren.
> Sie können bestehende Beacon-Konfigurationen nicht aktualisieren, um verschlüsselte und signierte Teilelisten global zu definieren.

Um eine Verbundstation zu konfigurieren, geben Sie die folgenden Werte an.

Name des Beacons

Der Name, den Sie bei der Abfrage eines verschlüsselten Felds verwenden.

Ein Beacon-Name kann derselbe Name wie ein verschlüsseltes Feld oder virtuelles Feld sein, er kann jedoch nicht derselbe Name wie ein unverschlüsseltes Feld sein. Keine zwei Beacons können denselben Beacon-Namen haben. Hilfe bei der Bestimmung des besten Beacon-Namens für Ihre Implementierung finden Sie unter Auswahl eines Beacon-Namens.

Charakter teilen

Das Zeichen, das verwendet wird, um die Teile zu trennen, aus denen Ihr Verbundsignal besteht.

Das Trennzeichen darf in den Klartextwerten der Felder, aus denen der Verbundbeacon aufgebaut ist, nicht vorkommen.

Verschlüsselte Teileliste

Identifiziert die `ENCRYPT_AND_SIGN` Felder, die im Compound Beacon enthalten sind.

Jeder Teil muss einen Namen und ein Präfix enthalten. Der Teilname muss der Name des Standard-Beacons sein, der aus dem verschlüsselten Feld erstellt wurde. Das Präfix kann eine beliebige Zeichenfolge sein, muss jedoch eindeutig sein. Ein verschlüsselter Teil kann nicht dasselbe Präfix wie ein signierter Teil haben. Es wird empfohlen, einen kurzen Wert zu verwenden, der den Teil von anderen Teilen unterscheidet, die vom Compound Beacon bedient werden.

Wir empfehlen, Ihre verschlüsselten Teile nach Möglichkeit global zu definieren. Sie könnten erwägen, einen verschlüsselten Teil lokal zu definieren, wenn Sie ihn nur in einem Compound Beacon verwenden möchten. Ein lokal definierter verschlüsselter Teil kann nicht dasselbe Präfix oder denselben Namen haben wie ein global definierter verschlüsselter Teil.

Java

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

C# / .NET

```
var encryptedPartList = new List<EncryptedPart>();
var encryptedPartExample = new EncryptedPart
 {
    Name = "compoundBeaconName",
```

```
    Prefix = "E-"
 };
encryptedPartList.Add(encryptedPartExample);
```

Rust

```
let encrypted_parts_list = vec![
    EncryptedPart::builder()
        .name("standard_beacon_name")
        .prefix("E-")
        .build()?
];
```

Signierte Teileliste

Identifiziert die signierten Felder, die im Compound Beacon enthalten sind.

> **Note**
>
> Signierte Teile sind optional. Sie können einen Compound-Beacon konfigurieren, der keine signierten Teile referenziert.

Jeder Teil muss einen Namen, eine Quelle und ein Präfix enthalten. Die Quelle ist das `SIGN_ONLY` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` ODER-Feld, das der Teil identifiziert. Die Quelle muss ein Feldname oder ein Index sein, der auf den Wert eines verschachtelten Felds verweist. Wenn Ihr Teilname die Quelle identifiziert, können Sie die Quelle weglassen und das AWS Database Encryption SDK verwendet den Namen automatisch als Quelle. Wir empfehlen, wann immer möglich, die Quelle als Teilnamen anzugeben. Das Präfix kann eine beliebige Zeichenfolge sein, muss jedoch eindeutig sein. Ein signierter Teil kann nicht dasselbe Präfix wie ein verschlüsselter Teil haben. Es wird empfohlen, einen kurzen Wert zu verwenden, der den Teil von anderen Teilen unterscheidet, die vom Compound Beacon bedient werden.

Wir empfehlen, Ihre signierten Teile nach Möglichkeit global zu definieren. Sie könnten erwägen, ein signiertes Teil lokal zu definieren, wenn Sie es nur in einem Compound Beacon verwenden möchten. Ein lokal definierter signierter Teil kann nicht dasselbe Präfix oder denselben Namen haben wie ein global definierter signierter Teil.

Java

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

C# / .NET

```
var signedPartsList = new List<SignedPart>
{
    new SignedPart { Name = "signedFieldName1", Prefix = "S-" },
    new SignedPart { Name = "signedFieldName2", Prefix = "SF-" }
};
```

Rust

```
let signed_parts_list = vec![
    SignedPart::builder()
        .name("signed_field_name_1")
        .prefix("S-")
        .build()?,
   SignedPart::builder()
        .name("signed_field_name_2")
        .prefix("SF-")
        .build()?,
];
```

Liste der Konstruktoren

Identifiziert die Konstruktoren, die die verschiedenen Arten definieren, wie die verschlüsselten und signierten Teile durch den Compound Beacon zusammengesetzt werden können. Sie können in Ihrer Konstruktorliste sowohl auf lokale als auch auf globale Bauteile verweisen.

Wenn Sie Ihr Compound-Beacon aus global definierten, verschlüsselten und signierten Teilen erstellen, müssen Sie eine Konstruktorliste angeben.

Wenn Sie keine global definierten, verschlüsselten oder signierten Teile verwenden, um Ihren Compound-Beacon zu erstellen, ist die Liste der Konstruktoren optional. Wenn Sie keine

Konstruktorliste angeben, stellt das AWS Database Encryption SDK den Compound Beacon mit dem folgenden Standardkonstruktor zusammen.

- Alle signierten Teile in der Reihenfolge, in der sie der signierten Teileliste hinzugefügt wurden
- Alle verschlüsselten Teile in der Reihenfolge, in der sie der verschlüsselten Teileliste hinzugefügt wurden
- Alle Teile sind erforderlich

Konstruktoren

Jeder Konstruktor ist eine geordnete Liste von Konstruktorteilen, die eine Art und Weise definiert, wie der Compound Beacon zusammengebaut werden kann. Die Konstruktorteile werden in der Reihenfolge zusammengefügt, in der sie der Liste hinzugefügt wurden, wobei jeder Teil durch das angegebene Trennzeichen getrennt wird.

Jeder Konstruktorteil benennt einen verschlüsselten Teil oder einen signierten Teil und definiert, ob dieser Teil innerhalb des Konstruktors erforderlich oder optional ist. Wenn Sie beispielsweise ein zusammengesetztes Beacon für, und abfragen möchten `Field1` `Field1.Field2Field1.Field2.Field3`, markieren Sie und `Field3` als optional `Field2` und erstellen Sie einen Konstruktor.

Jeder Konstruktor muss mindestens einen erforderlichen Teil haben. Wir empfehlen, den ersten Teil in jedem Konstruktor als erforderlich festzulegen, damit Sie den `BEGINS_WITH` Operator in Ihren Abfragen verwenden können.

Ein Konstruktor ist erfolgreich, wenn alle erforderlichen Teile im Datensatz vorhanden sind. Wenn Sie einen neuen Datensatz schreiben, ermittelt der Verbundbeacon anhand der Konstruktorliste, ob der Beacon aus den bereitgestellten Werten zusammengesetzt werden kann. Es versucht, den Beacon in der Reihenfolge zusammenzustellen, in der die Konstruktoren der Konstruktorliste hinzugefügt wurden, und verwendet den ersten Konstruktor, der erfolgreich ist. Wenn keine Konstruktoren erfolgreich sind, wird der Beacon nicht in den Datensatz geschrieben.

Alle Leser und Autoren sollten dieselbe Reihenfolge der Konstruktoren angeben, um sicherzustellen, dass ihre Abfrageergebnisse korrekt sind.

Verwenden Sie die folgenden Verfahren, um Ihre eigene Konstruktorliste anzugeben.

1. Erstellen Sie für jeden verschlüsselten und signierten Teil einen Konstruktorteil, um zu definieren, ob dieser Teil erforderlich ist oder nicht.

Der Name des Konstruktorteils muss der Name des Standard-Beacons oder des signierten Felds sein, für das er steht.

Java

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

C# / .NET

```
var field1ConstructorPart = new ConstructorPart { Name = "Field1", Required
 = true };
```

Rust

```
let field_1_constructor_part = ConstructorPart::builder()
    .name("field_1")
    .required(true)
    .build()?;
```

2. Erstellen Sie mithilfe der Konstruktorteile, die Sie in Schritt 1 erstellt haben, einen Konstruktor für jede mögliche Art und Weise, wie das Verbundsignal zusammengebaut werden kann.

   Wenn Sie beispielsweise nach `Field1.Field2.Field3` und abfragen möchten`Field4.Field2.Field3`, müssen Sie zwei Konstruktoren erstellen. `Field1`und `Field4` können beide erforderlich sein, da sie in zwei separaten Konstruktoren definiert sind.

   Java

```
// 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();
```

C# / .NET

```
// Create a list for Field1.Field2.Field3 queries
 var field123ConstructorPartList = new Constructor
{
    Parts = new List<ConstructorPart> { field1ConstructorPart,
 field2ConstructorPart, field3ConstructorPart }
};
// Create a list for Field4.Field2.Field1 queries
var field421ConstructorPartList = new Constructor
{
    Parts = new List<ConstructorPart> { field4ConstructorPart,
 field2ConstructorPart, field1ConstructorPart }
};
```

Rust

```
// Create a list for field1.field2.field3 queries
let field1_field2_field3_constructor = Constructor::builder()
    .parts(vec![
        field1_constructor_part,
        field2_constroctor_part.clone(),
        field3_constructor_part,
    ])
    .build()?;

// Create a list for field4.field2.field1 queries
let field4_field2_field1_constructor = Constructor::builder()
    .parts(vec![
        field4_constructor_part,
        field2_constroctor_part.clone(),
        field1_constructor_part,
    ])
    .build()?;
```

3. Erstellen Sie eine Konstruktorliste, die alle Konstruktoren enthält, die Sie in Schritt 2 erstellt haben.

   Java

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

   C# / .NET

   ```
   var constructorList = new List<Constructor>
   {
       field123Constructor,
       field421Constructor
   };
   ```

   Rust

   ```
   let constructor_list = vec![
       field1_field2_field3_constructor,
       field4_field2_field1_constructor,
   ];
   ```

4. Geben Sie den an`constructorList`, wenn Sie Ihren Verbundbeacon erstellen.

Beispielkonfigurationen

Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Die folgenden Beispiele zeigen, wie Standard- und Verbund-Beacons konfiguriert werden. Die folgenden Konfigurationen bieten keine Beacon-Längen. Hilfe bei der Bestimmung der geeigneten Beacon-Länge für Ihre Konfiguration finden Sie unter Wählen Sie eine Beacon-Länge.

Vollständige Codebeispiele, die die Konfiguration und Verwendung von Beacons demonstrieren, finden Sie in den durchsuchbaren Verschlüsselungsbeispielen für Java, .NET und Rust im -dynamodb-Repository unter. aws-database-encryption-sdk GitHub

Themen

Standard-Beacons

Wenn Sie das `inspector_id_last4` Feld nach exakten Übereinstimmungen abfragen möchten, erstellen Sie ein Standard-Beacon mit der folgenden Konfiguration.

Java

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

C# / .NET

```
var standardBeaconList = new List<StandardBeacon>>);
StandardBeacon exampleStandardBeacon = new StandardBeacon
  {
    Name = "inspector_id_last4",
    Length = 10
  };
standardBeaconList.Add(exampleStandardBeacon);
```

Rust

```
let last4_beacon = StandardBeacon::builder()
    .name("inspector_id_last4")
    .length(10)
    .build()?;

let unit_beacon = StandardBeacon::builder().name("unit").length(30).build()?;

let standard_beacon_list = vec![last4_beacon, unit_beacon];
```

Zusammengesetzte Beacons

Wenn Sie die `UnitInspection` Datenbank auf `inspector_id_last4` und abfragen möchten`inspector_id_last4.unit`, erstellen Sie ein zusammengesetztes Beacon mit der folgenden Konfiguration. Für diesen Compound Beacon sind nur verschlüsselte Teile erforderlich.

Java

```
// 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();
```

C# / .NET

```
// 1. Create standard beacons for the inspector_id_last4 and unit fields.
StandardBeacon inspectorBeacon = new StandardBeacon
 {
   Name = "inspector_id_last4",
   Length = 10
 };
standardBeaconList.Add(inspectorBeacon);
StandardBeacon unitBeacon = new StandardBeacon
 {
    Name = "unit",
    Length = 30
 };
standardBeaconList.Add(unitBeacon);

// 2. Define the encrypted parts.
var last4EncryptedPart = new EncryptedPart

// 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"
var last4EncryptedPart = new EncryptedPart
 {
   Name = "inspector_id_last4",
   Prefix = "I-"
 };
encryptedPartList.Add(last4EncryptedPart);

var unitEncryptedPart = new EncryptedPart
 {
   Name = "unit",
   Prefix = "U-"
```

```
 };
encryptedPartList.Add(unitEncryptedPart);

// 3. Create the compound beacon.
// This compound beacon only requires a name, split character,
// and list of encrypted parts
var compoundBeaconList = new List<CompoundBeacon>>);
var inspectorCompoundBeacon = new CompoundBeacon
  {
      Name = "inspector_id_last4",
      Split = ".",
      Encrypted = encryptedPartList
  };
compoundBeaconList.Add(inspectorCompoundBeacon);
```

Rust

```
// 1. Create standard beacons for the inspector_id_last4 and unit fields.
let last4_beacon = StandardBeacon::builder()
    .name("inspector_id_last4")
    .length(10)
    .build()?;

let unit_beacon = StandardBeacon::builder().name("unit").length(30).build()?;

let standard_beacon_list = vec![last4_beacon, unit_beacon];

// 2. Define the encrypted parts.
// 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"
let encrypted_parts_list = vec![
    EncryptedPart::builder()
        .name("inspector_id_last4")
        .prefix("I-")
        .build()?,
    EncryptedPart::builder().name("unit").prefix("U-").build()?,
];

// 3. Create the compound beacon
// This compound beacon only requires a name, split character,
// and list of encrypted parts
```

```
let compound_beacon_list = vec![CompoundBeacon::builder()
    .name("last4UnitCompound")
    .split(".")
    .encrypted(encrypted_parts_list)
    .build()?];
```

Verwendung von Beacons

Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Mit Beacons können Sie verschlüsselte Datensätze durchsuchen, ohne die gesamte abgefragte Datenbank zu entschlüsseln. Beacons sind so konzipiert, dass sie in neuen, nicht aufgefüllten Datenbanken implementiert werden können. Jedes Beacon, das in einer vorhandenen Datenbank konfiguriert ist, ordnet nur neue Datensätze zu, die in die Datenbank geschrieben wurden. Beacons werden anhand des Klartextwerts eines Felds berechnet. Sobald das Feld verschlüsselt ist, kann das Beacon keine vorhandenen Daten zuordnen. Nachdem Sie neue Datensätze mit einem Beacon geschrieben haben, können Sie die Konfiguration des Beacons nicht mehr aktualisieren. Sie können jedoch neue Beacons für neue Felder hinzufügen, die Sie Ihrem Datensatz hinzufügen.

Nachdem Sie Ihre Beacons konfiguriert haben, müssen Sie die folgenden Schritte ausführen, bevor Sie beginnen, Ihre Datenbank zu füllen und Abfragen an Ihren Beacons durchzuführen.

1. Erstellen Sie einen hierarchischen Schlüsselbund AWS KMS

   Um eine durchsuchbare Verschlüsselung zu verwenden, müssen Sie den AWS KMS hierarchischen Schlüsselbund verwenden, um die Datenschlüssel zu generieren, zu verschlüsseln und zu entschlüsseln, die zum Schutz Ihrer Daten verwendet werden.

   Nachdem Sie Ihre Beacons konfiguriert haben, stellen Sie die Voraussetzungen für den hierarchischen Schlüsselbund zusammen und erstellen Sie Ihren hierarchischen Schlüsselbund.

   Weitere Informationen darüber, warum der hierarchische Schlüsselbund erforderlich ist, finden Sie unter Verwenden des hierarchischen Schlüsselbunds für durchsuchbare Verschlüsselung.
2.

Definieren Sie die Beacon-Version

Geben Sie Ihre`keyStore`,`keySource`, eine Liste aller von Ihnen konfigurierten Standard-Beacons, eine Liste aller von Ihnen konfigurierten Verbund-Beacons, eine Liste der verschlüsselten Teile, eine Liste der signierten Teile und eine Beacon-Version an. Sie müssen die 1 Beacon-Version angeben. Hinweise zur Definition Ihres finden Sie `keySource` unterDefinieren Sie Ihre Beacon-Schlüsselquelle.

Das folgende Java-Beispiel definiert die Beacon-Version für eine Single-Tenant-Datenbank. Hilfe bei der Definition der Beacon-Version für eine Mehrmandantendatenbank finden Sie unter Durchsuchbare Verschlüsselung für Mehrmandantendatenbanken.

Java

```
 List<BeaconVersion> beaconVersions = new ArrayList<>();
beaconVersions.add(
    BeaconVersion.builder()
        .standardBeacons(standardBeaconList)
        .compoundBeacons(compoundBeaconList)
        .encryptedParts(encryptedPartsList)
        .signedParts(signedPartsList)
        .version(1) // MUST be 1
        .keyStore(keyStore)
        .keySource(BeaconKeySource.builder()
            .single(SingleKeyStore.builder()
                .keyId(branchKeyId)
                .cacheTTL(6000)
                .build())
            .build())
        .build()
);
```

C# / .NET

```
var beaconVersions = new List<BeaconVersion>
{
    new BeaconVersion
    {
        StandardBeacons = standardBeaconList,
        CompoundBeacons = compoundBeaconList,
        EncryptedParts = encryptedPartsList,
```

```
        SignedParts = signedPartsList,
        Version = 1, // MUST be 1
        KeyStore = branchKeyStoreName,
        KeySource = new BeaconKeySource
        {
            Single = new SingleKeyStore
            {
                KeyId = branch-key-id,
                CacheTTL = 6000
            }
        }
    }
};
```

Rust

```
let beacon_version = BeaconVersion::builder()
    .standard_beacons(standard_beacon_list)
    .compound_beacons(compound_beacon_list)
    .version(1) // MUST be 1
    .key_store(key_store.clone())
    .key_source(BeaconKeySource::Single(
        SingleKeyStore::builder()
            // `keyId` references a beacon key.
            // For every branch key we create in the keystore,
            // we also create a beacon key.
            // This beacon key is not the same as the branch key,
            // but is created with the same ID as the branch key.
            .key_id(branch_key_id)
            .cache_ttl(6000)
            .build()?,
    ))
    .build()?;
let beacon_versions = vec![beacon_version];
```

3. Konfigurieren Sie sekundäre Indizes

   Nachdem Sie Ihre Beacons konfiguriert haben, müssen Sie einen sekundären Index konfigurieren, der die einzelnen Beacons widerspiegelt, bevor Sie in den verschlüsselten Feldern suchen können. Weitere Informationen finden Sie unter Konfiguration sekundärer Indizes mit Beacons.

4. Definieren Sie Ihre kryptografischen Aktionen

Alle Felder, die zum Aufbau eines Standard-Beacons verwendet werden, müssen markiert sein. `ENCRYPT_AND_SIGN` Alle anderen Felder, die zum Bau von Beacons verwendet werden, müssen mit oder markiert `SIGN_ONLY` sein. `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`

5. Konfigurieren Sie einen AWS Database Encryption SDK-Client

   Informationen zur Konfiguration eines AWS Database Encryption SDK-Clients, der die Tabellenelemente in Ihrer DynamoDB-Tabelle schützt, finden Sie unter Clientseitige Java-Verschlüsselungsbibliothek für DynamoDB.

Beacons abfragen

Der Typ des Beacons, den Sie konfigurieren, bestimmt die Art der Abfragen, die Sie ausführen können. Standard-Beacons verwenden Filterausdrücke, um Gleichheitssuchen durchzuführen. Zusammengesetzte Beacons kombinieren wörtliche Klartext-Zeichenketten und Standard-Beacons, um komplexe Abfragen durchzuführen. Wenn Sie verschlüsselte Daten abfragen, suchen Sie nach dem Namen des Beacons.

Sie können die Werte von zwei Standard-Beacons nicht vergleichen, selbst wenn sie denselben zugrunde liegenden Klartext enthalten. Die beiden Standard-Beacons erzeugen zwei verschiedene HMAC-Tags für dieselben Klartext-Werte. Daher können Standard-Beacons die folgenden Abfragen nicht ausführen.

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

Compound Beacons können die folgenden Abfragen ausführen.

- BEGINS_WITH(*a*), wobei der gesamte Wert des Feldes *a* wiedergegeben wird, mit dem die zusammengestellte Verbundstation beginnt. Sie können den BEGINS_WITH Operator nicht verwenden, um einen Wert zu identifizieren, der mit einer bestimmten Teilzeichenfolge beginnt. Sie können jedoch, where BEGINS_WITH(*S_*)*S_*, das Präfix für ein Teil verwenden, mit dem die zusammengebaute Verbundleuchte beginnt.
- CONTAINS(*a*), wobei der gesamte Wert eines Feldes *a* wiedergegeben wird, das die zusammengebaute Verbundleuchte enthält. Sie können den CONTAINS Operator nicht verwenden,

um einen Datensatz zu identifizieren, der eine bestimmte Teilzeichenfolge oder einen Wert innerhalb eines Satzes enthält.

Sie können beispielsweise keine Abfrage `CONTAINS(path, "a"` ausführen, die den Wert in einem Satz *a* widerspiegelt.

- Sie können signierte Teile von Compound-Beacons vergleichen. Wenn Sie signierte Teile vergleichen, können Sie optional das Präfix eines verschlüsselten Teils an einen oder mehrere signierte Teile anhängen, aber Sie können den Wert eines verschlüsselten Felds nicht in eine Abfrage einbeziehen.

  Sie können beispielsweise signierte Teile vergleichen und nach *signedField1* = *signedField2* oder *value* `IN (`*signedField1*`, `*signedField2*`, ...)` abfragen.

  Sie können signierte Teile auch mit dem Präfix eines verschlüsselten Bauteils vergleichen, indem Sie auf „Query on“ klicken*signedField1.A_* = *signedField2.B_*.
- *field* `BETWEEN` *a* `AND` *b*, wo *a* und *b* sind signierte Teile. Sie können optional das Präfix eines verschlüsselten Teils an einen oder mehrere signierte Teile anhängen, aber Sie können den Wert eines verschlüsselten Felds nicht in eine Abfrage einbeziehen.

Sie müssen das Präfix für jeden Teil angeben, den Sie in eine Abfrage auf einem Compound Beacon einbeziehen. Wenn Sie beispielsweise einen Verbundbeacon aus zwei Feldern `encryptedField` und `signedField` erstellt haben`compoundBeacon`, müssen Sie bei der Abfrage des Beacons die für diese beiden Teile konfigurierten Präfixe angeben.

```
compoundBeacon = E_encryptedFieldValue.S_signedFieldValue
```

Durchsuchbare Verschlüsselung für Multitenant-Datenbanken

Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt . AWS Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Um eine durchsuchbare Verschlüsselung in Ihrer Datenbank zu implementieren, müssen Sie einen AWS KMS hierarchischen Schlüsselbund verwenden. Der AWS KMS hierarchische Schlüsselbund generiert, verschlüsselt und entschlüsselt die Datenschlüssel, die zum Schutz Ihrer Datensätze

verwendet werden. Er erstellt auch den Beacon-Schlüssel, der zur Generierung von Beacons verwendet wird. Wenn Sie den AWS KMS hierarchischen Schlüsselbund mit Datenbanken mit mehreren Mandanten verwenden, gibt es für jeden Mandanten einen eigenen Branch- und Beacon-Schlüssel. Um verschlüsselte Daten in einer Multitenant-Datenbank abzufragen, müssen Sie die Beacon-Schlüsselmaterialien identifizieren, die zur Generierung des abgefragten Beacons verwendet wurden. Weitere Informationen finden Sie unter the section called “Verwendung des hierarchischen Schlüsselbunds für durchsuchbare Verschlüsselung”.

Wenn Sie die Beacon-Version für eine Multitenant-Datenbank definieren, geben Sie eine Liste aller von Ihnen konfigurierten Standard-Beacons, eine Liste aller von Ihnen konfigurierten Verbund-Beacons, eine Beacon-Version und eine an. `keySource` Sie müssen Ihre Beacon-Schlüsselquelle als eine `MultiKeyStore` Cache-Gültigkeitsdauer für den lokalen Beacon-Schlüssel-Cache und eine maximale Cachegröße für den lokalen Beacon-Schlüssel-Cache definieren und diese angeben. `keyFieldName`

Wenn Sie signierte Beacons konfiguriert haben, müssen diese in Ihrem enthalten sein. `compoundBeaconList` Signierte Beacons sind eine Art von zusammengesetzten Beacons, die komplexe Abfragen von End-Feldern indizieren und ausführen. `SIGN_ONLY` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`

Java

```
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()
    );
```

C# / .NET

```
var beaconVersions = new List<BeaconVersion>
{
    new BeaconVersion
    {
        StandardBeacons = standardBeaconList,
        CompoundBeacons = compoundBeaconList,
        EncryptedParts = encryptedPartsList,
        SignedParts = signedPartsList,
        Version = 1, // MUST be 1
        KeyStore = branchKeyStoreName,
        KeySource = new BeaconKeySource
        {
            Multi = new MultiKeyStore
            {
                KeyId = branch-key-id,
                CacheTTL = 6000,
                MaxCacheSize = 10
            }
        }
    }
};
```

Rust

```
let beacon_version = BeaconVersion::builder()
    .standard_beacons(standard_beacon_list)
    .compound_beacons(compound_beacon_list)
    .version(1) // MUST be 1
    .key_store(key_store.clone())
    .key_source(BeaconKeySource::Multi(
        MultiKeyStore::builder()
            // `keyId` references a beacon key.
            // For every branch key we create in the keystore,
            // we also create a beacon key.
            // This beacon key is not the same as the branch key,
            // but is created with the same ID as the branch key.
            .key_id(branch_key_id)
            .cache_ttl(6000)
            .max_cache_size(10)
            .build()?,
    ))
```

```
    .build()?;
let beacon_versions = vec![beacon_version];
```

keyFieldName

Das `keyFieldName`definiert den Namen des Felds, in dem der dem Beacon `branch-key-id` zugeordnete Schlüssel gespeichert wird, der zur Generierung von Beacons für einen bestimmten Mandanten verwendet wurde.

Wenn Sie neue Datensätze in Ihre Datenbank schreiben, wird der Beacon-Schlüssel`branch-key-id`, der zur Generierung von Beacons für diesen Datensatz verwendet wurde, in diesem Feld gespeichert.

Standardmäßig `keyField` ist das ein konzeptionelles Feld, das nicht explizit in Ihrer Datenbank gespeichert wird. Das AWS Database Encryption SDK identifiziert den `branch-key-id` anhand des verschlüsselten Datenschlüssels in der Materialbeschreibung und speichert den Wert im Konzept`keyField`, sodass Sie in Ihren Compound Beacons und signierten Beacons darauf verweisen können. Da die Materialbeschreibung signiert ist, gilt das Konzept `keyField` als signiertes Teil.

Sie können das Feld auch als ODER-Feld `keyField` in Ihre kryptografischen Aktionen aufnehmen, um das `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Feld explizit in Ihrer Datenbank zu speichern. `SIGN_ONLY` In diesem Fall müssen Sie das `branch-key-id` in `keyField` jedes Mal, wenn Sie einen Datensatz in Ihre Datenbank schreiben, manuell hinzufügen.

Abfragen von Beacons in einer mandantenfähigen Datenbank

Um ein Beacon abzufragen, müssen Sie das `keyField` in Ihre Abfrage aufnehmen, um die entsprechenden Beacon-Schlüsselmaterialien zu identifizieren, die für die Neuberechnung des Beacons erforderlich sind. Sie müssen den Schlüssel angeben, der dem Beacon `branch-key-id` zugeordnet ist, der zur Generierung der Beacons für einen Datensatz verwendet wurde. Sie können den Anzeigenamen, der den Namen eines Mandanten identifiziert, nicht `branch-key-id` in der Branch-Schlüssel-ID angeben. Sie können den auf folgende Weise `keyField` in Ihre Abfragen einbeziehen.

Zusammengesetzte Beacons

Unabhängig davon, ob Sie sie explizit `keyField` in Ihren Aufzeichnungen speichern oder nicht, können Sie sie `keyField` direkt als signierten Teil in Ihre Compound-Beacons aufnehmen. Der `keyField` signierte Teil muss erforderlich sein.

Wenn Sie beispielsweise ein Verbundsignal aus zwei Feldern erstellen möchten`compoundBeacon`, müssen Sie auch das `keyField` als signierten Teil angeben. `encryptedField` `signedField` Auf diese Weise können Sie die folgende Abfrage ausführen`compoundBeacon`.

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

Signierte Beacons

Das AWS Database Encryption SDK verwendet Standard- und Verbundbeacons, um durchsuchbare Verschlüsselungslösungen bereitzustellen. Diese Beacons müssen mindestens ein verschlüsseltes Feld enthalten. Das AWS Database Encryption SDK unterstützt jedoch auch signierte Beacons, die vollständig aus Klartext `SIGN_ONLY` und Feldern konfiguriert werden können. `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`

Signierte Beacons können aus einem einzigen Teil aufgebaut werden. Unabhängig davon, ob Sie das explizit `keyField` in Ihren Aufzeichnungen speichern oder nicht, können Sie daraus ein signiertes Beacon erstellen `keyField` und es verwenden, um zusammengesetzte Abfragen zu erstellen, die eine Abfrage auf dem `keyField` signierten Beacon mit einer Abfrage auf einem Ihrer anderen Beacons kombinieren. Sie könnten beispielsweise die folgende Abfrage ausführen.

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

Hilfe zur Konfiguration signierter Beacons finden Sie unter Signierte Beacons erstellen

Fragen Sie direkt auf dem **`keyField`**

Wenn Sie das `keyField` in Ihren kryptografischen Aktionen angegeben und das Feld explizit in Ihrem Datensatz gespeichert haben, können Sie eine zusammengesetzte Abfrage erstellen, die eine Abfrage auf Ihrem Beacon mit einer Abfrage auf dem kombiniert. `keyField` Sie können eine direkte Abfrage auf dem wählen, `keyField` wenn Sie ein Standard-Beacon abfragen möchten. Sie könnten beispielsweise die folgende Abfrage ausführen.

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

AWS Datenbankverschlüsselungs-SDK für DynamoDB

Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt . AWS Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Das AWS Database Encryption SDK für DynamoDB ist eine Softwarebibliothek, mit der Sie clientseitige Verschlüsselung in Ihr Amazon DynamoDB-Design integrieren können. Das AWS Database Encryption SDK für DynamoDB bietet Verschlüsselung auf Attributebene und ermöglicht es Ihnen, anzugeben, welche Elemente verschlüsselt werden sollen und welche Elemente in die Signaturen aufgenommen werden sollen, die die Authentizität Ihrer Daten sicherstellen. Durch die Verschlüsselung Ihrer sensiblen Daten während der Übertragung und im Speicher wird sichergestellt, dass Ihre Klartextdaten nicht für Dritte verfügbar sind, auch nicht. AWS

> **Note**
>
> Das AWS Database Encryption SDK unterstützt PartiQL nicht.

In DynamoDB ist eine Tabelle eine Sammlung von Elementen. Jedes Element ist eine Sammlung von Attributen. Jedes Attribut verfügt über einen Namen und einen Wert. Das AWS Database Encryption SDK für DynamoDB verschlüsselt die Werte von Attributen. Dann berechnet er eine Signatur unter Verwendung der Attribute. Sie geben an, welche Attributwerte verschlüsselt und welche in die Signatur der kryptografischen Aktionen aufgenommen werden sollen.

Die Themen in diesem Kapitel bieten einen Überblick über das AWS Database Encryption SDK für DynamoDB, einschließlich der verschlüsselten Felder, Anleitungen zur Client-Installation und -Konfiguration sowie Java-Beispiele, die Ihnen den Einstieg erleichtern.

Themen

- Clientseitige und serverseitige Verschlüsselung
- Welche Felder sind verschlüsselt und signiert?
- Durchsuchbare Verschlüsselung in DynamoDB
- Aktualisierung Ihres Datenmodells
- AWS Database Encryption SDK für DynamoDB, verfügbare Programmiersprachen

- Legacy-DynamoDB-Verschlüsselungsclient

Clientseitige und serverseitige Verschlüsselung

> Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Das AWS Database Encryption SDK für DynamoDB unterstützt die clientseitige Verschlüsselung, bei der Sie Ihre Tabellendaten verschlüsseln, bevor Sie sie an Ihre Datenbank senden. DynamoDB bietet jedoch eine serverseitige Funktion zur Verschlüsselung im Ruhezustand, die Ihre Tabelle transparent verschlüsselt, wenn sie auf der Festplatte gespeichert wird, und sie entschlüsselt, wenn Sie auf die Tabelle zugreifen.

Welche Tools Sie wählen, hängt von der Sensibilität Ihrer Daten und den Sicherheitsanforderungen Ihrer Anwendung ab. Sie können sowohl das AWS Database Encryption SDK für DynamoDB als auch Encryption at Rest verwenden. Wenn Sie verschlüsselte und signierte Elemente an DynamoDB senden, erkennt DynamoDB die Elemente nicht als geschützt. Er erkennt nur typische Tabellenelemente mit binären Attributwerten.

Serverseitige Verschlüsselung im Ruhezustand

DynamoDB unterstützt Encryption at Rest, eine serverseitige Verschlüsselungsfunktion, bei der DynamoDB Ihre Tabellen transparent für Sie verschlüsselt, wenn die Tabelle dauerhaft auf der Festplatte gespeichert wird, und sie entschlüsselt, wenn Sie auf die Tabellendaten zugreifen.

Wenn Sie ein AWS SDK für die Interaktion mit DynamoDB verwenden, werden Ihre Daten standardmäßig bei der Übertragung über eine HTTPS-Verbindung verschlüsselt, am DynamoDB-Endpunkt entschlüsselt und dann erneut verschlüsselt, bevor sie in DynamoDB gespeichert werden.

- Standardmäßig Verschlüsselung. DynamoDB verschlüsselt und entschlüsselt alle Tabellen transparent, wenn sie geschrieben werden. Es ist nicht möglich, die Verschlüsselung im Ruhezustand zu aktivieren oder zu deaktivieren.
- DynamoDB erstellt und verwaltet die kryptografischen Schlüssel.Der eindeutige Schlüssel für jede Tabelle ist durch eine, die niemals AWS Key Management Service()AWS KMS unverschlüsselt verlässt AWS KMS key, geschützt. Standardmäßig verwendet DynamoDB ein AWS-eigener Schlüsselim DynamoDB-Dienstkonto, aber Sie können einen Von AWS verwalteter Schlüsseloder

einen vom Kunden verwalteten Schlüssel in Ihrem Konto wählen, um einige oder alle Ihre Tabellen zu schützen.

- Alle Tabellendaten sind auf der Festplatte verschlüsselt.Wenn eine verschlüsselte Tabelle auf der Festplatte gespeichert wird, verschlüsselt DynamoDB alle Tabellendaten, einschließlich des Primärschlüssels und der lokalen und globalen Sekundärindizes. Wenn Ihre Tabelle einen Sortierschlüssel hat, werden einige der Sortierschlüssel, die Bereichsgrenzen markieren, in Klartext in den Metadaten der Tabelle gespeichert.
- Objekte, die sich auf Tabellen beziehen, werden ebenfalls verschlüsselt. Verschlüsselung im Ruhezustand schützt DynamoDB-Streams, globale Tabellen und Backups, wann immer sie auf dauerhafte Medien geschrieben werden.
- Ihre Elemente werden entschlüsselt, wenn Sie darauf zugreifen.Wenn Sie auf die Tabelle zugreifen, entschlüsselt DynamoDB den Teil der Tabelle, der Ihr Zielelement enthält, und gibt das Klartextelement an Sie zurück.

AWS Datenbankverschlüsselungs-SDK für DynamoDB

Die clientseitige Verschlüsselung bietet end-to-end Schutz für Ihre Daten bei der Übertragung und im Speicher, von der Quelle bis zur Speicherung in DynamoDB. Ihre Klartextdaten werden niemals Dritten zugänglich gemacht, auch nicht. AWS Sie können das AWS Database Encryption SDK für DynamoDB mit neuen DynamoDB-Tabellen verwenden oder Ihre vorhandenen Amazon DynamoDB-Tabellen auf die neueste Version des Database Encryption SDK für DynamoDB migrieren. AWS

- Ihre Daten sind während des Transports und im Ruhezustand geschützt. Es wird niemals Dritten zugänglich gemacht, auch nicht. AWS
- Sie können Ihre Tabellenelemente signieren. Sie können das AWS Database Encryption SDK for DynamoDB anweisen, eine Signatur für das gesamte oder einen Teil eines Tabellenelements, einschließlich der Primärschlüsselattribute, zu berechnen. Mit dieser Signatur können Sie nicht autorisierte Änderungen am gesamten Element erkennen, einschließlich des Hinzufügens oder Löschens von Attributen oder des Vertauschens von Attributwerten.
- Sie bestimmen, wie Ihre Daten geschützt werden, indem Sie einen Schlüsselbund auswählen. Ihr Schlüsselbund bestimmt die Umschließungsschlüssel, die Ihre Datenschlüssel und letztlich Ihre Daten schützen. Verwenden Sie die sichersten Verpackungsschlüssel, die für Ihre Aufgabe praktisch sind.
- Das AWS Database Encryption SDK für DynamoDB verschlüsselt nicht die gesamte Tabelle. Sie wählen aus, welche Attribute in Ihren Elementen verschlüsselt werden. Das AWS Database Encryption SDK für DynamoDB verschlüsselt nicht ein ganzes Element. Es verschlüsselt weder

Attributnamen noch die Namen oder Werte der Primärschlüsselattribute (Partitionsschlüssel und Sortierschlüssel).

AWS Encryption SDK

Wenn Sie Daten verschlüsseln, die Sie in DynamoDB speichern, empfehlen wir das AWS Database Encryption SDK für DynamoDB.

Die AWS Encryption SDK ist eine clientseitige Verschlüsselungsbibliothek, die Ihnen hilft, generische Daten zu verschlüsseln und zu entschlüsseln. Obwohl sie beliebige Datentypen schützen kann, ist sie nicht darauf ausgelegt, mit strukturierten Daten wie Datenbankeinträgen zu arbeiten. Im Gegensatz zum AWS Database Encryption SDK für DynamoDB AWS Encryption SDK kann das keine Integritätsprüfung auf Elementebene bereitstellen und hat keine Logik, um Attribute zu erkennen oder die Verschlüsselung von Primärschlüsseln zu verhindern.

Wenn Sie das verwenden AWS Encryption SDK , um ein Element Ihrer Tabelle zu verschlüsseln, denken Sie daran, dass es nicht mit dem AWS Database Encryption SDK für DynamoDB kompatibel ist. Sie können nicht mit einer Bibliothek verschlüsseln und mit einer anderen entschlüsseln.

Welche Felder sind verschlüsselt und signiert?

Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Das AWS Database Encryption SDK für DynamoDB ist eine clientseitige Verschlüsselungsbibliothek, die speziell für Amazon DynamoDB DynamoDB-Anwendungen entwickelt wurde. Amazon DynamoDB speichert Daten in Tabellen, bei denen es sich um eine Sammlung von Elementen handelt. Jedes Element ist eine Sammlung von Attributen. Jedes Attribut verfügt über einen Namen und einen Wert. Das AWS Database Encryption SDK für DynamoDB verschlüsselt die Werte von Attributen. Dann berechnet er eine Signatur unter Verwendung der Attribute. Sie können festlegen, welche Attributwerte verschlüsselt und welche in die Signatur aufgenommen werden sollen.

Die Verschlüsselung schützt die Vertraulichkeit des Attributwerts. Das Signieren sorgt für die Integrität aller signierten Attribute und deren Beziehung zueinander, und ermöglicht eine Authentifizierung. Es ermöglicht Ihnen, nicht autorisierte Änderungen am gesamten Element zu

erkennen, einschließlich des Hinzufügens oder Löschens von Attributen oder des Ersetzens eines verschlüsselten Werts durch einen anderen.

In einem verschlüsselten Element verbleiben einige Daten im Klartext, einschließlich des Tabellennamens, aller Attributnamen, der Attributwerte, die Sie nicht verschlüsseln, der Namen und Werte der Primärschlüsselattribute (Partitionsschlüssel und Sortierschlüssel) und der Attributtypen. Speichern Sie keine sensiblen Daten in diesen Feldern.

Weitere Informationen zur Funktionsweise des AWS Database Encryption SDK für DynamoDB finden Sie unter. So funktioniert das AWS Database Encryption SDK

> **Note**
>
> Alle Erwähnungen von Attributaktionen in den Themen zum AWS Database Encryption SDK für DynamoDB beziehen sich auf kryptografische Aktionen.

Themen

- Verschlüsseln von Attributwerten
- Signieren des Elements

Verschlüsseln von Attributwerten

Das AWS Database Encryption SDK für DynamoDB verschlüsselt die Werte (aber nicht den Attributnamen oder -typ) der von Ihnen angegebenen Attribute. Um festzulegen, welche Attributwerte verschlüsselt werden, verwenden Sie Attribut-Aktionen.

Das folgende Element beispielsweise enthält die Attribute `example` und `test`.

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

Wenn Sie das Attribut `example` verschlüsseln, aber nicht das Attribut `test`, sehen die Ergebnisse wie folgt aus. Der Wert des verschlüsselten `example`-Attributs ist ein Binärwert anstelle einer Zeichenfolge.

```
'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'
...
```

Die Primärschlüsselattribute — Partitionsschlüssel und Sortierschlüssel — jedes Elements müssen im Klartext bleiben, da DynamoDB sie verwendet, um das Element in der Tabelle zu finden. Sie sollten signiert, aber nicht verschlüsselt werden.

Das AWS Database Encryption SDK für DynamoDB identifiziert die Primärschlüsselattribute für Sie und stellt sicher, dass ihre Werte signiert, aber nicht verschlüsselt sind. Und wenn Sie Ihren Primärschlüssel identifizieren und dann versuchen, ihn zu verschlüsseln, wirft der Client eine Ausnahme auf.

Der Client speichert die Materialbeschreibung in einem neuen Attribut (`aws_dbe_head`), das er dem Artikel hinzufügt. Die Materialbeschreibung beschreibt, wie der Artikel verschlüsselt und signiert wurde. Der Client verwendet diese Informationen, um das Element zu überprüfen und zu entschlüsseln. Das Feld, in dem die Materialbeschreibung gespeichert ist, ist nicht verschlüsselt.

Signieren des Elements

Nach der Verschlüsselung der angegebenen Attributwerte berechnet das AWS Database Encryption SDK for DynamoDB Hash-Based Message Authentication Codes (HMACs) und eine digitale Signatur über die Kanonisierung der Materialbeschreibung, des Verschlüsselungskontextes und jedes mit oder markierten `ENCRYPT_AND_SIGN` Felds in den Attributaktionen. `SIGN_ONLY` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` ECDSA-Signaturen sind standardmäßig aktiviert, aber nicht erforderlich. Der Client speichert die HMACs UND-Signaturen in einem neuen Attribut (`aws_dbe_foot`), das er dem Element hinzufügt.

Durchsuchbare Verschlüsselung in DynamoDB

Um Ihre Amazon DynamoDB-Tabellen für durchsuchbare Verschlüsselung zu konfigurieren, müssen Sie den AWS KMS hierarchischen Schlüsselbund verwenden, um die Datenschlüssel zu generieren, zu verschlüsseln und zu entschlüsseln, die zum Schutz Ihrer Elemente verwendet werden. Sie müssen den auch in Ihre Tabellenverschlüsselungskonfiguration einbeziehen. `SearchConfig`

> **Note**
>
> Wenn Sie die clientseitige Java-Verschlüsselungsbibliothek für DynamoDB verwenden, müssen Sie das AWS Low-Level-Datenbankverschlüsselungs-SDK für DynamoDB-API verwenden, um Ihre Tabellenelemente zu verschlüsseln, zu signieren, zu verifizieren und

zu entschlüsseln. Der DynamoDB Enhanced Client und niedrigere Versionen unterstützen `DynamoDBItemEncryptor` keine durchsuchbare Verschlüsselung.

Themen

Konfiguration sekundärer Indizes mit Beacons

Nachdem Sie Ihre Beacons konfiguriert haben, müssen Sie einen sekundären Index konfigurieren, der die einzelnen Beacons widerspiegelt, bevor Sie nach den verschlüsselten Attributen suchen können.

Wenn Sie einen Standard- oder Verbundbeacon konfigurieren, fügt das AWS Database Encryption SDK dem Beacon-Namen das `aws_dbe_b_` Präfix hinzu, sodass der Server Beacons leicht identifizieren kann. Wenn Sie beispielsweise einen zusammengesetzten Beacon benennen`compoundBeacon`, lautet der vollständige Beacon-Name tatsächlich. `aws_dbe_b_compoundBeacon` Wenn Sie Sekundärindizes konfigurieren möchten, die einen Standard- oder Verbundbeacon enthalten, müssen Sie bei der Identifizierung des Beacon-Namens das `aws_dbe_b_` Präfix angeben.

Schlüssel partitionieren und sortieren

Sie können Primärschlüsselwerte nicht verschlüsseln. Ihre Partitions- und Sortierschlüssel müssen signiert sein. Ihre Primärschlüsselwerte können kein Standard- oder Verbundbeacon sein.

Ihre Primärschlüsselwerte müssen`SIGN_ONLY`, sofern Sie keine `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Attribute angeben, auch die Partitions- und Sortierattribute sein`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

Ihre Primärschlüsselwerte können signierte Beacons sein. Wenn Sie für jeden Ihrer Primärschlüsselwerte unterschiedliche signierte Beacons konfiguriert haben, müssen Sie den Attributnamen, der den Primärschlüsselwert identifiziert, als signierten Beacon-Namen angeben. Das AWS Database Encryption SDK fügt das `aws_dbe_b_` Präfix jedoch nicht zu signierten Beacons hinzu. Selbst wenn Sie unterschiedliche signierte Beacons für Ihre Primärschlüsselwerte konfiguriert haben, müssen Sie bei der Konfiguration eines Sekundärindexes nur die Attributnamen für die Primärschlüsselwerte angeben.

Lokale sekundäre Indizes

Der Sortierschlüssel für einen lokalen Sekundärindex kann ein Beacon sein.

Wenn Sie einen Beacon für den Sortierschlüssel angeben, muss der Typ String sein. Wenn Sie einen Standard- oder Verbundbeacon für den Sortierschlüssel angeben, müssen Sie das `aws_dbe_b_` Präfix angeben, wenn Sie den Beacon-Namen angeben. Wenn Sie einen signierten Beacon angeben, geben Sie den Beacon-Namen ohne Präfix an.

Globale sekundäre Indizes

Sowohl die Partitions- als auch die Sortierschlüssel für einen globalen sekundären Index können Beacons sein.

Wenn Sie einen Beacon für die Partition oder den Sortierschlüssel angeben, muss der Typ String sein. Wenn Sie einen Standard- oder Verbundbeacon für den Sortierschlüssel angeben, müssen Sie das `aws_dbe_b_` Präfix angeben, wenn Sie den Beacon-Namen angeben. Wenn Sie einen signierten Beacon angeben, geben Sie den Beacon-Namen ohne Präfix an.

Attributprojektionen

Eine Projektion ist der Satz von Attributen, die aus einer Tabelle in einen sekundären Index kopiert werden. Der Partitionsschlüssel und der Sortierschlüssel der Tabelle werden immer in den Index projiziert. Sie können andere Attribute projizieren, um die Abfrageanforderungen Ihrer Anwendung zu unterstützen. DynamoDB bietet drei verschiedene Optionen für Attributprojektionen: `KEYS_ONLY`, `INCLUDE`, und. `ALL`

Wenn Sie die INCLUDE-Attributprojektion verwenden, um auf einem Beacon zu suchen, müssen Sie die Namen für alle Attribute angeben, aus denen das Beacon aufgebaut ist, sowie den Beacon-Namen mit dem Präfix. `aws_dbe_b_` Wenn Sie beispielsweise einen Verbundbeacon,, von, `compoundBeacon` und konfiguriert haben `field1` `field2`, müssen Sie `field3`,, und in der `aws_dbe_b_compoundBeacon` `field1` Projektion `field2` angeben. `field3`

Ein globaler sekundärer Index kann nur die in der Projektion explizit angegebenen Attribute verwenden, ein lokaler sekundärer Index kann jedoch jedes beliebige Attribut verwenden.

Beacon-Ausgaben werden getestet

Wenn Sie zusammengesetzte Beacons konfiguriert oder Ihre Beacons mithilfe virtueller Felder erstellt haben, empfehlen wir, zu überprüfen, ob diese Beacons die erwartete Ausgabe erzeugen, bevor Sie Ihre DynamoDB-Tabelle füllen.

Das AWS Database Encryption SDK bietet den `DynamoDbEncryptionTransforms` Service, der Sie bei der Fehlerbehebung bei der Ausgabe virtueller Felder und zusammengesetzter Beacons unterstützt.

Testen virtueller Felder

Der folgende Ausschnitt erstellt Testelemente, definiert den `DynamoDbEncryptionTransforms` Dienst mit der DynamoDB-Tabellenverschlüsselungskonfiguration und zeigt, wie überprüft werden kann, ob das virtuelle Feld die erwartete Ausgabe erzeugt. `ResolveAttributes`

Java

Sehen Sie sich das vollständige Codebeispiel an: .java VirtualBeaconSearchableEncryptionExample

```
// Create test items
final PutItemRequest itemWithHasTestResultPutRequest = PutItemRequest.builder()
    .tableName(ddbTableName)
    .item(itemWithHasTestResult)
    .build();

final PutItemResponse itemWithHasTestResultPutResponse =
 ddb.putItem(itemWithHasTestResultPutRequest);

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

final PutItemResponse itemWithNoHasTestResultPutResponse =
 ddb.putItem(itemWithNoHasTestResultPutRequest);

// Define the DynamoDbEncryptionTransforms service
final DynamoDbEncryptionTransforms trans = DynamoDbEncryptionTransforms.builder()
    .DynamoDbTablesEncryptionConfig(encryptionConfig).build();

// Verify configuration
final ResolveAttributesInput resolveInput = ResolveAttributesInput.builder()
    .TableName(ddbTableName)
    .Item(itemWithHasTestResult)
    .Version(1)
    .build();
final ResolveAttributesOutput resolveOutput = trans.ResolveAttributes(resolveInput);
```

```
// Verify that VirtualFields has the expected value
Map<String, String> vf = new HashMap<>();
vf.put("stateAndHasTestResult", "CAt");
assert resolveOutput.VirtualFields().equals(vf);
```

C# / .NET

Sehen Sie sich das vollständige Codebeispiel an: VirtualBeaconSearchableEncryptionExample.cs.

```
 // Create item with hasTestResult=true
var itemWithHasTestResult = new Dictionary<String, AttributeValue>
{
    ["customer_id"] = new AttributeValue("ABC-123"),
    ["create_time"] = new AttributeValue { N = "1681495205" },
    ["state"] = new AttributeValue("CA"),
    ["hasTestResult"] = new AttributeValue { BOOL = true }
};

// Create item with hasTestResult=false
var itemWithNoHasTestResult = new Dictionary<String, AttributeValue>
{
    ["customer_id"] = new AttributeValue("DEF-456"),
    ["create_time"] = new AttributeValue { N = "1681495205" },
    ["state"] = new AttributeValue("CA"),
    ["hasTestResult"] = new AttributeValue { BOOL = false }
};

// Define the DynamoDbEncryptionTransforms service
var trans = new DynamoDbEncryptionTransforms(encryptionConfig);

// Verify configuration
var resolveInput = new ResolveAttributesInput
{
    TableName = ddbTableName,
    Item = itemWithHasTestResult,
    Version = 1
};
var resolveOutput = trans.ResolveAttributes(resolveInput);

// Verify that VirtualFields has the expected value
Debug.Assert(resolveOutput.VirtualFields.Count == 1);
```

```
Debug.Assert(resolveOutput.VirtualFields["stateAndHasTestResult"] == "CAt");
```

Rust

Sehen Sie sich das vollständige Codebeispiel an: virtual_beacon_searchable_encryption.rs.

```
// Create item with hasTestResult=true
let item_with_has_test_result = HashMap::from([
    (
        "customer_id".to_string(),
        AttributeValue::S("ABC-123".to_string()),
    ),
    (
        "create_time".to_string(),
        AttributeValue::N("1681495205".to_string()),
    ),
    ("state".to_string(), AttributeValue::S("CA".to_string())),
    ("hasTestResult".to_string(), AttributeValue::Bool(true)),
]);

// Create item with hasTestResult=false
let item_with_no_has_test_result = HashMap::from([
    (
        "customer_id".to_string(),
        AttributeValue::S("DEF-456".to_string()),
    ),
    (
        "create_time".to_string(),
        AttributeValue::N("1681495205".to_string()),
    ),
    ("state".to_string(), AttributeValue::S("CA".to_string())),
    ("hasTestResult".to_string(), AttributeValue::Bool(false)),
]);

// Define the transform service
let trans = transform_client::Client::from_conf(encryption_config.clone())?;

// Verify the configuration
let resolve_output = trans
    .resolve_attributes()
    .table_name(ddb_table_name)
    .item(item_with_has_test_result.clone())
    .version(1)
    .send()
```

```
    .await?;

// Verify that VirtualFields has the expected value
let virtual_fields = resolve_output.virtual_fields.unwrap();
assert_eq!(virtual_fields.len(), 1);
assert_eq!(virtual_fields["stateAndHasTestResult"], "CAt");
```

Testen von Compound-Beacons

Der folgende Ausschnitt erstellt ein Testelement, definiert den `DynamoDbEncryptionTransforms` Dienst mit der DynamoDB-Tabellenverschlüsselungskonfiguration und zeigt, wie überprüft werden kann, ob der Compound Beacon die erwartete Ausgabe erzeugt. `ResolveAttributes`

Java

Sehen Sie sich das vollständige Codebeispiel an: .java CompoundBeaconSearchableEncryptionExample

```
// Create an item with both attributes used in the compound beacon.
final HashMap<String, AttributeValue> item = new HashMap<>();
item.put("work_id", AttributeValue.builder().s("9ce39272-8068-4efd-a211-
cd162ad65d4c").build());
item.put("inspection_date", AttributeValue.builder().s("2023-06-13").build());
item.put("inspector_id_last4", AttributeValue.builder().s("5678").build());
item.put("unit", AttributeValue.builder().s("011899988199").build());

// Define the DynamoDbEncryptionTransforms service
final DynamoDbEncryptionTransforms trans = DynamoDbEncryptionTransforms.builder()
    .DynamoDbTablesEncryptionConfig(encryptionConfig).build();

// Verify configuration
final ResolveAttributesInput resolveInput = ResolveAttributesInput.builder()
    .TableName(ddbTableName)
    .Item(item)
    .Version(1)
    .build();

final ResolveAttributesOutput resolveOutput = trans.ResolveAttributes(resolveInput);

// Verify that CompoundBeacons has the expected value
Map<String, String> cbs = new HashMap<>();
cbs.put("last4UnitCompound", "L-5678.U-011899988199");
```

```
assert resolveOutput.CompoundBeacons().equals(cbs);
// Note : the compound beacon actually stored in the table is not
 "L-5678.U-011899988199"
// but rather something like "L-abc.U-123", as both parts are EncryptedParts
// and therefore the text is replaced by the associated beacon
```

C# / .NET

Sehen Sie sich das vollständige Codebeispiel an: .cs CompoundBeaconSearchableEncryptionExample

```
// Create an item with both attributes used in the compound beacon
var item = new Dictionary<String, AttributeValue>
{
    ["work_id"] = new AttributeValue("9ce39272-8068-4efd-a211-cd162ad65d4c"),
    ["inspection_date"] = new AttributeValue("2023-06-13"),
    ["inspector_id_last4"] = new AttributeValue("5678"),
    ["unit"] = new AttributeValue("011899988199")
};

// Define the DynamoDbEncryptionTransforms service
var trans = new DynamoDbEncryptionTransforms(encryptionConfig);

// Verify configuration
var resolveInput = new ResolveAttributesInput
{
    TableName = ddbTableName,
    Item = item,
    Version = 1
};
var resolveOutput = trans.ResolveAttributes(resolveInput);


// Verify that CompoundBeacons has the expected value
Debug.Assert(resolveOutput.CompoundBeacons.Count == 1);
Debug.Assert(resolveOutput.CompoundBeacons["last4UnitCompound"] ==
 "L-5678.U-011899988199");
// Note : the compound beacon actually stored in the table is not
 "L-5678.U-011899988199"
// but rather something like "L-abc.U-123", as both parts are EncryptedParts
// and therefore the text is replaced by the associated beacon
```

Rust

Sehen Sie sich das vollständige Codebeispiel an: compound_beacon_searchable_encryption.rs

```
// Create an item with both attributes used in the compound beacon
let item = HashMap::from([
    (
        "work_id".to_string(),
        AttributeValue::S("9ce39272-8068-4efd-a211-cd162ad65d4c".to_string()),
    ),
    (
        "inspection_date".to_string(),
        AttributeValue::S("2023-06-13".to_string()),
    ),
    (
        "inspector_id_last4".to_string(),
        AttributeValue::S("5678".to_string()),
    ),
    (
        "unit".to_string(),
        AttributeValue::S("011899988199".to_string()),
    ),
]);

// Define the transforms service
let trans = transform_client::Client::from_conf(encryption_config.clone())?;

// Verify configuration
let resolve_output = trans
    .resolve_attributes()
    .table_name(ddb_table_name)
    .item(item.clone())
    .version(1)
    .send()
    .await?;

// Verify that CompoundBeacons has the expected value
let compound_beacons = resolve_output.compound_beacons.unwrap();
assert_eq!(compound_beacons.len(), 1);
assert_eq!(
    compound_beacons["last4UnitCompound"],
    "L-5678.U-011899988199"
);
// but rather something like "L-abc.U-123", as both parts are EncryptedParts
```

```
// and therefore the text is replaced by the associated beacon
```

Aktualisierung Ihres Datenmodells

Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt . AWS Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Wenn Sie das AWS Database Encryption SDK für DynamoDB konfigurieren, geben Sie Attributaktionen an. Beim Verschlüsseln identifiziert das AWS Database Encryption SDK anhand der Attributaktionen, welche Attribute verschlüsselt und signiert, welche Attribute signiert (aber nicht verschlüsselt) und welche ignoriert werden sollen. Sie definieren auch zulässige unsignierte Attribute, um dem Client explizit mitzuteilen, welche Attribute von den Signaturen ausgeschlossen sind. Beim Entschlüsseln verwendet das AWS Database Encryption SDK die erlaubten unsignierten Attribute, die Sie definiert haben, um zu identifizieren, welche Attribute nicht in den Signaturen enthalten sind. Attributaktionen werden nicht im verschlüsselten Element gespeichert und das AWS Database Encryption SDK aktualisiert Ihre Attributaktionen nicht automatisch.

Wählen Sie Ihre Attributaktionen sorgfältig aus. Verwenden Sie im Zweifelsfall Verschlüsseln und signieren. Nachdem Sie das AWS Database Encryption SDK zum Schutz Ihrer Elemente verwendet haben, können Sie ein `ENCRYPT_AND_SIGN` vorhandenes oder `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` -Attribut nicht mehr in ändern`DO_NOTHING`. `SIGN_ONLY` Sie können jedoch ohne Bedenken die folgenden Änderungen vornehmen.

- Fügen Sie die neuen Attribute `ENCRYPT_AND_SIGN`, `SIGN_ONLY` und `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` hinzu
- Entfernen Sie vorhandene Attribute
- Ändern Sie ein vorhandenes `ENCRYPT_AND_SIGN`-Attribut in `SIGN_ONLY` oder `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`
- Ändern Sie ein vorhandenes `SIGN_ONLY`- oder `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`-Attribut in `ENCRYPT_AND_SIGN`
- Fügen Sie ein neues `DO_NOTHING`-Attribut hinzu
- Ändern Sie ein vorhandenes `SIGN_ONLY`-Attribut in `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`

- Ändern Sie ein vorhandenes SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT-Attribut in SIGN_ONLY

Überlegungen zur durchsuchbaren Verschlüsselung

Bevor Sie Ihr Datenmodell aktualisieren, sollten Sie sorgfältig überlegen, wie sich Ihre Aktualisierungen auf Beacons auswirken könnten, die Sie anhand der Attribute erstellt haben. Nachdem Sie mit einem Beacon neue Datensätze geschrieben haben, können Sie die Konfiguration des Beacons nicht mehr aktualisieren. Sie können die Attributaktionen, die den Attributen zugeordnet sind, die Sie zum Aufbau von Beacons verwendet haben, nicht aktualisieren. Wenn Sie ein vorhandenes Attribut und den zugehörigen Beacon entfernen, können Sie mit diesem Beacon keine vorhandenen Datensätze abfragen. Sie können neue Beacons für neue Felder erstellen, die Sie Ihrem Datensatz hinzufügen, aber Sie können bestehende Beacons nicht so aktualisieren, dass sie das neue Feld aufnehmen.

Überlegungen zu den Attributen SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT

Standardmäßig sind die Partitions- und Sortierschlüssel die einzigen Attribute, die im Verschlüsselungskontext enthalten sind. Sie könnten erwägen, zusätzliche Felder zu definieren, `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` damit der Anbieter der Branch-Schlüssel-ID für Ihren AWS KMS hierarchischen Schlüsselbund ermitteln kann, welcher Filialschlüssel für die Entschlüsselung aus dem Verschlüsselungskontext erforderlich ist. Weitere Informationen finden Sie unter Lieferant für die Filialschlüssel-ID. Wenn Sie `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Attribute angeben, müssen auch die Partitions- und Sortierattribute angegeben werden`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

> **Note**
>
> Um die `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` kryptografische Aktion verwenden zu können, müssen Sie Version 3.3 oder höher des AWS Database Encryption SDK verwenden. Stellen Sie die neue Version für alle Leser bereit, bevor Sie Ihr Datenmodell so aktualisieren, dass es diese enthält`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

Fügen Sie die neuen Attribute ENCRYPT_AND_SIGN, SIGN_ONLY und SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT hinzu

Um ein neues, oder Attribut hinzuzufügen, definieren Sie das neue Attribut in Ihren Attributaktionen. ENCRYPT_AND_SIGN SIGN_ONLY SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT

Sie können ein vorhandenes DO_NOTHING Attribut nicht entfernen und es alsENCRYPT_AND_SIGN,SIGN_ONLY, SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT -Attribut wieder hinzufügen.

Verwenden einer Datenklasse mit Anmerkungen

Wenn Sie Ihre Attributaktionen mit a definiert habenTableSchema, fügen Sie das neue Attribut Ihrer annotierten Datenklasse hinzu. Wenn Sie keine Attributaktions-Anmerkung für das neue Attribut angeben, verschlüsselt und signiert der Client das neue Attribut standardmäßig (es sei denn, das Attribut ist Teil des Primärschlüssels). Wenn Sie nur das neue Attribut signieren möchten, müssen Sie das neue Attribut mit der @DynamoDbEncryptionSignAndIncludeInEncryptionContext Anmerkung @DynamoDbEncryptionSignOnly oder hinzufügen.

Verwenden Sie ein Objektmodell

Wenn Sie Ihre Attributaktionen manuell definiert haben, fügen Sie das neue Attribut zu den Attributaktionen in Ihrem Objektmodell hinzu und geben Sie ENCRYPT_AND_SIGNSIGN_ONLY, oder SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT als Attributaktion an.

Entfernen Sie vorhandene Attribute

Wenn Sie entscheiden, dass Sie ein Attribut nicht mehr benötigen, können Sie das Schreiben von Daten in dieses Attribut beenden oder es formell aus Ihren Attributaktionen entfernen. Wenn Sie aufhören, neue Daten in ein Attribut zu schreiben, wird das Attribut weiterhin in Ihren Attributaktionen angezeigt. Dies kann hilfreich sein, wenn Sie das Attribut in future erneut verwenden müssen. Wenn Sie das Attribut formal aus Ihren Attributaktionen entfernen, wird es nicht aus Ihrem Datensatz entfernt. Ihr Datensatz wird weiterhin Elemente enthalten, die dieses Attribut enthalten.

Um ein vorhandenesENCRYPT_AND_SIGN, SIGN_ONLYSIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT, oder DO_NOTHING Attribut formell zu entfernen, aktualisieren Sie Ihre Attributaktionen.

Wenn Sie ein DO_NOTHING Attribut entfernen, dürfen Sie dieses Attribut nicht aus Ihren zulässigen Attributen ohne Vorzeichen entfernen. Auch wenn Sie keine neuen Werte mehr in dieses Attribut schreiben, muss der Client trotzdem wissen, dass das Attribut vorzeichenlos ist, um vorhandene Elemente lesen zu können, die das Attribut enthalten.

Verwenden einer Datenklasse mit Anmerkungen

Wenn Sie Ihre Attributaktionen mit a definiert habenTableSchema, entfernen Sie das Attribut aus Ihrer annotierten Datenklasse.

Verwenden Sie ein Objektmodell

Wenn Sie Ihre Attributaktionen manuell definiert haben, entfernen Sie das Attribut aus den Attributaktionen in Ihrem Objektmodell.

Ändern Sie ein vorhandenes ENCRYPT_AND_SIGN-Attribut in SIGN_ONLY oder SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT

Um ein vorhandenes ENCRYPT_AND_SIGN Attribut in SIGN_ONLY SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT oder zu ändern, müssen Sie Ihre Attributaktionen aktualisieren. Nachdem Sie das Update bereitgestellt haben, kann der Client vorhandene Werte, die in das Attribut geschrieben wurden, verifizieren und entschlüsseln, signiert jedoch nur neue Werte, die in das Attribut geschrieben wurden.

> **Note**
>
> Überlegen Sie sich sorgfältig Ihre Sicherheitsanforderungen, bevor Sie ein vorhandenes ENCRYPT_AND_SIGN Attribut in SIGN_ONLY oder SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT ändern. Jedes Attribut, das sensible Daten speichern kann, sollte verschlüsselt werden.

Verwendung einer annotierten Datenklasse

Wenn Sie Ihre Attributaktionen mit einem definiert habenTableSchema, aktualisieren Sie das vorhandene Attribut, sodass die @DynamoDbEncryptionSignAndIncludeInEncryptionContext Anmerkung @DynamoDbEncryptionSignOnly oder in Ihre annotierte Datenklasse aufgenommen wird.

Verwenden Sie ein Objektmodell

Wenn Sie Ihre Attributaktionen manuell definiert haben, aktualisieren Sie die dem vorhandenen Attribut zugeordnete Attributaktion von `ENCRYPT_AND_SIGN` bis `SIGN_ONLY` oder `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` in Ihrem Objektmodell.

Ändern Sie ein vorhandenes `SIGN_ONLY`- oder `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`-Attribut in `ENCRYPT_AND_SIGN`

Um ein vorhandenes Attribut `SIGN_ONLY` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` oder ein Attribut in zu ändern, `ENCRYPT_AND_SIGN` müssen Sie Ihre Attributaktionen aktualisieren. Nachdem Sie das Update bereitgestellt haben, kann der Client die vorhandenen Werte überprüfen, die in das Attribut geschrieben wurden, und verschlüsselt und signiert neue Werte, die in das Attribut geschrieben wurden.

Verwenden einer Datenklasse mit Anmerkungen

Wenn Sie Ihre Attributaktionen mit einem definiert haben`TableSchema`, entfernen Sie die `@DynamoDbEncryptionSignAndIncludeInEncryptionContext` Anmerkung `@DynamoDbEncryptionSignOnly` oder aus dem vorhandenen Attribut.

Verwenden eines Objektmodells

Wenn Sie Ihre Attributaktionen manuell definiert haben, aktualisieren Sie die Attributaktion, die dem Attribut von `SIGN_ONLY` oder `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` nach zugeordnet ist, `ENCRYPT_AND_SIGN` in Ihrem Objektmodell.

Fügen Sie ein neues **`DO_NOTHING`-Attribut** hinzu

Um das Fehlerrisiko beim Hinzufügen eines neuen `DO_NOTHING` Attributs zu verringern, empfehlen wir, bei der Benennung Ihrer `DO_NOTHING` Attribute ein eindeutiges Präfix anzugeben und dieses Präfix dann zu verwenden, um Ihre zulässigen Attribute ohne Vorzeichen zu definieren.

Sie können kein `ENCRYPT_AND_SIGN` vorhandenes `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Attribut oder aus Ihrer annotierten Datenklasse entfernen und das Attribut dann wieder als `DO_NOTHING` Attribut hinzufügen. `SIGN_ONLY` Sie können nur völlig neue `DO_NOTHING` Attribute hinzufügen.

Die Schritte, die Sie unternehmen, um ein neues DO_NOTHING Attribut hinzuzufügen, hängen davon ab, ob Sie Ihre zulässigen Attribute ohne Vorzeichen explizit in einer Liste oder mit einem Präfix definiert haben.

Verwenden Sie ein zulässiges Präfix für vorzeichenlose Attribute

Wenn Sie Ihre Attributaktionen mit einem definiert habenTableSchema, fügen Sie das neue DO_NOTHING Attribut mit der Anmerkung zu Ihrer annotierten Datenklasse hinzu. @DynamoDbEncryptionDoNothing Wenn Sie Ihre Attributaktionen manuell definiert haben, aktualisieren Sie Ihre Attributaktionen, sodass sie das neue Attribut enthalten. Achten Sie darauf, das neue Attribut explizit mit der DO_NOTHING Attributaktion zu konfigurieren. Sie müssen dasselbe eindeutige Präfix in den Namen des neuen Attributs aufnehmen.

Verwenden Sie eine Liste mit zulässigen Attributen ohne Vorzeichen

1. Fügen Sie das neue DO_NOTHING Attribut zu Ihrer Liste der zulässigen unsignierten Attribute hinzu und stellen Sie die aktualisierte Liste bereit.
2. Stellen Sie die Änderung aus Schritt 1 bereit.

   Sie können erst mit Schritt 3 fortfahren, wenn die Änderung auf alle Hosts übertragen wurde, die diese Daten lesen müssen.
3. Fügen Sie das neue DO_NOTHING Attribut zu Ihren Attributaktionen hinzu.

   a. Wenn Sie Ihre Attributaktionen mit einem definiert habenTableSchema, fügen Sie das neue DO_NOTHING Attribut mit der @DynamoDbEncryptionDoNothing Anmerkung zu Ihrer annotierten Datenklasse hinzu.

   b. Wenn Sie Ihre Attributaktionen manuell definiert haben, aktualisieren Sie Ihre Attributaktionen, sodass sie das neue Attribut enthalten. Achten Sie darauf, das neue Attribut explizit mit der DO_NOTHING Attributaktion zu konfigurieren.
4. Stellen Sie die Änderung aus Schritt 3 bereit.

Ändern Sie ein vorhandenes SIGN_ONLY-Attribut in SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT

Um ein vorhandenes Attribut in zu ändern, müssen Sie Ihre Attributaktionen aktualisieren. SIGN_ONLY SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT Nachdem Sie das Update bereitgestellt haben, kann der Client die vorhandenen Werte überprüfen, die in das Attribut

geschrieben wurden, und signiert weiterhin neue Werte, die in das Attribut geschrieben wurden. Neue Werte, die in das Attribut geschrieben werden, werden in den Verschlüsselungskontext aufgenommen.

Wenn Sie `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Attribute angeben, müssen auch die Partitions- und Sortierattribute angegeben werden`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

Verwenden Sie eine annotierte Datenklasse

Wenn Sie Ihre Attributaktionen mit a definiert haben`TableSchema`, aktualisieren Sie die dem Attribut zugeordnete Attributaktion von `@DynamoDbEncryptionSignOnly` bis`@DynamoDbEncryptionSignAndIncludeInEncryptionContext`.

Verwenden Sie ein Objektmodell

Wenn Sie Ihre Attributaktionen manuell definiert haben, aktualisieren Sie die dem Attribut zugeordnete Attributaktion von `SIGN_ONLY` bis `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` in Ihrem Objektmodell.

Ändern Sie ein vorhandenes SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT-Attribut in SIGN_ONLY

Um ein vorhandenes Attribut in zu ändern, müssen Sie Ihre Attributaktionen aktualisieren. `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` `SIGN_ONLY` Nachdem Sie das Update bereitgestellt haben, kann der Client die vorhandenen Werte überprüfen, die in das Attribut geschrieben wurden, und signiert weiterhin neue Werte, die in das Attribut geschrieben wurden. Neue Werte, die in das Attribut geschrieben werden, werden nicht in den Verschlüsselungskontext aufgenommen.

Bevor Sie ein vorhandenes `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Attribut in ändern`SIGN_ONLY`, sollten Sie sorgfältig abwägen, wie sich Ihre Aktualisierungen auf die Funktionalität Ihres Branch Key ID-Anbieters auswirken könnten.

Verwenden einer Datenklasse mit Anmerkungen

Wenn Sie Ihre Attributaktionen mit a definiert haben`TableSchema`, aktualisieren Sie die dem Attribut zugeordnete Attributaktion von

`@DynamoDbEncryptionSignAndIncludeInEncryptionContext` bis`@DynamoDbEncryptionSignOnly`.

Verwenden Sie ein Objektmodell

Wenn Sie Ihre Attributaktionen manuell definiert haben, aktualisieren Sie die dem Attribut zugeordnete Attributaktion von `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` bis `SIGN_ONLY` in Ihrem Objektmodell.

AWS Database Encryption SDK für DynamoDB, verfügbare Programmiersprachen

Das AWS Database Encryption SDK für DynamoDB ist für die folgenden Programmiersprachen verfügbar. Die sprachspezifischen Bibliotheken sind unterschiedlich, aber die daraus resultierenden Implementierungen sind interoperabel. Sie können mit einer Sprachimplementierung verschlüsseln und mit einer anderen entschlüsseln. Die Interoperabilität ist möglicherweise von Spracheinschränkungen abhängig. Wenn dies der Fall ist, werden diese Einschränkungen im Thema zur Sprachimplementierung beschrieben.

Themen

- Java
- .NET
- Rust

Java

> Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt . AWS Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

In diesem Thema wird erklärt, wie Version 4 installiert und verwendet wird. x der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB. Einzelheiten zur Programmierung mit dem AWS Database Encryption SDK für DynamoDB finden Sie in den Java-Beispielen im aws-database-encryption-sdk-dynamodb-Repository unter. GitHub

> **Note**
>
> Die folgenden Themen konzentrieren sich auf Version 4. x der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB.
> Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt. AWS Das AWS Database Encryption SDK unterstützt weiterhin ältere Versionen des DynamoDB Encryption Client.

Themen

Voraussetzungen

Bevor Sie Version 4 installieren. x der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB, stellen Sie sicher, dass Sie die folgenden Voraussetzungen erfüllen.

Eine Java-Entwicklungsumgebung

Sie benötigen Java 8 oder höher. Klicken Sie auf der Oracle-Website auf Java SE Downloads und laden und installieren Sie anschließend das Java SE Development Kit (JDK).

Wenn Sie das Oracle JDK verwenden, müssen Sie auch die Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files herunterladen und installieren.

AWS SDK for Java 2.x

Das AWS Database Encryption SDK für DynamoDB erfordert das DynamoDB Enhanced Client-Modul von. AWS SDK for Java 2.x Sie können das gesamte SDK oder nur dieses Modul installieren.

Informationen zur Aktualisierung Ihrer Version von finden Sie unter Migration von Version 1.x auf 2.x von. AWS SDK für Java AWS SDK für Java

Das AWS SDK für Java ist über Apache Maven verfügbar. Sie können eine Abhängigkeit für das gesamte AWS SDK für Java Modul oder nur für das `dynamodb-enhanced` Modul deklarieren.

Installieren Sie das AWS SDK für Java mit Apache Maven

- Um das gesamte AWS SDK für Java als Abhängigkeit zu importieren, deklarieren Sie es in Ihrer `pom.xml`-Datei.
- Um eine Abhängigkeit nur für das Amazon DynamoDB DynamoDB-Modul in der zu erstellen AWS SDK für Java, folgen Sie den Anweisungen zur Angabe bestimmter Module. Stellen Sie „`groupId`bis“ `software.amazon.awssdk` und „Bis“ ein. `artifactID` `dynamodb-enhanced`

> **Note**
>
> Wenn Sie den AWS KMS Schlüsselbund oder den AWS KMS hierarchischen Schlüsselbund verwenden, müssen Sie auch eine Abhängigkeit für das Modul erstellen. AWS KMS Stellen Sie „bis“ `software.amazon.awssdk` und „`groupId`Bis“ ein. `artifactID` `kms`

Installation

Sie können Version 4 installieren. x der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB auf folgende Weise.

Verwenden von Apache Maven

Der Amazon DynamoDB Encryption Client für Java ist über Apache Maven mit der folgenden Abhängigkeitsdefinition verfügbar.

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

Verwenden von Gradle Kotlin

Sie können Gradle verwenden, um eine Abhängigkeit vom Amazon DynamoDB Encryption Client for Java zu deklarieren, indem Sie Folgendes zum Abschnitt mit den Abhängigkeiten Ihres Gradle-Projekts hinzufügen.

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

manuell

Um die clientseitige Java-Verschlüsselungsbibliothek für DynamoDB zu installieren, klonen Sie das aws-database-encryption-sdk-dynamodb-Repository oder laden Sie es herunter. GitHub

Schauen Sie sich nach der Installation des SDK zunächst den Beispielcode in diesem Handbuch und die Java-Beispiele im aws-database-encryption-sdk-dynamodb-Repository an. GitHub

Verwendung der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB

Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt . AWS Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

In diesem Thema werden einige der Funktionen und Hilfsklassen in Version 4 erklärt. x der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB.

Einzelheiten zur Programmierung mit der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB finden Sie in den Java-Beispielen, den Java-Beispielen im aws-database-encryption-sdk-dynamodb-Repository auf. GitHub

Themen

- Elementverschlüssler
- Attributaktionen in der AWS Datenbankverschlüsselungs-SDK für DynamoDB
- Verschlüsselungskonfiguration in der AWS Datenbankverschlüsselungs-SDK für DynamoDB
- Aktualisierung von Elementen mit dem AWS SDK für Datenbankverschlüsselung
- Signierte Sets entschlüsseln

Elementverschlüssler

Im Kern ist das AWS Database Encryption SDK für DynamoDB ein Elementverschlüsseler. Sie können Version 4 verwenden. x der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB, um Ihre DynamoDB-Tabellenelemente auf folgende Weise zu verschlüsseln, zu signieren, zu verifizieren und zu entschlüsseln.

Der erweiterte DynamoDB-Client

Sie können den DynamoDB Enhanced Client so konfigurieren, `DynamoDbEncryptionInterceptor` dass er Elemente automatisch clientseitig mit Ihren DynamoDB-Anfragen verschlüsselt und signiert. `PutItem` Mit dem DynamoDB Enhanced Client können Sie Ihre Attributaktionen mithilfe einer annotierten Datenklasse definieren. Wir empfehlen, wann immer möglich den DynamoDB Enhanced Client zu verwenden.

Der DynamoDB Enhanced Client unterstützt keine durchsuchbare Verschlüsselung.

> **Note**
>
> Das AWS Database Encryption SDK unterstützt keine Anmerkungen zu verschachtelten Attributen.

Die DynamoDB-API auf niedriger Ebene

Sie können die Low-Level-DynamoDB-API so konfigurieren, `DynamoDbEncryptionInterceptor` dass Elemente automatisch clientseitig mit Ihren DynamoDB-Anfragen verschlüsselt und signiert werden. `PutItem`

Sie müssen die Low-Level-DynamoDB-API verwenden, um durchsuchbare Verschlüsselung zu verwenden.

Die untergeordnete Ebene **`DynamoDbItemEncryptor`**

Die untergeordnete Ebene verschlüsselt und signiert oder entschlüsselt und verifiziert Ihre Tabellenelemente `DynamoDbItemEncryptor` direkt, ohne DynamoDB aufzurufen. Es stellt keine DynamoDB `PutItem` oder `GetItem` Anfragen. Sie können beispielsweise die untergeordnete Ebene verwenden, `DynamoDbItemEncryptor` um ein DynamoDB-Element, das Sie bereits abgerufen haben, direkt zu entschlüsseln und zu verifizieren.

Die untergeordnete Ebene unterstützt keine durchsuchbare VerschlüsselungDynamoDbItemEncryptor.

Attributaktionen in der AWS Datenbankverschlüsselungs-SDK für DynamoDB

Attributaktionen bestimmen, welche Attributwerte verschlüsselt und signiert werden, welche nur signiert sind, welche signiert und in den Verschlüsselungskontext aufgenommen werden und welche ignoriert werden.

> **Note**
>
> Um die `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` kryptografische Aktion verwenden zu können, müssen Sie Version 3.3 oder höher des AWS Database Encryption SDK verwenden. Stellen Sie die neue Version für alle Lesegeräte bereit, bevor Sie Ihr Datenmodell so aktualisieren, dass es diese enthält`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

Wenn Sie die Low-Level-DynamoDB-API oder die Low-Level-API verwenden`DynamoDbItemEncryptor`, müssen Sie Ihre Attributaktionen manuell definieren. Wenn Sie den DynamoDB Enhanced Client verwenden, können Sie Ihre Attributaktionen entweder manuell definieren oder Sie können eine annotierte Datenklasse verwenden, um eine zu generieren. `TableSchema` Um den Konfigurationsprozess zu vereinfachen, empfehlen wir die Verwendung einer annotierten Datenklasse. Wenn Sie eine annotierte Datenklasse verwenden, müssen Sie Ihr Objekt nur einmal modellieren.

> **Note**
>
> Nachdem Sie Ihre Attributaktionen definiert haben, müssen Sie definieren, welche Attribute von den Signaturen ausgeschlossen werden. Um das future Hinzufügen neuer Attribute ohne Vorzeichen zu vereinfachen, empfehlen wir, ein eindeutiges Präfix (wie `":"`) zu wählen, um Ihre vorzeichenlosen Attribute zu identifizieren. Nehmen Sie dieses Präfix in den Attributnamen für alle Attribute auf, die Sie bei der Definition Ihres DynamoDB-Schemas und Ihrer Attributaktionen markiert `DO_NOTHING` haben.

Verwenden Sie eine Datenklasse mit Anmerkungen

Verwenden Sie eine annotierte Datenklasse, um Ihre Attributaktionen mit dem DynamoDB Enhanced Client und zu spezifizieren. `DynamoDbEncryptionInterceptor` Das AWS Database Encryption SDK für DynamoDB verwendet die standardmäßigen DynamoDB-Attributanmerkungen, die den Attributtyp definieren, um zu bestimmen, wie ein Attribut geschützt werden soll. Standardmäßig sind alle Attribute verschlüsselt und signiert, mit Ausnahme der Primärschlüssel, die zwar signiert, aber nicht verschlüsselt sind.

> **Note**
>
> Um die `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` kryptografische Aktion verwenden zu können, müssen Sie Version 3.3 oder höher des Database Encryption SDK verwenden. AWS Stellen Sie die neue Version für alle Lesegeräte bereit, bevor Sie Ihr Datenmodell so aktualisieren, dass es diese enthält`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

Weitere Hinweise zu SimpleClass.javaden DynamoDB Enhanced Client-Anmerkungen finden Sie im aws-database-encryption-sdk-dynamodb-Repository unter GitHub .

Standardmäßig sind Primärschlüsselattribute signiert, aber nicht verschlüsselt (), und alle anderen Attribute sind verschlüsselt und signiert (). `SIGN_ONLY` `ENCRYPT_AND_SIGN` Wenn Sie Attribute als definieren`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, müssen dies auch die Partitions- und Sortierattribute sein`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Um Ausnahmen anzugeben, verwenden Sie die Verschlüsselungsanmerkungen, die in der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB definiert sind. Wenn Sie beispielsweise möchten, dass ein bestimmtes Attribut nur signiert wird, verwenden Sie die Anmerkung. `@DynamoDbEncryptionSignOnly` Wenn Sie möchten, dass ein bestimmtes Attribut signiert und in den Verschlüsselungskontext aufgenommen wird, verwenden Sie die`@DynamoDbEncryptionSignAndIncludeInEncryptionContext`. Wenn Sie möchten, dass ein bestimmtes Attribut weder signiert noch verschlüsselt (`DO_NOTHING`) wird, verwenden Sie die `@DynamoDbEncryptionDoNothing` Anmerkung.

> **Note**
>
> Das AWS Database Encryption SDK unterstützt keine Anmerkungen zu verschachtelten Attributen.

Das folgende Beispiel zeigt die Anmerkungen, die zur Definition von `ENCRYPT_AND_SIGNSIGN_ONLY`, und `DO_NOTHING` Attributaktionen verwendet werden. Ein Beispiel, das die zur Definition verwendeten Anmerkungen zeigt`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, finden Sie unter. SimpleClass4.java

```
@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;
    }

}
```

Verwenden Sie Ihre annotierte Datenklasse, um die zu erstellen, `TableSchema` wie im folgenden Codeausschnitt gezeigt.

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

Definieren Sie Ihre Attributaktionen manuell

Um Attributaktionen manuell zu spezifizieren, erstellen Sie ein `Map` Objekt, in dem die Name-Wert-Paare für Attributnamen und die angegebenen Aktionen stehen.

Geben Sie `ENCRYPT_AND_SIGN` an, dass ein Attribut verschlüsselt und signiert werden soll. Geben Sie `SIGN_ONLY` an, dass ein Attribut signiert, aber nicht verschlüsselt werden soll. Geben Sie `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` an, dass ein Attribut signiert und in den Verschlüsselungskontext aufgenommen werden soll. Sie können ein Attribut nicht verschlüsseln, ohne es auch zu signieren. Geben Sie `DO_NOTHING` an, ob ein Attribut ignoriert werden soll.

Die Partitions- und Sortierattribute müssen entweder `SIGN_ONLY` oder lauten`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Wenn Sie Attribute als definieren`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, müssen dies auch die Partitions- und Sortierattribute sein`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

> **Note**
>
> Um die `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` kryptografische Aktion verwenden zu können, müssen Sie Version 3.3 oder höher des AWS Database Encryption SDK verwenden. Stellen Sie die neue Version für alle Lesegeräte bereit, bevor Sie Ihr Datenmodell so aktualisieren, dass es diese enthält`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

```
final Map<String, CryptoAction> attributeActionsOnEncrypt = new HashMap<>();
// The partition attribute must be signed
attributeActionsOnEncrypt.put("partition_key",
 CryptoAction.SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT);
// The sort attribute must be signed
attributeActionsOnEncrypt.put("sort_key",
 CryptoAction.SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT);
attributeActionsOnEncrypt.put("attribute1", CryptoAction.ENCRYPT_AND_SIGN);
attributeActionsOnEncrypt.put("attribute2", CryptoAction.SIGN_ONLY);
attributeActionsOnEncrypt.put("attribute3",
 CryptoAction.SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT);
attributeActionsOnEncrypt.put(":attribute4", CryptoAction.DO_NOTHING);
```

Verschlüsselungskonfiguration in der AWS Datenbankverschlüsselungs-SDK für DynamoDB

Wenn Sie das AWS Database Encryption SDK verwenden, müssen Sie explizit eine Verschlüsselungskonfiguration für Ihre DynamoDB-Tabelle definieren. Die in Ihrer Verschlüsselungskonfiguration erforderlichen Werte hängen davon ab, ob Sie Ihre Attributaktionen manuell oder mit einer annotierten Datenklasse definiert haben.

Der folgende Ausschnitt definiert eine DynamoDB-Tabellenverschlüsselungskonfiguration unter Verwendung des DynamoDB Enhanced Client und erlaubte unsignierte Attribute `TableSchema`, die durch ein eindeutiges Präfix definiert sind.

```
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());
```

Logischer Tabellenname

Ein logischer Tabellenname für Ihre DynamoDB-Tabelle.

Der logische Tabellenname ist kryptografisch an alle in der Tabelle gespeicherten Daten gebunden, um DynamoDB-Wiederherstellungsvorgänge zu vereinfachen. Es wird dringend empfohlen, Ihren DynamoDB-Tabellennamen als logischen Tabellennamen anzugeben, wenn Sie Ihre Verschlüsselungskonfiguration zum ersten Mal definieren. Sie müssen immer denselben logischen Tabellennamen angeben. Damit die Entschlüsselung erfolgreich ist, muss der Name der logischen Tabelle mit dem Namen übereinstimmen, der bei der Verschlüsselung angegeben wurde. Falls sich Ihr DynamoDB-Tabellenname nach dem Wiederherstellen Ihrer DynamoDB-Tabelle aus einer Sicherung ändert, stellt der logische Tabellenname sicher, dass der Entschlüsselungsvorgang die Tabelle weiterhin erkennt.

Zulässige Attribute ohne Vorzeichen

Die `DO_NOTHING` in Ihren Attributaktionen markierten Attribute.

Die zulässigen Attribute ohne Vorzeichen teilen dem Client mit, welche Attribute von den Signaturen ausgeschlossen sind. Der Client geht davon aus, dass alle anderen Attribute in der Signatur enthalten sind. Beim Entschlüsseln eines Datensatzes bestimmt der Client dann aus den von Ihnen angegebenen zulässigen unsignierten Attributen, welche er überprüfen muss und welche ignoriert werden sollen. Sie können kein Attribut aus Ihren zulässigen Attributen ohne Vorzeichen entfernen.

Sie können die zulässigen Attribute ohne Vorzeichen explizit definieren, indem Sie ein Array erstellen, das alle Ihre `DO_NOTHING` Attribute auflistet. Sie können bei der Benennung Ihrer `DO_NOTHING` Attribute auch ein eindeutiges Präfix angeben und das Präfix verwenden, um dem Client mitzuteilen, welche Attribute vorzeichenlos sind. Wir empfehlen dringend, ein eindeutiges Präfix anzugeben, da dies das Hinzufügen eines neuen `DO_NOTHING` Attributs in der future vereinfacht. Weitere Informationen finden Sie unter Aktualisierung Ihres Datenmodells.

Wenn Sie kein Präfix für alle `DO_NOTHING` Attribute angeben, können Sie ein `allowedUnsignedAttributes` Array konfigurieren, das explizit alle Attribute auflistet, von denen der Client erwarten sollte, dass sie nicht signiert sind, wenn er sie bei der Entschlüsselung findet. Sie sollten Ihre erlaubten vorzeichenlosen Attribute nur dann explizit definieren, wenn dies unbedingt erforderlich ist.

Suchkonfiguration (optional)

Das `SearchConfig` definiert die Beacon-Version.

Der `SearchConfig` muss angegeben werden, um durchsuchbare Verschlüsselung oder signierte Beacons verwenden zu können.

Algorithm Suite (optional)

Die `algorithmSuiteId` definiert, welche Algorithmus-Suite das AWS Database Encryption SDK verwendet.

Sofern Sie nicht explizit eine alternative Algorithmussuite angeben, verwendet das AWS Database Encryption SDK die Standard-Algorithmussuite. Die Standard-Algorithmussuite verwendet den AES-GCM Algorithmus mit Schlüsselableitung, digitalen Signaturen und Schlüsselzusage. Obwohl die Standard-Algorithmus-Suite wahrscheinlich für die meisten Anwendungen geeignet ist, können Sie auch eine alternative Algorithmussuite wählen. Einige Vertrauensmodelle würden beispielsweise durch eine Algorithmus-Suite ohne digitale Signaturen erfüllt. Hinweise zu den Algorithmus-Suites, die das AWS Database Encryption SDK unterstützt, finden Sie unterUnterstützte Algorithmus-Suiten im AWS Database Encryption SDK.

Um die AES-GCM Algorithmus-Suite ohne digitale ECDSA-Signaturen auszuwählen, nehmen Sie den folgenden Ausschnitt in Ihre Tabellenverschlüsselungskonfiguration auf.

```
.algorithmSuiteId(
    DBEAlgorithmSuiteId.ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY_SYMSIG_HMAC_SHA384)
```

Aktualisierung von Elementen mit dem AWS SDK für Datenbankverschlüsselung

Das AWS Database Encryption SDK unterstützt ddb: nicht UpdateItem für Elemente, die verschlüsselt oder signiert wurden. Um ein verschlüsseltes oder signiertes Element zu aktualisieren, müssen Sie ddb: verwenden. PutItem Wenn Sie in Ihrer `PutItem` Anfrage denselben Primärschlüssel wie ein vorhandenes Element angeben, ersetzt das neue Element das vorhandene

Element vollständig. Sie können CLOBBER auch verwenden, um alle Attribute beim Speichern zu löschen und zu ersetzen, nachdem Sie Ihre Artikel aktualisiert haben.

Signierte Sets entschlüsseln

Wenn Sie in den Versionen 3.0.0 und 3.1.0 des AWS Database Encryption SDK ein Set-Typ-Attribut als definieren`SIGN_ONLY`, werden die Werte des Satzes in der Reihenfolge kanonisiert, in der sie bereitgestellt werden. DynamoDB behält die Reihenfolge der Sätze nicht bei. Daher besteht die Möglichkeit, dass die Signaturvalidierung des Elements, das den Satz enthält, fehlschlägt. Die Signaturvalidierung schlägt fehl, wenn die Werte des Satzes in einer anderen Reihenfolge zurückgegeben werden, als sie dem AWS Database Encryption SDK zur Verfügung gestellt wurden, auch wenn die Satzattribute dieselben Werte enthalten.

> **Note**
>
> Versionen 3.1.1 und höher des AWS Database Encryption SDK kanonisieren die Werte aller festgelegten Typattribute, sodass die Werte in derselben Reihenfolge gelesen werden, in der sie in DynamoDB geschrieben wurden.

Wenn die Signaturvalidierung fehlschlägt, schlägt der Entschlüsselungsvorgang fehl und es wird die folgende Fehlermeldung zurückgegeben.

```
software.amazon.cryptography.dbencryptionsdk.structuredencryption.model. StructuredEncrypti
onException: Es wurde kein Empfänger-Tag gefunden.
```

Wenn Sie die obige Fehlermeldung erhalten und glauben, dass das Objekt, das Sie zu entschlüsseln versuchen, einen Satz enthält, der mit Version 3.0.0 oder 3.1.0 signiert wurde, finden Sie im DecryptWithPermuteVerzeichnis des aws-database-encryption-sdk-dynamodb-java-Repositorys weitere Informationen zur erfolgreichen Validierung des Satzes. GitHub

Java-Beispiele

Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt . AWS Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Die folgenden Beispiele zeigen Ihnen, wie Sie die clientseitige Java-Verschlüsselungsbibliothek für DynamoDB verwenden, um die Tabellenelemente in Ihrer Anwendung zu schützen. Weitere Beispiele (und eigene Beispiele) finden Sie in den Java-Beispielen im -dynamodb-Repository unter aws-database-encryption-sdk. GitHub

Die folgenden Beispiele zeigen, wie die clientseitige Java-Verschlüsselungsbibliothek für DynamoDB in einer neuen, nicht aufgefüllten Amazon DynamoDB-Tabelle konfiguriert wird. Wenn Sie Ihre vorhandenen Amazon DynamoDB-Tabellen für die clientseitige Verschlüsselung konfigurieren möchten, finden Sie weitere Informationen unter. Fügen Sie Version 4.x zu einer vorhandenen Tabelle hinzu

Themen

- Verwenden des erweiterten DynamoDB-Clients
- Verwenden der Low-Level-DynamoDB-API
- Verwenden Sie die untergeordnete Ebene DynamoDbItemEncryptor

Verwenden des erweiterten DynamoDB-Clients

Das folgende Beispiel zeigt, wie Sie den DynamoDB Enhanced Client und `DynamoDbEncryptionInterceptor` mit einem AWS KMS Schlüsselbund verwenden, um DynamoDB-Tabellenelemente als Teil Ihrer DynamoDB-API-Aufrufe zu verschlüsseln.

Sie können jeden unterstützten Schlüsselbund mit dem DynamoDB Enhanced Client verwenden, wir empfehlen jedoch, wann immer möglich, einen der AWS KMS Schlüsselringe zu verwenden.

> **Note**
>
> Der DynamoDB Enhanced Client unterstützt keine durchsuchbare Verschlüsselung. Verwenden Sie die `DynamoDbEncryptionInterceptor` zusammen mit der Low-Level-DynamoDB-API, um eine durchsuchbare Verschlüsselung zu verwenden.

Sehen Sie sich das vollständige Codebeispiel an: .java EnhancedPutGetExample

Schritt 1: Erstellen Sie den Schlüsselbund AWS KMS

Das folgende Beispiel verwendet`CreateAwsKmsMrkMultiKeyring`, um einen AWS KMS Schlüsselbund mit einem symmetrischen Verschlüsselungs-KMS-Schlüssel zu erstellen. Die

`CreateAwsKmsMrkMultiKeyring` Methode stellt sicher, dass der Schlüsselbund sowohl Schlüssel mit einer Region als auch Schlüssel mit mehreren Regionen korrekt verarbeitet.

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

Schritt 2: Erstellen Sie ein Tabellenschema aus der annotierten Datenklasse

Im folgenden Beispiel wird die annotierte Datenklasse verwendet, um die zu erstellen. `TableSchema`

In diesem Beispiel wird davon ausgegangen, dass die mit Anmerkungen versehenen Datenklassen- und Attributaktionen mithilfe der SimpleClass Datei .java definiert wurden. Weitere Hinweise zum Kommentieren Ihrer Attributaktionen finden Sie unter. Verwenden Sie eine Datenklasse mit Anmerkungen

> **Note**
>
> Das AWS Database Encryption SDK unterstützt keine Anmerkungen zu verschachtelten Attributen.

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

Schritt 3: Definieren Sie, welche Attribute von den Signaturen ausgeschlossen werden

Im folgenden Beispiel wird davon ausgegangen, dass alle `DO_NOTHING` Attribute das eindeutige Präfix `":"` haben, und verwendet dieses Präfix, um die zulässigen Attribute ohne Vorzeichen zu definieren. Der Client geht davon aus, dass alle Attributnamen mit dem Präfix `":"` von den Signaturen ausgeschlossen sind. Weitere Informationen finden Sie unter Allowed unsigned attributes.

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

Schritt 4: Erstellen Sie die Verschlüsselungskonfiguration

Das folgende Beispiel definiert eine `tableConfigs` Map, die die Verschlüsselungskonfiguration für die DynamoDB-Tabelle darstellt.

In diesem Beispiel wird der DynamoDB-Tabellenname als logischer Tabellenname angegeben. Es wird dringend empfohlen, Ihren DynamoDB-Tabellennamen als logischen Tabellennamen anzugeben, wenn Sie Ihre Verschlüsselungskonfiguration zum ersten Mal definieren. Weitere Informationen finden Sie unter Verschlüsselungskonfiguration in der AWS Datenbankverschlüsselungs-SDK für DynamoDB.

> **Note**
>
> Um durchsuchbare Verschlüsselung oder signierte Beacons zu verwenden, müssen Sie die auch `SearchConfig`in Ihre Verschlüsselungskonfiguration aufnehmen.

```
final Map<String, DynamoDbEnhancedTableEncryptionConfig> tableConfigs = new
 HashMap<>();
tableConfigs.put(ddbTableName,
    DynamoDbEnhancedTableEncryptionConfig.builder()
        .logicalTableName(ddbTableName)
        .keyring(kmsKeyring)
        .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
        .schemaOnEncrypt(tableSchema)
        .build());
```

Schritt 5: Erstellt das **`DynamoDbEncryptionInterceptor`**

Im folgenden Beispiel wird `DynamoDbEncryptionInterceptor` mit dem `tableConfigs` aus Schritt 4 ein neues erstellt.

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

Schritt 6: Einen neuen AWS SDK-DynamoDB-Client erstellen

Im folgenden Beispiel wird ein neuer AWS SDK-DynamoDB-Client mit dem `interceptor` aus Schritt 5 erstellt.

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

Schritt 7: DynamoDB Enhanced Client erstellen und eine Tabelle erstellen

Im folgenden Beispiel wird der DynamoDB Enhanced Client mithilfe des AWS SDK-DynamoDB-Clients erstellt, der in Schritt 6 erstellt wurde, und es wird eine Tabelle mit der annotierten Datenklasse erstellt.

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

Schritt 8: Verschlüsseln und signieren Sie ein Tabellenelement

Im folgenden Beispiel wird mithilfe des DynamoDB Enhanced Client ein Element in die DynamoDB-Tabelle eingefügt. Das Element wird clientseitig verschlüsselt und signiert, bevor es an DynamoDB gesendet wird.

```
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);
```

Verwenden der Low-Level-DynamoDB-API

Das folgende Beispiel zeigt, wie Sie die Low-Level-DynamoDB-API mit einem AWS KMS Schlüsselbund verwenden, um Elemente automatisch clientseitig mit Ihren DynamoDB-Anfragen zu verschlüsseln und zu signieren. `PutItem`

Sie können jeden unterstützten Schlüsselbund verwenden, wir empfehlen jedoch, wann immer möglich, einen der Schlüsselbunde zu verwenden. AWS KMS

Sehen Sie sich das vollständige Codebeispiel an: .java BasicPutGetExample

Schritt 1: Erstellen Sie den Schlüsselbund AWS KMS

Das folgende Beispiel verwendet`CreateAwsKmsMrkMultiKeyring`, um einen AWS KMS Schlüsselbund mit einem symmetrischen Verschlüsselungs-KMS-Schlüssel zu erstellen. Die `CreateAwsKmsMrkMultiKeyring` Methode stellt sicher, dass der Schlüsselbund sowohl Schlüssel mit einer Region als auch Schlüssel mit mehreren Regionen korrekt verarbeitet.

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

Schritt 2: Konfigurieren Sie Ihre Attributaktionen

Das folgende Beispiel definiert eine `attributeActionsOnEncrypt` Map, die Beispiel-Attributaktionen für ein Tabellenelement darstellt.

> **Note**
>
> Das folgende Beispiel definiert keine Attribute als`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Wenn Sie `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Attribute angeben, müssen auch die Partitions- und Sortierattribute angegeben werden`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

```
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);
```

Schritt 3: Definieren Sie, welche Attribute von den Signaturen ausgeschlossen werden

Im folgenden Beispiel wird davon ausgegangen, dass alle `DO_NOTHING` Attribute das eindeutige Präfix `":"` haben, und verwendet dieses Präfix, um die zulässigen Attribute ohne Vorzeichen zu definieren. Der Client geht davon aus, dass alle Attributnamen mit dem Präfix `":"` von den Signaturen ausgeschlossen sind. Weitere Informationen finden Sie unter Allowed unsigned attributes.

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

Schritt 4: Definieren Sie die Konfiguration der DynamoDB-Tabellenverschlüsselung

Das folgende Beispiel definiert eine `tableConfigs` Map, die die Verschlüsselungskonfiguration für diese DynamoDB-Tabelle darstellt.

In diesem Beispiel wird der DynamoDB-Tabellenname als logischer Tabellenname angegeben. Es wird dringend empfohlen, Ihren DynamoDB-Tabellennamen als logischen Tabellennamen anzugeben, wenn Sie Ihre Verschlüsselungskonfiguration zum ersten Mal definieren. Weitere Informationen finden Sie unter Verschlüsselungskonfiguration in der AWS Datenbankverschlüsselungs-SDK für DynamoDB.

> **Note**
>
> Um durchsuchbare Verschlüsselung oder signierte Beacons zu verwenden, müssen Sie die auch `SearchConfig`in Ihre Verschlüsselungskonfiguration aufnehmen.

```
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);
```

Schritt 5: Erstellen Sie das **`DynamoDbEncryptionInterceptor`**

Im folgenden Beispiel wird das `DynamoDbEncryptionInterceptor` mit dem `tableConfigs` aus Schritt 4 erstellt.

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

Schritt 6: Einen neuen AWS SDK-DynamoDB-Client erstellen

Im folgenden Beispiel wird ein neuer AWS SDK-DynamoDB-Client mit dem `interceptor` aus Schritt 5 erstellt.

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

Schritt 7: Verschlüsseln und Signieren eines DynamoDB-Tabellenelements

Das folgende Beispiel definiert eine `item` Map, die ein Beispieltabellenelement darstellt, und platziert das Element in der DynamoDB-Tabelle. Das Element wird clientseitig verschlüsselt und signiert, bevor es an DynamoDB gesendet wird.

```
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);
```

Verwenden Sie die untergeordnete Ebene DynamoDbItemEncryptor

Das folgende Beispiel zeigt, wie Sie die untergeordnete Ebene `DynamoDbItemEncryptor` mit einem AWS KMS Schlüsselbund verwenden, um Tabellenelemente direkt zu verschlüsseln und zu signieren. Das `DynamoDbItemEncryptor` fügt das Element nicht in Ihre DynamoDB-Tabelle ein.

Sie können jeden unterstützten Schlüsselbund mit dem DynamoDB Enhanced Client verwenden, wir empfehlen jedoch, wann immer möglich, einen der AWS KMS Schlüsselringe zu verwenden.

> **Note**
>
> Die untergeordnete Ebene unterstützt keine durchsuchbare Verschlüsselung. `DynamoDbItemEncryptor` Verwenden Sie die `DynamoDbEncryptionInterceptor` zusammen mit der Low-Level-DynamoDB-API, um eine durchsuchbare Verschlüsselung zu verwenden.

Sehen Sie sich das vollständige Codebeispiel an: .java ItemEncryptDecryptExample

Schritt 1: Erstellen Sie den Schlüsselbund AWS KMS

Das folgende Beispiel verwendet`CreateAwsKmsMrkMultiKeyring`, um einen AWS KMS Schlüsselbund mit einem symmetrischen Verschlüsselungs-KMS-Schlüssel zu erstellen. Die `CreateAwsKmsMrkMultiKeyring` Methode stellt sicher, dass der Schlüsselbund sowohl Schlüssel mit einer Region als auch Schlüssel mit mehreren Regionen korrekt verarbeitet.

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

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

Schritt 2: Konfigurieren Sie Ihre Attributaktionen

Das folgende Beispiel definiert eine `attributeActionsOnEncrypt` Map, die Beispiel-Attributaktionen für ein Tabellenelement darstellt.

> **Note**
>
> Das folgende Beispiel definiert keine Attribute als`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Wenn Sie `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Attribute angeben, müssen auch die Partitions- und Sortierattribute angegeben werden`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

```
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);
```

Schritt 3: Definieren Sie, welche Attribute von den Signaturen ausgeschlossen werden

Im folgenden Beispiel wird davon ausgegangen, dass alle `DO_NOTHING` Attribute das eindeutige Präfix `":"` haben, und verwendet dieses Präfix, um die zulässigen Attribute ohne Vorzeichen zu definieren. Der Client geht davon aus, dass alle Attributnamen mit dem Präfix `":"` von den Signaturen ausgeschlossen sind. Weitere Informationen finden Sie unter Allowed unsigned attributes.

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

Schritt 4: Definieren Sie die **`DynamoDbItemEncryptor`** Konfiguration

Das folgende Beispiel definiert die Konfiguration für die`DynamoDbItemEncryptor`.

In diesem Beispiel wird der DynamoDB-Tabellenname als logischer Tabellenname angegeben. Es wird dringend empfohlen, Ihren DynamoDB-Tabellennamen als logischen Tabellennamen anzugeben, wenn Sie Ihre Verschlüsselungskonfiguration zum ersten Mal definieren. Weitere Informationen finden Sie unter Verschlüsselungskonfiguration in der AWS Datenbankverschlüsselungs-SDK für DynamoDB.

```
final DynamoDbItemEncryptorConfig config = DynamoDbItemEncryptorConfig.builder()
        .logicalTableName(ddbTableName)
        .partitionKeyName("partition_key")
        .sortKeyName("sort_key")
        .attributeActionsOnEncrypt(attributeActionsOnEncrypt)
        .keyring(kmsKeyring)
        .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
        .build();
```

Schritt 5: Erstellen Sie das **`DynamoDbItemEncryptor`**

Im folgenden Beispiel wird `DynamoDbItemEncryptor` mit dem `config` aus Schritt 4 ein neues erstellt.

```
final DynamoDbItemEncryptor itemEncryptor = DynamoDbItemEncryptor.builder()
        .DynamoDbItemEncryptorConfig(config)
        .build();
```

Schritt 6: Verschlüsseln und signieren Sie ein Tabellenelement direkt

Im folgenden Beispiel wird ein Element direkt verschlüsselt und signiert mit dem. `DynamoDbItemEncryptor` Das `DynamoDbItemEncryptor` fügt das Element nicht in Ihre DynamoDB-Tabelle ein.

```
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();
```

Konfigurieren Sie eine vorhandene DynamoDB-Tabelle für die Verwendung von AWS Datenbankverschlüsselungs-SDK für DynamoDB

Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt . AWS Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Mit Version 4. x der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB können Sie Ihre vorhandenen Amazon DynamoDB-Tabellen für die clientseitige Verschlüsselung konfigurieren. Dieses Thema enthält Anleitungen zu den drei Schritten, die Sie ausführen müssen, um Version 4 hinzuzufügen. x zu einer vorhandenen, gefüllten DynamoDB-Tabelle.

Voraussetzungen

Version 4. x der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB erfordert den DynamoDB Enhanced Client, der unter bereitgestellt wird. AWS SDK for Java 2.x Wenn Sie den DynamoDBMapper weiterhin verwenden, müssen Sie auf den DynamoDB AWS SDK for Java 2.x Enhanced Client migrieren.

Folgen Sie den Anweisungen für die Migration von Version 1.x auf 2.x von. AWS SDK für Java

Folgen Sie dann den Anweisungen unter Erste Schritte mit der DynamoDB Enhanced Client API.

Bevor Sie Ihre Tabelle für die Verwendung der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB konfigurieren, müssen Sie eine Datenklasse `TableSchema` mit Anmerkungen generieren und einen erweiterten Client erstellen.

Schritt 1: Bereiten Sie das Lesen und Schreiben verschlüsselter Elemente vor

Gehen Sie wie folgt vor, um Ihren AWS Database Encryption SDK-Client auf das Lesen und Schreiben verschlüsselter Elemente vorzubereiten. Nachdem Sie die folgenden Änderungen vorgenommen haben, liest und schreibt Ihr Client weiterhin Klartext-Elemente. Neue Elemente, die in die Tabelle geschrieben werden, werden nicht verschlüsselt oder signiert, aber es kann verschlüsselte Elemente entschlüsseln, sobald sie erscheinen. Diese Änderungen bereiten den

Client darauf vor, mit der Verschlüsselung neuer Elemente zu beginnen. Die folgenden Änderungen müssen auf jedem Lesegerät installiert werden, bevor Sie mit dem nächsten Schritt fortfahren.

1. Definieren Sie Ihre Attributaktionen

   Aktualisieren Sie Ihre Datenklasse mit Anmerkungen, sodass sie Attributaktionen enthält, die definieren, welche Attributwerte verschlüsselt und signiert werden, welche nur signiert und welche ignoriert werden.

   Weitere Hinweise zu den DynamoDB Enhanced Client-Anmerkungen finden Sie SimpleClass.javaim aws-database-encryption-sdk-dynamodb-Repository unter GitHub.

   Standardmäßig sind Primärschlüsselattribute signiert, aber nicht verschlüsselt () und alle anderen Attribute sind verschlüsselt und signiert (). `SIGN_ONLY` `ENCRYPT_AND_SIGN` Um Ausnahmen anzugeben, verwenden Sie die Verschlüsselungsanmerkungen, die in der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB definiert sind. Wenn Sie beispielsweise möchten, dass ein bestimmtes Attribut nur ein Zeichen sein soll, verwenden Sie die Anmerkung. `@DynamoDbEncryptionSignOnly` Wenn Sie möchten, dass ein bestimmtes Attribut signiert und in den Verschlüsselungskontext aufgenommen wird, verwenden Sie die `@DynamoDbEncryptionSignAndIncludeInEncryptionContext` Anmerkung. Wenn Sie möchten, dass ein bestimmtes Attribut weder signiert noch verschlüsselt (`DO_NOTHING`) wird, verwenden Sie die `@DynamoDbEncryptionDoNothing` Anmerkung.

   > **Note**
   >
   > Wenn Sie `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Attribute angeben, müssen dies auch die Partitions- und Sortierattribute sein`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Ein Beispiel, das die zur Definition verwendeten Anmerkungen zeigt`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, finden Sie unter SimpleClass4.java.

   Anmerkungen finden Sie beispielsweise unter. Verwenden Sie eine Datenklasse mit Anmerkungen

2. Definieren Sie, welche Attribute von den Signaturen ausgeschlossen werden

   Im folgenden Beispiel wird davon ausgegangen, dass alle `DO_NOTHING` Attribute das eindeutige Präfix `":"` haben, und verwendet dieses Präfix, um die zulässigen Attribute ohne Vorzeichen

zu definieren. Der Client geht davon aus, dass alle Attributnamen mit dem Präfix `":"` von den Signaturen ausgeschlossen sind. Weitere Informationen finden Sie unter Allowed unsigned attributes.

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

3. Erstellen Sie einen Schlüsselbund

   Im folgenden Beispiel wird ein AWS KMS Schlüsselbund erstellt. Der AWS KMS Schlüsselbund verwendet symmetrische Verschlüsselung oder asymmetrisches RSA, um Datenschlüssel AWS KMS keys zu generieren, zu verschlüsseln und zu entschlüsseln.

   In diesem Beispiel wird ein AWS KMS Schlüsselbund `CreateMrkMultiKeyring` mit einem KMS-Schlüssel mit symmetrischer Verschlüsselung erstellt. Die `CreateAwsKmsMrkMultiKeyring` Methode stellt sicher, dass der Schlüsselbund sowohl Schlüssel mit einer Region als auch Schlüssel mit mehreren Regionen korrekt verarbeitet.

```
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. Definieren Sie die Konfiguration der DynamoDB-Tabellenverschlüsselung

   Das folgende Beispiel definiert eine `tableConfigs` Map, die die Verschlüsselungskonfiguration für diese DynamoDB-Tabelle darstellt.

   In diesem Beispiel wird der DynamoDB-Tabellenname als logischer Tabellenname angegeben. Es wird dringend empfohlen, Ihren DynamoDB-Tabellennamen als logischen Tabellennamen anzugeben, wenn Sie Ihre Verschlüsselungskonfiguration zum ersten Mal definieren. Weitere Informationen finden Sie unter Verschlüsselungskonfiguration in der AWS Datenbankverschlüsselungs-SDK für DynamoDB.

   Sie müssen `FORCE_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT` als Klartext-Override angeben. Diese Richtlinie liest und schreibt weiterhin Klartext-Elemente, liest verschlüsselte Elemente und bereitet den Client darauf vor, verschlüsselte Elemente zu schreiben.

```
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)

 .plaintextOverride(PlaintextOverride.FORCE_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT)
        .build();
tableConfigs.put(ddbTableName, config);
```

5. Erstellen der **`DynamoDbEncryptionInterceptor`**

   Im folgenden Beispiel wird das `DynamoDbEncryptionInterceptor` `tableConfigs` aus Schritt 3 erstellt.

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

Schritt 2: Schreiben Sie verschlüsselte und signierte Elemente

Aktualisieren Sie die Klartext-Richtlinie in Ihrer `DynamoDbEncryptionInterceptor` Konfiguration, damit der Client verschlüsselte und signierte Elemente schreiben kann. Nachdem Sie die folgende Änderung implementiert haben, verschlüsselt und signiert der Client neue Elemente auf der Grundlage der Attributaktionen, die Sie in Schritt 1 konfiguriert haben. Der Client kann Klartext-Elemente sowie verschlüsselte und signierte Elemente lesen.

Bevor Sie mit Schritt 3 fortfahren, müssen Sie alle vorhandenen Klartextelemente in Ihrer Tabelle verschlüsseln und signieren. Es gibt keine einzelne Metrik oder Abfrage, die Sie ausführen können, um Ihre vorhandenen Klartextelemente schnell zu verschlüsseln. Verwenden Sie den Prozess, der für Ihr System am sinnvollsten ist. Sie könnten beispielsweise einen asynchronen Prozess verwenden, der die Tabelle langsam scannt und dann die Elemente mithilfe der von Ihnen definierten Attributaktionen und der Verschlüsselungskonfiguration neu schreibt. Um die Klartext-Elemente in Ihrer Tabelle zu identifizieren, empfehlen wir, nach allen Elementen zu suchen, die nicht die

`aws_dbe_foot` Attribute `aws_dbe_head` und enthalten, die das AWS Database Encryption SDK Elementen hinzufügt, wenn sie verschlüsselt und signiert sind.

Im folgenden Beispiel wird die Konfiguration der Tabellenverschlüsselung aus Schritt 1 aktualisiert. Sie müssen die Klartext-Override mit `FORBID_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT` aktualisieren. Diese Richtlinie liest weiterhin Klartext-Elemente, liest und schreibt aber auch verschlüsselte Elemente. Erstellen Sie ein neues `DynamoDbEncryptionInterceptor` mit dem aktualisierten`tableConfigs`.

```
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)

 .plaintextOverride(PlaintextOverride.FORBID_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT)
        .build();
tableConfigs.put(ddbTableName, config);
```

Schritt 3: Nur verschlüsselte und signierte Elemente lesen

Nachdem Sie alle Ihre Elemente verschlüsselt und signiert haben, aktualisieren Sie die Klartext-Überschreibung in Ihrer `DynamoDbEncryptionInterceptor` Konfiguration, sodass der Client nur verschlüsselte und signierte Elemente lesen und schreiben kann. Nachdem Sie die folgende Änderung implementiert haben, verschlüsselt und signiert der Client neue Elemente auf der Grundlage der Attributaktionen, die Sie in Schritt 1 konfiguriert haben. Der Client kann nur verschlüsselte und signierte Elemente lesen.

Im folgenden Beispiel wird die Konfiguration der Tabellenverschlüsselung aus Schritt 2 aktualisiert. Sie können entweder die Klartext-Override mit der Klartext-Richtlinie aktualisieren `FORBID_WRITE_PLAINTEXT_FORBID_READ_PLAINTEXT` oder die Klartext-Richtlinie aus Ihrer Konfiguration entfernen. Der Client liest und schreibt standardmäßig nur verschlüsselte und signierte Elemente. Erstellen Sie ein neues `DynamoDbEncryptionInterceptor` mit dem aktualisierten`tableConfigs`.

```
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)
        // Optional: you can also remove the plaintext policy from your configuration

 .plaintextOverride(PlaintextOverride.FORBID_WRITE_PLAINTEXT_FORBID_READ_PLAINTEXT)
        .build();
tableConfigs.put(ddbTableName, config);
```

Migrieren Sie auf Version 4.x der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB

Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Version 3. x der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB ist eine grundlegende Neufassung der 2. x-Codebasis. Sie enthält zahlreiche Updates, wie z. B. ein neues strukturiertes Datenformat, verbesserte Mehrmandantenunterstützung, nahtlose Schemaänderungen und Unterstützung für durchsuchbare Verschlüsselung. Version 4. x des AWS Database Encryption SDK for Java ersetzt das eingebettete 2. X-Bibliothek mit einer Implementierung, die auf dem AWS SDK for Java 2 basiert. x, wodurch die Abhängigkeit vom AWS SDK for Java 1 entfernt wird. x. Dieses Thema enthält Anleitungen zur Migration Ihres Codes auf Version 4. x.

Migration von Version 1.x auf 2.x

Migrieren Sie auf Version 2. x bevor Sie auf Version 3 migrieren. x oder 4. x. Ausführung 2. x hat das Symbol für den neuesten Anbieter von `MostRecentProvider` zu geändert`CachingMostRecentProvider`. Wenn Sie derzeit Version 1 verwenden. x der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB mit dem `MostRecentProvider` Symbol, auf das Sie den Symbolnamen in Ihrem Code aktualisieren müssen. `CachingMostRecentProvider` Weitere Informationen finden Sie unter Updates für den neuesten Anbieter.

Migration von Version 2.x auf 3.x

Die folgenden Verfahren beschreiben, wie Sie Ihren Code von Version 2 migrieren. x auf Version 3. x der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB.

Schritt 1. Bereiten Sie sich darauf vor, Elemente im neuen Format zu lesen

Gehen Sie wie folgt vor, um Ihren AWS Database Encryption SDK-Client darauf vorzubereiten, Elemente im neuen Format zu lesen. Nachdem Sie die folgenden Änderungen implementiert haben, wird sich Ihr Client weiterhin genauso verhalten wie in Version 2. x. Ihr Kunde wird weiterhin Elemente in der Version 2 lesen und schreiben. X-Format, aber diese Änderungen bereiten den Client darauf vor, Elemente im neuen Format zu lesen.

Aktualisieren Sie Ihre Version AWS SDK für Java auf Version 2.x

Version 3. x der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB erfordert den DynamoDB Enhanced Client. Der DynamoDB Enhanced Client ersetzt den DynamoDBMapper, der in früheren Versionen verwendet wurde. Um den erweiterten Client zu verwenden, müssen Sie den verwenden. AWS SDK for Java 2.x

Folgen Sie den Anweisungen für die Migration von Version 1.x auf 2.x von. AWS SDK für Java

Weitere Informationen darüber, welche AWS SDK for Java 2.x Module erforderlich sind, finden Sie unter. Voraussetzungen

Konfigurieren Sie Ihren Client so, dass er Elemente liest, die mit älteren Versionen verschlüsselt wurden

Die folgenden Verfahren bieten einen Überblick über die Schritte, die im folgenden Codebeispiel demonstriert werden.

1. Erstellen Sie einen Schlüsselbund.

   Keyrings und Cryptographic Materials Manager ersetzen die Anbieter für kryptografisches Material, die in früheren Versionen der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB verwendet wurden.

> **Important**
>
> Die Umschließungsschlüssel, die Sie bei der Erstellung eines Schlüsselbunds angeben, müssen dieselben Wickelschlüssel sein, die Sie in Version 2 mit Ihrem Anbieter für kryptografisches Material verwendet haben. x.

2. Erstellen Sie ein Tabellenschema über Ihrer annotierten Klasse.

   In diesem Schritt werden die Attributaktionen definiert, die verwendet werden, wenn Sie mit dem Schreiben von Elementen im neuen Format beginnen.

   Anleitungen zur Verwendung des neuen DynamoDB Enhanced Client finden Sie unter Generate a `TableSchema` im AWS SDK für Java Developer Guide.

   Im folgenden Beispiel wird davon ausgegangen, dass Sie Ihre annotierte Klasse aus Version 2 aktualisiert haben. x verwendet die Anmerkungen zu den neuen Attributaktionen. Weitere Hinweise zum Kommentieren Ihrer Attributaktionen finden Sie unter. Verwenden Sie eine Datenklasse mit Anmerkungen

   > **Note**
   >
   > Wenn Sie `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Attribute angeben, müssen auch die Partitions- und Sortierattribute angegeben werden`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Ein Beispiel, das die zur Definition verwendeten Anmerkungen zeigt`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, finden Sie unter SimpleClass4.java.

3. Definieren Sie, welche Attribute von der Signatur ausgeschlossen werden.
4. Konfigurieren Sie eine explizite Zuordnung der Attributaktionen, die in Ihrer modellierten Klasse von Version 2.x konfiguriert sind.

   In diesem Schritt werden die Attributaktionen definiert, die zum Schreiben von Elementen im alten Format verwendet wurden.

5. Konfigurieren `DynamoDBEncryptor` Sie die, die Sie in Version 2 verwendet haben. x der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB.
6. Konfigurieren Sie das ältere Verhalten.

7. Erstellen Sie einen `DynamoDbEncryptionInterceptor`.
8. Erstellen Sie einen neuen AWS SDK-DynamoDB-Client.
9. Erstellen Sie die `DynamoDBEnhancedClient` und erstellen Sie eine Tabelle mit Ihrer modellierten Klasse.

   Weitere Informationen zum DynamoDB Enhanced Client finden Sie unter Erstellen eines erweiterten Clients.

```
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 GitHub repository.
        //    All primary key attributes must be signed but not encrypted
        //    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);
    }
}
```

Schritt 2. Schreiben Sie Elemente im neuen Format

Nachdem Sie die Änderungen aus Schritt 1 für alle Leser bereitgestellt haben, führen Sie die folgenden Schritte aus, um Ihren AWS Database Encryption SDK-Client so zu konfigurieren, dass er Elemente im neuen Format schreibt. Nachdem Sie die folgenden Änderungen implementiert haben, liest Ihr Client weiterhin Elemente im alten Format und beginnt, Elemente im neuen Format zu schreiben und zu lesen.

Die folgenden Verfahren bieten einen Überblick über die Schritte, die im folgenden Codebeispiel demonstriert werden.

1. Fahren Sie mit der Konfiguration Ihres Schlüsselbundes, des Tabellenschemas, der veralteten Attributaktionen und `DynamoDBEncryptor` so fort`allowedUnsignedAttributes`, wie Sie es in Schritt 1 getan haben.
2. Aktualisieren Sie Ihr bisheriges Verhalten, sodass nur neue Elemente mit dem neuen Format geschrieben werden.
3. Erstellen eines `DynamoDbEncryptionInterceptor`
4. Erstellen Sie einen neuen AWS SDK-DynamoDB-Client.
5. Erstellen Sie die `DynamoDBEnhancedClient` und erstellen Sie eine Tabelle mit Ihrer modellierten Klasse.

   Weitere Informationen zum DynamoDB Enhanced Client finden Sie unter Erstellen eines erweiterten Clients.

```
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);
    }
}
```

Nach der Bereitstellung der Änderungen in Schritt 2 müssen Sie alle alten Elemente in Ihrer Tabelle erneut mit dem neuen Format verschlüsseln, bevor Sie mit Schritt 3 fortfahren können. Es gibt keine einzelne Metrik oder Abfrage, die Sie ausführen können, um Ihre vorhandenen Elemente schnell zu verschlüsseln. Verwenden Sie den Prozess, der für Ihr System am sinnvollsten ist. Sie könnten beispielsweise einen asynchronen Prozess verwenden, der die Tabelle langsam scannt und dann die Elemente mithilfe der neuen Attributaktionen und der Verschlüsselungskonfiguration, die Sie definiert haben, neu schreibt.

Schritt 3. Nur Elemente im neuen Format lesen und schreiben

Nachdem Sie alle Elemente in Ihrer Tabelle mit dem neuen Format erneut verschlüsselt haben, können Sie das alte Verhalten aus Ihrer Konfiguration entfernen. Gehen Sie wie folgt vor, um Ihren Client so zu konfigurieren, dass er nur Elemente im neuen Format liest und schreibt.

Die folgenden Verfahren bieten einen Überblick über die Schritte, die im folgenden Codebeispiel demonstriert werden.

1. Fahren Sie mit der Konfiguration Ihres Schlüsselbundes und des `allowedUnsignedAttributes` Tabellenschemas wie in Schritt 1 fort. Entfernen Sie die veralteten Attributaktionen und `DynamoDBEncryptor` aus Ihrer Konfiguration.
2. Erstellen Sie einen `DynamoDbEncryptionInterceptor`.
3. Erstellen Sie einen neuen AWS SDK-DynamoDB-Client.
4. Erstellen Sie die `DynamoDBEnhancedClient` und erstellen Sie eine Tabelle mit Ihrer modellierten Klasse.

   Weitere Informationen zum DynamoDB Enhanced Client finden Sie unter Erstellen eines erweiterten Clients.

```
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);
    }
}
```

Migration von Version 3.x auf 4.x

Version 4. x ersetzt das AWS SDK for Java 1.x durch das AWS SDK for Java 2.x in der eingebetteten Version 2. x-Code. Die Version 3. x APIs sind unverändert.

Wenn Ihr Code nur Version 3 verwendet. x APIs und verwendet nicht die eingebettete Version 2. x-Code, es sind keine Änderungen erforderlich.

Wenn Ihr Code die eingebettete Version 2 verwendet. x-Code, Sie müssen Ihren Code aktualisieren, um das AWS SDK for Java 2 verwenden zu können. X-Symbole. Beispiele für die aktualisierten Symbole finden Sie unter Java-Beispiele. Die eingebettete Version 2. x-Code ist nicht mehr enthaltenDynamoDBMapper, da das AWS SDK for Java 2 nicht mehr enthalten ist. x unterstützt nichtDynamoDBMapper.

.NET

In diesem Thema wird erklärt, wie Version 3 installiert und verwendet wird. x der clientseitigen .NET-Verschlüsselungsbibliothek für DynamoDB. Einzelheiten zur Programmierung mit dem AWS Database Encryption SDK für DynamoDB finden Sie in den.NET-Beispielen im aws-database-encryption-sdk -dynamodb-Repository unter. GitHub

Die clientseitige .NET-Verschlüsselungsbibliothek für DynamoDB richtet sich an Entwickler, die Anwendungen in C# und anderen .NET-Programmiersprachen schreiben. Sie wird unter Windows, macOS und Linux unterstützt.

Alle Programmiersprachenimplementierungen des AWS Database Encryption SDK für DynamoDB sind interoperabel. Leere Werte für Listen- oder Zuordnungsdatentypen werden jedoch SDK für .NET nicht unterstützt. Das heißt, wenn Sie die clientseitige Java-Verschlüsselungsbibliothek für DynamoDB verwenden, um ein Element zu schreiben, das leere Werte für einen Listen- oder Zuordnungsdatentyp enthält, können Sie dieses Element nicht mit der clientseitigen .NET-Verschlüsselungsbibliothek für DynamoDB entschlüsseln und lesen.

Themen

Installation der clientseitigen .NET-Verschlüsselungsbibliothek für DynamoDB

Die.NET-clientseitige Verschlüsselungsbibliothek für DynamoDB ist als AWS.Cryptography verfügbar. DbEncryptionSDK. DynamoDbverpacken in NuGet. Einzelheiten zur Installation und Erstellung der Bibliothek finden Sie in der.NET-README.md-Datei im -dynamodb-Repository. aws-database-encryption-sdk Die.NET-clientseitige Verschlüsselungsbibliothek für DynamoDB erfordert die Schlüssel, SDK für .NET auch wenn Sie keine () verwenden AWS Key Management Service .AWS KMS Die SDK für .NET wird mit dem Paket installiert. NuGet

Version 3. x der clientseitigen .NET-Verschlüsselungsbibliothek für DynamoDB unterstützt .NET 6.0 und .NET Framework net48 und höher.

Debuggen mit.NET

Die.NET-clientseitige Verschlüsselungsbibliothek für DynamoDB generiert keine Protokolle. Ausnahmen in der clientseitigen .NET-Verschlüsselungsbibliothek für DynamoDB generieren eine Ausnahmemeldung, aber keine Stack-Traces.

Um Ihnen beim Debuggen zu helfen, stellen Sie sicher, dass Sie die Anmeldung bei aktivieren. SDK für .NET Mithilfe der Protokolle und Fehlermeldungen von SDK für .NET können Sie Fehler, die in der auftreten, SDK für .NET von denen in der clientseitigen .NET-Verschlüsselungsbibliothek für DynamoDB unterscheiden. Hilfe zur SDK für .NET Protokollierung finden Sie AWSLoggingim

Entwicklerhandbuch.AWS SDK für .NET (Um das Thema zu lesen, erweitern Sie den Abschnitt Öffnen, um .NET Framework-Inhalte anzuzeigen.)

Verwenden der clientseitigen .NET-Verschlüsselungsbibliothek für DynamoDB

In diesem Thema werden einige der Funktionen und Hilfsklassen in Version 3 erklärt. x der clientseitigen .NET-Verschlüsselungsbibliothek für DynamoDB.

Einzelheiten zur Programmierung mit der clientseitigen .NET-Verschlüsselungsbibliothek für DynamoDB finden Sie in den.NET-Beispielen im -dynamodb-Repository unter. aws-database-encryption-sdk GitHub

Themen

- Elementverschlüssler
- Attributaktionen im AWS Database Encryption SDK für DynamoDB
- Verschlüsselungskonfiguration im AWS Database Encryption SDK für DynamoDB
- Elemente mit dem Database Encryption SDK aktualisieren AWS

Elementverschlüssler

Im Kern ist das AWS Database Encryption SDK für DynamoDB ein Elementverschlüsseler. Sie können Version 3 verwenden. x der clientseitigen .NET-Verschlüsselungsbibliothek für DynamoDB, um Ihre DynamoDB-Tabellenelemente auf folgende Weise zu verschlüsseln, zu signieren, zu verifizieren und zu entschlüsseln.

Das AWS Low-Level-Datenbankverschlüsselungs-SDK für die DynamoDB-API

> Sie können Ihre Tabellenverschlüsselungskonfiguration verwenden, um einen DynamoDB-Client zu erstellen, der Elemente automatisch clientseitig mit Ihren DynamoDB-Anfragen verschlüsselt und signiert. `PutItem` Sie können diesen Client direkt verwenden, oder Sie können ein Dokumentmodell oder ein Objektpersistenzmodell erstellen.
>
> Sie müssen das AWS Low-Level-Datenbankverschlüsselungs-SDK für DynamoDB-API verwenden, um durchsuchbare Verschlüsselung verwenden zu können.

Die untergeordnete Ebene **`DynamoDbItemEncryptor`**

> Die untergeordnete Ebene verschlüsselt und signiert oder entschlüsselt und verifiziert Ihre Tabellenelemente `DynamoDbItemEncryptor` direkt, ohne DynamoDB aufzurufen. Es stellt keine

DynamoDB `PutItem` oder `GetItem` Anfragen. Sie können beispielsweise die untergeordnete Ebene verwenden, `DynamoDbItemEncryptor` um ein DynamoDB-Element, das Sie bereits abgerufen haben, direkt zu entschlüsseln und zu verifizieren. Wenn Sie die untergeordnete Ebene verwenden`DynamoDbItemEncryptor`, empfehlen wir die Verwendung des Low-Level-Programmiermodells, das für die Kommunikation mit SDK für .NET DynamoDB vorgesehen ist.

Die untergeordnete Ebene unterstützt keine durchsuchbare Verschlüsselung`DynamoDbItemEncryptor`.

Attributaktionen im AWS Database Encryption SDK für DynamoDB

Attributaktionen bestimmen, welche Attributwerte verschlüsselt und signiert werden, welche nur signiert sind, welche signiert und in den Verschlüsselungskontext aufgenommen werden und welche ignoriert werden.

Um Attributaktionen mit dem .NET-Client anzugeben, definieren Sie Attributaktionen manuell mithilfe eines Objektmodells. Geben Sie Ihre Attributaktionen an, indem Sie ein `Dictionary` Objekt erstellen, in dem die Name-Wert-Paare für Attributnamen und die angegebenen Aktionen stehen.

Geben Sie `ENCRYPT_AND_SIGN` an, dass ein Attribut verschlüsselt und signiert werden soll. Geben Sie `SIGN_ONLY` an, dass ein Attribut signiert, aber nicht verschlüsselt werden soll. Geben Sie `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` an, dass ein Attribut signiert und in den Verschlüsselungskontext aufgenommen werden soll. Sie können ein Attribut nicht verschlüsseln, ohne es auch zu signieren. Geben Sie `DO_NOTHING` an, ob ein Attribut ignoriert werden soll.

Die Partitions- und Sortierattribute müssen entweder `SIGN_ONLY` oder lauten`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Wenn Sie Attribute als definieren`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, müssen dies auch die Partitions- und Sortierattribute sein`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

> **Note**
>
> Nachdem Sie Ihre Attributaktionen definiert haben, müssen Sie definieren, welche Attribute von den Signaturen ausgeschlossen werden. Um das future Hinzufügen neuer Attribute ohne Vorzeichen zu vereinfachen, empfehlen wir, ein eindeutiges Präfix (wie ":„) zu wählen, um Ihre vorzeichenlosen Attribute zu identifizieren. Nehmen Sie dieses Präfix in den Attributnamen für alle Attribute auf, die Sie bei der Definition Ihres DynamoDB-Schemas und Ihrer Attributaktionen markiert `DO_NOTHING` haben.

Das folgende Objektmodell veranschaulicht, wie Sie`ENCRYPT_AND_SIGN`, `SIGN_ONLY`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT, und `DO_NOTHING` Attributaktionen mit dem .NET-Client angeben. In diesem Beispiel wird das Präfix `":"` verwendet, um `DO_NOTHING` Attribute zu identifizieren.

> **Note**
>
> Um die `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` kryptografische Aktion verwenden zu können, müssen Sie Version 3.3 oder höher des AWS Database Encryption SDK verwenden. Stellen Sie die neue Version für alle Leser bereit, bevor Sie Ihr Datenmodell so aktualisieren, dass es diese enthält`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

```
var attributeActionsOnEncrypt = new Dictionary<string, CryptoAction>
{
    ["partition_key"] = CryptoAction.SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT, // The
 partition attribute must be signed
    ["sort_key"] = CryptoAction.SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT, // The sort
 attribute must be signed
    ["attribute1"] = CryptoAction.ENCRYPT_AND_SIGN,
    ["attribute2"] = CryptoAction.SIGN_ONLY,
    ["attribute3"] = CryptoAction.SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT,
    [":attribute4"] = CryptoAction.DO_NOTHING
};
```

Verschlüsselungskonfiguration im AWS Database Encryption SDK für DynamoDB

Wenn Sie das AWS Database Encryption SDK verwenden, müssen Sie explizit eine Verschlüsselungskonfiguration für Ihre DynamoDB-Tabelle definieren. Die in Ihrer Verschlüsselungskonfiguration erforderlichen Werte hängen davon ab, ob Sie Ihre Attributaktionen manuell oder mit einer annotierten Datenklasse definiert haben.

Der folgende Ausschnitt definiert eine DynamoDB-Tabellenverschlüsselungskonfiguration unter Verwendung des AWS Low-Level-Datenbankverschlüsselungs-SDK für DynamoDB-API und zulässige unsignierte Attribute, die durch ein eindeutiges Präfix definiert sind.

```
Dictionary<String, DynamoDbTableEncryptionConfig> tableConfigs =
    new Dictionary<String, DynamoDbTableEncryptionConfig>();
DynamoDbTableEncryptionConfig config = new DynamoDbTableEncryptionConfig
{
```

```
    LogicalTableName = ddbTableName,
    PartitionKeyName = "partition_key",
    SortKeyName = "sort_key",
    AttributeActionsOnEncrypt = attributeActionsOnEncrypt,
    Keyring = kmsKeyring,
    AllowedUnsignedAttributePrefix = unsignAttrPrefix,
    // Optional: SearchConfig only required if you use beacons
    Search = new SearchConfig
    {
        WriteVersion = 1, // MUST be 1
        Versions = beaconVersions
    }
};
tableConfigs.Add(ddbTableName, config);
```

Logischer Tabellenname

Ein logischer Tabellenname für Ihre DynamoDB-Tabelle.

Der logische Tabellenname ist kryptografisch an alle in der Tabelle gespeicherten Daten gebunden, um DynamoDB-Wiederherstellungsvorgänge zu vereinfachen. Es wird dringend empfohlen, Ihren DynamoDB-Tabellennamen als logischen Tabellennamen anzugeben, wenn Sie Ihre Verschlüsselungskonfiguration zum ersten Mal definieren. Sie müssen immer denselben logischen Tabellennamen angeben. Damit die Entschlüsselung erfolgreich ist, muss der Name der logischen Tabelle mit dem Namen übereinstimmen, der bei der Verschlüsselung angegeben wurde. Falls sich Ihr DynamoDB-Tabellenname nach dem Wiederherstellen Ihrer DynamoDB-Tabelle aus einer Sicherung ändert, stellt der logische Tabellenname sicher, dass der Entschlüsselungsvorgang die Tabelle weiterhin erkennt.

Zulässige Attribute ohne Vorzeichen

Die `DO_NOTHING` in Ihren Attributaktionen markierten Attribute.

Die zulässigen Attribute ohne Vorzeichen teilen dem Client mit, welche Attribute von den Signaturen ausgeschlossen sind. Der Client geht davon aus, dass alle anderen Attribute in der Signatur enthalten sind. Beim Entschlüsseln eines Datensatzes bestimmt der Client dann anhand der von Ihnen angegebenen zulässigen Attribute ohne Vorzeichen, welche er überprüfen muss und welche ignoriert werden sollen. Sie können kein Attribut aus Ihren zulässigen Attributen ohne Vorzeichen entfernen.

Sie können die zulässigen Attribute ohne Vorzeichen explizit definieren, indem Sie ein Array erstellen, das alle Ihre `DO_NOTHING` Attribute auflistet. Sie können bei der Benennung Ihrer

`DO_NOTHING` Attribute auch ein eindeutiges Präfix angeben und das Präfix verwenden, um dem Client mitzuteilen, welche Attribute vorzeichenlos sind. Wir empfehlen dringend, ein eindeutiges Präfix anzugeben, da dies das Hinzufügen eines neuen `DO_NOTHING` Attributs in der future vereinfacht. Weitere Informationen finden Sie unter Aktualisierung Ihres Datenmodells.

Wenn Sie kein Präfix für alle `DO_NOTHING` Attribute angeben, können Sie ein `allowedUnsignedAttributes` Array konfigurieren, das explizit alle Attribute auflistet, von denen der Client erwarten sollte, dass sie nicht signiert sind, wenn er sie bei der Entschlüsselung findet. Sie sollten Ihre erlaubten vorzeichenlosen Attribute nur dann explizit definieren, wenn dies unbedingt erforderlich ist.

Suchkonfiguration (optional)

Das `SearchConfig` definiert die Beacon-Version.

Der `SearchConfig` muss angegeben werden, um durchsuchbare Verschlüsselung oder signierte Beacons verwenden zu können.

Algorithm Suite (optional)

Die `algorithmSuiteId` definiert, welche Algorithmus-Suite das AWS Database Encryption SDK verwendet.

Sofern Sie nicht explizit eine alternative Algorithmussuite angeben, verwendet das AWS Database Encryption SDK die Standard-Algorithmussuite. Die Standard-Algorithmussuite verwendet den AES-GCM-Algorithmus mit Schlüsselableitung, digitalen Signaturen und Schlüsselzusage. Obwohl die Standard-Algorithmus-Suite wahrscheinlich für die meisten Anwendungen geeignet ist, können Sie auch eine alternative Algorithmussuite wählen. Einige Vertrauensmodelle würden beispielsweise durch eine Algorithmus-Suite ohne digitale Signaturen erfüllt. Hinweise zu den Algorithmus-Suites, die das AWS Database Encryption SDK unterstützt, finden Sie unterUnterstützte Algorithmus-Suiten im AWS Database Encryption SDK.

Um die AES-GCM-Algorithmussuite ohne digitale ECDSA-Signaturen auszuwählen, nehmen Sie den folgenden Ausschnitt in Ihre Tabellenverschlüsselungskonfiguration auf.

```
AlgorithmSuiteId =
 DBEAlgorithmSuiteId.ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY_SYMSIG_HMAC_SHA384
```

Elemente mit dem Database Encryption SDK aktualisieren AWS

Das AWS Database Encryption SDK unterstützt ddb: nicht UpdateItem für Elemente, die verschlüsselte oder signierte Attribute enthalten. Um ein verschlüsseltes oder signiertes Attribut zu aktualisieren, müssen Sie ddb: verwenden. PutItem Wenn Sie in Ihrer `PutItem` Anfrage denselben Primärschlüssel wie ein vorhandenes Element angeben, ersetzt das neue Element das vorhandene Element vollständig. Sie können CLOBBER auch verwenden, um alle Attribute beim Speichern zu löschen und zu ersetzen, nachdem Sie Ihre Artikel aktualisiert haben.

.NET-Beispiele

Die folgenden Beispiele zeigen, wie Sie die clientseitige .NET-Verschlüsselungsbibliothek für DynamoDB verwenden, um die Tabellenelemente in Ihrer Anwendung zu schützen. Weitere Beispiele (und eigene Beispiele) finden Sie in den .NET-Beispielen im -dynamodb-Repository unter. aws-database-encryption-sdk GitHub

Die folgenden Beispiele zeigen, wie die clientseitige .NET-Verschlüsselungsbibliothek für DynamoDB in einer neuen, nicht aufgefüllten Amazon DynamoDB-Tabelle konfiguriert wird. Wenn Sie Ihre vorhandenen Amazon DynamoDB-Tabellen für die clientseitige Verschlüsselung konfigurieren möchten, finden Sie weitere Informationen unter. Fügen Sie Version 3.x zu einer vorhandenen Tabelle hinzu

Themen

- Verwenden des AWS Low-Level-Datenbankverschlüsselungs-SDK für DynamoDB-API
- Verwenden Sie die untergeordnete Ebene DynamoDbItemEncryptor

Verwenden des AWS Low-Level-Datenbankverschlüsselungs-SDK für DynamoDB-API

Das folgende Beispiel zeigt, wie Sie das AWS Low-Level-Datenbankverschlüsselungs-SDK für die DynamoDB-API mit einem AWS KMS Schlüsselbund verwenden, um Elemente automatisch clientseitig mit Ihren DynamoDB-Anfragen zu verschlüsseln und zu signieren. `PutItem`

Sie können jeden unterstützten Schlüsselbund verwenden, wir empfehlen jedoch, wann immer möglich, einen der Schlüsselringe zu verwenden. AWS KMS

Sehen Sie sich das vollständige Codebeispiel an: .cs BasicPutGetExample

Schritt 1: Erstellen Sie den Schlüsselbund AWS KMS

Das folgende Beispiel verwendet`CreateAwsKmsMrkMultiKeyring`, um einen AWS KMS Schlüsselbund mit einem symmetrischen Verschlüsselungs-KMS-Schlüssel zu erstellen. Die `CreateAwsKmsMrkMultiKeyring` Methode stellt sicher, dass der Schlüsselbund sowohl Schlüssel mit einer Region als auch Schlüssel mit mehreren Regionen korrekt verarbeitet.

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var keyringInput = new CreateAwsKmsMrkMultiKeyringInput { Generator = kmsKeyId };
var kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

Schritt 2: Konfigurieren Sie Ihre Attributaktionen

Das folgende Beispiel definiert ein `attributeActionsOnEncrypt` Wörterbuch, das Beispiele für Attributaktionen für ein Tabellenelement darstellt.

> **Note**
>
> Das folgende Beispiel definiert keine Attribute alsSIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT. Wenn Sie SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT Attribute angeben, müssen auch die Partitions- und Sortierattribute angegeben werdenSIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT.

```
var attributeActionsOnEncrypt = new Dictionary<string, CryptoAction>
{
    ["partition_key"] = CryptoAction.SIGN_ONLY, // The partition attribute must be
 SIGN_ONLY
    ["sort_key"] = CryptoAction.SIGN_ONLY, // The sort attribute must be SIGN_ONLY
    ["attribute1"] = CryptoAction.ENCRYPT_AND_SIGN,
    ["attribute2"] = CryptoAction.SIGN_ONLY,
    [":attribute3"] = CryptoAction.DO_NOTHING
};
```

Schritt 3: Definieren Sie, welche Attribute von den Signaturen ausgeschlossen werden

Im folgenden Beispiel wird davon ausgegangen, dass alle `DO_NOTHING` Attribute das eindeutige Präfix `":"` haben, und verwendet dieses Präfix, um die zulässigen Attribute ohne Vorzeichen zu definieren. Der Client geht davon aus, dass alle Attributnamen mit dem Präfix `":"` von den

Signaturen ausgeschlossen sind. Weitere Informationen finden Sie unter Allowed unsigned attributes.

```
const String unsignAttrPrefix = ":";
```

Schritt 4: Definieren Sie die Konfiguration der DynamoDB-Tabellenverschlüsselung

Das folgende Beispiel definiert eine `tableConfigs` Map, die die Verschlüsselungskonfiguration für diese DynamoDB-Tabelle darstellt.

In diesem Beispiel wird der DynamoDB-Tabellenname als logischer Tabellenname angegeben. Es wird dringend empfohlen, Ihren DynamoDB-Tabellennamen als logischen Tabellennamen anzugeben, wenn Sie Ihre Verschlüsselungskonfiguration zum ersten Mal definieren. Weitere Informationen finden Sie unter Verschlüsselungskonfiguration im AWS Database Encryption SDK für DynamoDB.

> **Note**
>
> Um durchsuchbare Verschlüsselung oder signierte Beacons zu verwenden, müssen Sie die auch `SearchConfig`in Ihre Verschlüsselungskonfiguration aufnehmen.

```
Dictionary<String, DynamoDbTableEncryptionConfig> tableConfigs =
    new Dictionary<String, DynamoDbTableEncryptionConfig>();
DynamoDbTableEncryptionConfig config = new DynamoDbTableEncryptionConfig
{
    LogicalTableName = ddbTableName,
    PartitionKeyName = "partition_key",
    SortKeyName = "sort_key",
    AttributeActionsOnEncrypt = attributeActionsOnEncrypt,
    Keyring = kmsKeyring,
    AllowedUnsignedAttributePrefix = unsignAttrPrefix
};
tableConfigs.Add(ddbTableName, config);
```

Schritt 5: Einen neuen AWS SDK-DynamoDB-Client erstellen

Im folgenden Beispiel wird ein neuer AWS SDK-DynamoDB-Client mit dem `TableEncryptionConfigs` aus Schritt 4 erstellt.

```
var ddb = new Client.DynamoDbClient(
```

```
    new DynamoDbTablesEncryptionConfig { TableEncryptionConfigs = tableConfigs });
```

Schritt 6: Verschlüsseln und Signieren eines DynamoDB-Tabellenelements

Das folgende Beispiel definiert ein `item` Dictionary, das ein Beispieltabellenelement darstellt und das Element in die DynamoDB-Tabelle einfügt. Das Element wird clientseitig verschlüsselt und signiert, bevor es an DynamoDB gesendet wird.

```
var item = new Dictionary<String, AttributeValue>
{
    ["partition_key"] = new AttributeValue("BasicPutGetExample"),
    ["sort_key"] = new AttributeValue { N = "0" },
    ["attribute1"] = new AttributeValue("encrypt and sign me!"),
    ["attribute2"] = new AttributeValue("sign me!"),
    [":attribute3"] = new AttributeValue("ignore me!")
};

PutItemRequest putRequest = new PutItemRequest
{
    TableName = ddbTableName,
    Item = item
};

PutItemResponse putResponse = await ddb.PutItemAsync(putRequest);
```

Verwenden Sie die untergeordnete Ebene **`DynamoDbItemEncryptor`**

Das folgende Beispiel zeigt, wie Sie die untergeordnete Ebene `DynamoDbItemEncryptor` mit einem AWS KMS Schlüsselbund verwenden, um Tabellenelemente direkt zu verschlüsseln und zu signieren. Das `DynamoDbItemEncryptor` fügt das Element nicht in Ihre DynamoDB-Tabelle ein.

Sie können jeden unterstützten Schlüsselbund mit dem DynamoDB Enhanced Client verwenden, wir empfehlen jedoch, wann immer möglich, einen der AWS KMS Schlüsselringe zu verwenden.

> **Note**
>
> Die untergeordnete Ebene unterstützt keine durchsuchbare Verschlüsselung. `DynamoDbItemEncryptor` Verwenden Sie das AWS Low-Level-Datenbankverschlüsselungs-SDK für DynamoDB-API, um durchsuchbare Verschlüsselung zu verwenden.

Sehen Sie sich das vollständige Codebeispiel an: .cs ItemEncryptDecryptExample

Schritt 1: Erstellen Sie den Schlüsselbund AWS KMS

Das folgende Beispiel verwendet`CreateAwsKmsMrkMultiKeyring`, um einen AWS KMS Schlüsselbund mit einem symmetrischen Verschlüsselungs-KMS-Schlüssel zu erstellen. Die `CreateAwsKmsMrkMultiKeyring` Methode stellt sicher, dass der Schlüsselbund sowohl Schlüssel mit einer Region als auch Schlüssel mit mehreren Regionen korrekt verarbeitet.

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var keyringInput = new CreateAwsKmsMrkMultiKeyringInput { Generator = kmsKeyId };
var kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

Schritt 2: Konfigurieren Sie Ihre Attributaktionen

Das folgende Beispiel definiert ein `attributeActionsOnEncrypt` Wörterbuch, das Beispiele für Attributaktionen für ein Tabellenelement darstellt.

> **Note**
>
> Das folgende Beispiel definiert keine Attribute als`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Wenn Sie `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Attribute angeben, müssen auch die Partitions- und Sortierattribute angegeben werden`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

```
var attributeActionsOnEncrypt = new Dictionary<String, CryptoAction>
{
    ["partition_key"] = CryptoAction.SIGN_ONLY, // The partition attribute must be
 SIGN_ONLY
    ["sort_key"] = CryptoAction.SIGN_ONLY, // The sort attribute must be SIGN_ONLY
    ["attribute1"] = CryptoAction.ENCRYPT_AND_SIGN,
    ["attribute2"] = CryptoAction.SIGN_ONLY,
    [":attribute3"] = CryptoAction.DO_NOTHING
};
```

Schritt 3: Definieren Sie, welche Attribute von den Signaturen ausgeschlossen werden

Im folgenden Beispiel wird davon ausgegangen, dass alle `DO_NOTHING` Attribute das eindeutige Präfix `":"` haben, und verwendet dieses Präfix, um die zulässigen Attribute ohne Vorzeichen

zu definieren. Der Client geht davon aus, dass alle Attributnamen mit dem Präfix `":"` von den Signaturen ausgeschlossen sind. Weitere Informationen finden Sie unter Allowed unsigned attributes.

```
String unsignAttrPrefix = ":";
```

Schritt 4: Definieren Sie die **`DynamoDbItemEncryptor`** Konfiguration

Das folgende Beispiel definiert die Konfiguration für die`DynamoDbItemEncryptor`.

In diesem Beispiel wird der DynamoDB-Tabellenname als logischer Tabellenname angegeben. Es wird dringend empfohlen, Ihren DynamoDB-Tabellennamen als logischen Tabellennamen anzugeben, wenn Sie Ihre Verschlüsselungskonfiguration zum ersten Mal definieren. Weitere Informationen finden Sie unter Verschlüsselungskonfiguration im AWS Database Encryption SDK für DynamoDB.

```
var config = new DynamoDbItemEncryptorConfig
{
    LogicalTableName = ddbTableName,
    PartitionKeyName = "partition_key",
    SortKeyName = "sort_key",
    AttributeActionsOnEncrypt = attributeActionsOnEncrypt,
    Keyring = kmsKeyring,
    AllowedUnsignedAttributePrefix = unsignAttrPrefix
};
```

Schritt 5: Erstellen Sie das **`DynamoDbItemEncryptor`**

Im folgenden Beispiel wird `DynamoDbItemEncryptor` mit dem `config` aus Schritt 4 ein neues erstellt.

```
var itemEncryptor = new DynamoDbItemEncryptor(config);
```

Schritt 6: Verschlüsseln und signieren Sie ein Tabellenelement direkt

Im folgenden Beispiel wird ein Element direkt verschlüsselt und signiert mit dem. `DynamoDbItemEncryptor` Das `DynamoDbItemEncryptor` fügt das Element nicht in Ihre DynamoDB-Tabelle ein.

```
var originalItem = new Dictionary<String, AttributeValue>
{
```

```
    ["partition_key"] = new AttributeValue("ItemEncryptDecryptExample"),
    ["sort_key"] = new AttributeValue { N = "0" },
    ["attribute1"] = new AttributeValue("encrypt and sign me!"),
    ["attribute2"] = new AttributeValue("sign me!"),
    [":attribute3"] = new AttributeValue("ignore me!")
};

var encryptedItem = itemEncryptor.EncryptItem(
    new EncryptItemInput { PlaintextItem = originalItem }
).EncryptedItem;
```

Konfigurieren Sie eine bestehende DynamoDB-Tabelle für die Verwendung des AWS Database Encryption SDK für DynamoDB

Mit Version 3. x der clientseitigen .NET-Verschlüsselungsbibliothek für DynamoDB können Sie Ihre vorhandenen Amazon DynamoDB-Tabellen für die clientseitige Verschlüsselung konfigurieren. Dieses Thema enthält Anleitungen zu den drei Schritten, die Sie ausführen müssen, um Version 3 hinzuzufügen. x zu einer vorhandenen, gefüllten DynamoDB-Tabelle.

Schritt 1: Bereiten Sie das Lesen und Schreiben verschlüsselter Elemente vor

Gehen Sie wie folgt vor, um Ihren AWS Database Encryption SDK-Client auf das Lesen und Schreiben verschlüsselter Elemente vorzubereiten. Nachdem Sie die folgenden Änderungen vorgenommen haben, liest und schreibt Ihr Client weiterhin Klartext-Elemente. Neue Elemente, die in die Tabelle geschrieben werden, werden nicht verschlüsselt oder signiert, aber es kann verschlüsselte Elemente entschlüsseln, sobald sie erscheinen. Diese Änderungen bereiten den Client darauf vor, mit der Verschlüsselung neuer Elemente zu beginnen. Die folgenden Änderungen müssen auf jedem Lesegerät installiert werden, bevor Sie mit dem nächsten Schritt fortfahren.

1. Definieren Sie Ihre Attributaktionen

   Erstellen Sie ein Objektmodell, um zu definieren, welche Attributwerte verschlüsselt und signiert werden, welche nur signiert und welche ignoriert werden.

   Standardmäßig sind Primärschlüsselattribute signiert, aber nicht verschlüsselt (`SIGN_ONLY`), und alle anderen Attribute sind verschlüsselt und signiert (`ENCRYPT_AND_SIGN`).

   Geben Sie `ENCRYPT_AND_SIGN` an, dass ein Attribut verschlüsselt und signiert werden soll. Geben Sie `SIGN_ONLY` an, dass ein Attribut signiert, aber nicht verschlüsselt werden soll. Geben Sie `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` an, dass das Attribut signiert und in den

Verschlüsselungskontext aufgenommen werden soll. Sie können ein Attribut nicht verschlüsseln, ohne es auch zu signieren. Geben Sie `DO_NOTHING` an, ob ein Attribut ignoriert werden soll. Weitere Informationen finden Sie unter Attributaktionen im AWS Database Encryption SDK für DynamoDB.

> **Note**
>
> Wenn Sie `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Attribute angeben, müssen auch die Partitions- und Sortierattribute angegeben werden`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

```
var attributeActionsOnEncrypt = new Dictionary<string, CryptoAction>
{
    ["partition_key"] = CryptoAction.SIGN_ONLY, // The partition attribute must be
 SIGN_ONLY
    ["sort_key"] = CryptoAction.SIGN_ONLY, // The sort attribute must be SIGN_ONLY
    ["attribute1"] = CryptoAction.ENCRYPT_AND_SIGN,
    ["attribute2"] = CryptoAction.SIGN_ONLY,
    [":attribute3"] = CryptoAction.DO_NOTHING
};
```

2. Definieren Sie, welche Attribute von den Signaturen ausgeschlossen werden

   Im folgenden Beispiel wird davon ausgegangen, dass alle `DO_NOTHING` Attribute das eindeutige Präfix `":"` haben, und verwendet dieses Präfix, um die zulässigen Attribute ohne Vorzeichen zu definieren. Der Client geht davon aus, dass alle Attributnamen mit dem Präfix `":"` von den Signaturen ausgeschlossen sind. Weitere Informationen finden Sie unter Allowed unsigned attributes.

   ```
   const String unsignAttrPrefix = ":";
   ```

3. Erstellen Sie einen Schlüsselbund

   Im folgenden Beispiel wird ein AWS KMS Schlüsselbund erstellt. Der AWS KMS Schlüsselbund verwendet symmetrische Verschlüsselung oder asymmetrisches RSA, um Datenschlüssel AWS KMS keys zu generieren, zu verschlüsseln und zu entschlüsseln.

   In diesem Beispiel wird ein AWS KMS Schlüsselbund `CreateMrkMultiKeyring` mit einem KMS-Schlüssel mit symmetrischer Verschlüsselung erstellt. Die

`CreateAwsKmsMrkMultiKeyring` Methode stellt sicher, dass der Schlüsselbund sowohl Schlüssel mit einer Region als auch Schlüssel mit mehreren Regionen korrekt verarbeitet.

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var keyringInput = new CreateAwsKmsMrkMultiKeyringInput { Generator = kmsKeyId };
var kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

4. Definieren Sie die Konfiguration der DynamoDB-Tabellenverschlüsselung

   Das folgende Beispiel definiert eine `tableConfigs` Map, die die Verschlüsselungskonfiguration für diese DynamoDB-Tabelle darstellt.

   In diesem Beispiel wird der DynamoDB-Tabellenname als logischer Tabellenname angegeben. Es wird dringend empfohlen, Ihren DynamoDB-Tabellennamen als logischen Tabellennamen anzugeben, wenn Sie Ihre Verschlüsselungskonfiguration zum ersten Mal definieren.

   Sie müssen `FORCE_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT` als Klartext-Override angeben. Diese Richtlinie liest und schreibt weiterhin Klartext-Elemente, liest verschlüsselte Elemente und bereitet den Client darauf vor, verschlüsselte Elemente zu schreiben.

   Weitere Hinweise zu den Werten, die in der Konfiguration der Tabellenverschlüsselung enthalten sind, finden Sie unterVerschlüsselungskonfiguration in der AWS Datenbankverschlüsselungs-SDK für DynamoDB.

```
Dictionary<String, DynamoDbTableEncryptionConfig> tableConfigs =
    new Dictionary<String, DynamoDbTableEncryptionConfig>();
DynamoDbTableEncryptionConfig config = new DynamoDbTableEncryptionConfig
{
    LogicalTableName = ddbTableName,
    PartitionKeyName = "partition_key",
    SortKeyName = "sort_key",
    AttributeActionsOnEncrypt = attributeActionsOnEncrypt,
    Keyring = kmsKeyring,
    AllowedUnsignedAttributePrefix = unsignAttrPrefix,
    PlaintextOverride = FORCE_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT
};
tableConfigs.Add(ddbTableName, config);
```

5. Einen neuen AWS SDK-DynamoDB-Client erstellen

   Im folgenden Beispiel wird ein neuer AWS SDK-DynamoDB-Client mit dem `TableEncryptionConfigs` aus Schritt 4 erstellt.

```
var ddb = new Client.DynamoDbClient(
    new DynamoDbTablesEncryptionConfig { TableEncryptionConfigs = tableConfigs });
```

Schritt 2: Schreiben Sie verschlüsselte und signierte Elemente

Aktualisieren Sie die Klartext-Richtlinie in Ihrer Tabellenverschlüsselungskonfiguration, damit der Client verschlüsselte und signierte Elemente schreiben kann. Nachdem Sie die folgende Änderung implementiert haben, verschlüsselt und signiert der Client neue Elemente auf der Grundlage der Attributaktionen, die Sie in Schritt 1 konfiguriert haben. Der Client kann Klartext-Elemente sowie verschlüsselte und signierte Elemente lesen.

Bevor Sie mit Schritt 3 fortfahren, müssen Sie alle vorhandenen Klartextelemente in Ihrer Tabelle verschlüsseln und signieren. Es gibt keine einzelne Metrik oder Abfrage, die Sie ausführen können, um Ihre vorhandenen Klartextelemente schnell zu verschlüsseln. Verwenden Sie den Prozess, der für Ihr System am sinnvollsten ist. Sie könnten beispielsweise einen asynchronen Prozess verwenden, der die Tabelle langsam scannt und dann die Elemente mithilfe der von Ihnen definierten Attributaktionen und der Verschlüsselungskonfiguration neu schreibt. Um die Klartext-Elemente in Ihrer Tabelle zu identifizieren, empfehlen wir, nach allen Elementen zu suchen, die nicht die `aws_dbe_foot` Attribute `aws_dbe_head` und enthalten, die das AWS Database Encryption SDK Elementen hinzufügt, wenn sie verschlüsselt und signiert sind.

Im folgenden Beispiel wird die Konfiguration der Tabellenverschlüsselung aus Schritt 1 aktualisiert. Sie müssen die Klartext-Override mit `FORBID_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT` aktualisieren. Diese Richtlinie liest weiterhin Klartext-Elemente, liest und schreibt aber auch verschlüsselte Elemente. Erstellen Sie einen neuen AWS SDK-DynamoDB-Client mit dem aktualisierten. `TableEncryptionConfigs`

```
Dictionary<String, DynamoDbTableEncryptionConfig> tableConfigs =
    new Dictionary<String, DynamoDbTableEncryptionConfig>();
DynamoDbTableEncryptionConfig config = new DynamoDbTableEncryptionConfig
{
    LogicalTableName = ddbTableName,
    PartitionKeyName = "partition_key",
    SortKeyName = "sort_key",
    AttributeActionsOnEncrypt = attributeActionsOnEncrypt,
    Keyring = kmsKeyring,
    AllowedUnsignedAttributePrefix = unsignAttrPrefix,
    PlaintextOverride = FORBID_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT
};
```

```
tableConfigs.Add(ddbTableName, config);
```

Schritt 3: Nur verschlüsselte und signierte Elemente lesen

Nachdem Sie alle Ihre Elemente verschlüsselt und signiert haben, aktualisieren Sie die Klartext-Überschreibung in Ihrer Tabellenverschlüsselungskonfiguration, sodass der Client nur verschlüsselte und signierte Elemente lesen und schreiben kann. Nachdem Sie die folgende Änderung implementiert haben, verschlüsselt und signiert der Client neue Elemente auf der Grundlage der Attributaktionen, die Sie in Schritt 1 konfiguriert haben. Der Client kann nur verschlüsselte und signierte Elemente lesen.

Im folgenden Beispiel wird die Konfiguration der Tabellenverschlüsselung aus Schritt 2 aktualisiert. Sie können entweder die Klartext-Override mit der Klartext-Richtlinie aktualisieren `FORBID_WRITE_PLAINTEXT_FORBID_READ_PLAINTEXT` oder die Klartext-Richtlinie aus Ihrer Konfiguration entfernen. Der Client liest und schreibt standardmäßig nur verschlüsselte und signierte Elemente. Erstellen Sie einen neuen AWS SDK-DynamoDB-Client mit dem aktualisierten. `TableEncryptionConfigs`

```
Dictionary<String, DynamoDbTableEncryptionConfig> tableConfigs =
    new Dictionary<String, DynamoDbTableEncryptionConfig>();
DynamoDbTableEncryptionConfig config = new DynamoDbTableEncryptionConfig
{
    LogicalTableName = ddbTableName,
    PartitionKeyName = "partition_key",
    SortKeyName = "sort_key",
    AttributeActionsOnEncrypt = attributeActionsOnEncrypt,
    Keyring = kmsKeyring,
    AllowedUnsignedAttributePrefix = unsignAttrPrefix,
    // Optional: you can also remove the plaintext policy from your configuration
    PlaintextOverride = FORBID_WRITE_PLAINTEXT_FORBID_READ_PLAINTEXT
};
tableConfigs.Add(ddbTableName, config);
```

Rust

In diesem Thema wird erklärt, wie Version 1 installiert und verwendet wird. x der clientseitigen Rust-Verschlüsselungsbibliothek für DynamoDB. Einzelheiten zur Programmierung mit dem AWS Database Encryption SDK für DynamoDB finden Sie in den Rust-Beispielen im aws-database-encryption-sdk -dynamodb-Repository unter. GitHub

Alle Programmiersprachenimplementierungen des AWS Database Encryption SDK für DynamoDB sind interoperabel.

Themen

Voraussetzungen

Bevor Sie die clientseitige Rust-Verschlüsselungsbibliothek für DynamoDB installieren, stellen Sie sicher, dass Sie die folgenden Voraussetzungen erfüllen.

Installieren Sie Rust und Cargo

Installieren Sie die aktuelle stabile Version von Rust mit Rustup.

Weitere Informationen zum Herunterladen und Installieren von Rustup finden Sie in den Installationsverfahren in The Cargo Book.

Installation

Die clientseitige Rust-Verschlüsselungsbibliothek für DynamoDB ist als Crate auf Crates.io verfügbar. aws-db-esdk Einzelheiten zur Installation und Erstellung der Bibliothek finden Sie in der Datei README.md im Repository -dynamodb. aws-database-encryption-sdk GitHub

manuell

Um die clientseitige Rust-Verschlüsselungsbibliothek für DynamoDB zu installieren, klonen Sie das -dynamodb-Repository oder laden Sie es herunter. aws-database-encryption-sdk GitHub

Installieren der neuesten Version

Führen Sie den folgenden Cargo-Befehl in Ihrem Projektverzeichnis aus:

```
cargo add aws-db-esdk
```

Oder fügen Sie Ihrer Cargo.toml die folgende Zeile hinzu:

```
aws-db-esdk = "<version>"
```

Verwendung der clientseitigen Rust-Verschlüsselungsbibliothek für DynamoDB

In diesem Thema werden einige der Funktionen und Hilfsklassen in Version 1 erklärt. x der clientseitigen Rust-Verschlüsselungsbibliothek für DynamoDB.

Einzelheiten zur Programmierung mit der clientseitigen Rust-Verschlüsselungsbibliothek für DynamoDB finden Sie in den Rust-Beispielen im -dynamodb-Repository unter. aws-database-encryption-sdk GitHub

Themen

- Elementverschlüssler
- Attributaktionen im AWS Database Encryption SDK für DynamoDB
- Verschlüsselungskonfiguration im AWS Database Encryption SDK für DynamoDB
- Elemente mit dem Database Encryption SDK aktualisieren AWS

Elementverschlüssler

Im Kern ist das AWS Database Encryption SDK für DynamoDB ein Elementverschlüsseler. Sie können Version 1 verwenden. x der clientseitigen Rust-Verschlüsselungsbibliothek für DynamoDB, um Ihre DynamoDB-Tabellenelemente auf folgende Weise zu verschlüsseln, zu signieren, zu verifizieren und zu entschlüsseln.

Das AWS Low-Level-Datenbankverschlüsselungs-SDK für die DynamoDB-API

Sie können Ihre Tabellenverschlüsselungskonfiguration verwenden, um einen DynamoDB-Client zu erstellen, der Elemente automatisch clientseitig mit Ihren DynamoDB-Anfragen verschlüsselt und signiert. `PutItem`

Sie müssen das AWS Low-Level-Datenbankverschlüsselungs-SDK für DynamoDB-API verwenden, um durchsuchbare Verschlüsselung verwenden zu können.

Ein Beispiel, das zeigt, wie das AWS Low-Level-Datenbankverschlüsselungs-SDK für die DynamoDB-API verwendet wird, finden Sie unter basic_get_put_example.rs im -dynamodb-Repository unter. aws-database-encryption-sdk GitHub

Die untergeordnete Ebene **`DynamoDbItemEncryptor`**

Die untergeordnete Ebene verschlüsselt und signiert oder entschlüsselt und verifiziert Ihre Tabellenelemente `DynamoDbItemEncryptor` direkt, ohne DynamoDB aufzurufen. Es stellt keine DynamoDB `PutItem` oder `GetItem` Anfragen. Sie können beispielsweise die untergeordnete

Ebene verwenden, `DynamoDbItemEncryptor` um ein DynamoDB-Element, das Sie bereits abgerufen haben, direkt zu entschlüsseln und zu verifizieren.

Die untergeordnete Ebene unterstützt keine durchsuchbare VerschlüsselungDynamoDbItemEncryptor.

Ein Beispiel, das zeigt, wie die untergeordnete Ebene verwendet wird, finden Sie unter item_encrypt_decrypt.rs im `DynamoDbItemEncryptor` -dynamodb-Repository unter. aws-database-encryption-sdk GitHub

Attributaktionen im AWS Database Encryption SDK für DynamoDB

Attributaktionen bestimmen, welche Attributwerte verschlüsselt und signiert werden, welche nur signiert sind, welche signiert und in den Verschlüsselungskontext aufgenommen werden und welche ignoriert werden.

Um Attributaktionen mit dem Rust-Client zu spezifizieren, definieren Sie Attributaktionen manuell mithilfe eines Objektmodells. Spezifizieren Sie Ihre Attributaktionen, indem Sie ein `HashMap` Objekt erstellen, in dem die Name-Wert-Paare für Attributnamen und die angegebenen Aktionen stehen.

Geben Sie `ENCRYPT_AND_SIGN` an, dass ein Attribut verschlüsselt und signiert werden soll. Geben Sie `SIGN_ONLY` an, dass ein Attribut signiert, aber nicht verschlüsselt werden soll. Geben Sie `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` an, dass ein Attribut signiert und in den Verschlüsselungskontext aufgenommen werden soll. Sie können ein Attribut nicht verschlüsseln, ohne es auch zu signieren. Geben Sie `DO_NOTHING` an, ob ein Attribut ignoriert werden soll.

Die Partitions- und Sortierattribute müssen entweder `SIGN_ONLY` oder lautenSIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT. Wenn Sie Attribute als definierenSIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT, müssen dies auch die Partitions- und Sortierattribute seinSIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT.

> **Note**
>
> Nachdem Sie Ihre Attributaktionen definiert haben, müssen Sie definieren, welche Attribute von den Signaturen ausgeschlossen werden. Um das future Hinzufügen neuer Attribute ohne Vorzeichen zu vereinfachen, empfehlen wir, ein eindeutiges Präfix (wie ":„) zu wählen, um Ihre vorzeichenlosen Attribute zu identifizieren. Nehmen Sie dieses Präfix in den Attributnamen für alle Attribute auf, die Sie bei der Definition Ihres DynamoDB-Schemas und Ihrer Attributaktionen markiert `DO_NOTHING` haben.

Das folgende Objektmodell zeigt, wie SieENCRYPT_AND_SIGN, SIGN_ONLYSIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT, und DO_NOTHING Attributaktionen mit dem Rust-Client angeben. In diesem Beispiel wird das Präfix ":" verwendet, um DO_NOTHING Attribute zu identifizieren.

```
let attribute_actions_on_encrypt = HashMap::from([
    ("partition_key".to_string(), CryptoAction::SignOnly),
    ("sort_key".to_string(), CryptoAction::SignOnly),
    ("attribute1".to_string(), CryptoAction::EncryptAndSign),
    ("attribute2".to_string(), CryptoAction::SignOnly),
    (":attribute3".to_string(), CryptoAction::DoNothing),
]);
```

Verschlüsselungskonfiguration im AWS Database Encryption SDK für DynamoDB

Wenn Sie das AWS Database Encryption SDK verwenden, müssen Sie explizit eine Verschlüsselungskonfiguration für Ihre DynamoDB-Tabelle definieren. Die in Ihrer Verschlüsselungskonfiguration erforderlichen Werte hängen davon ab, ob Sie Ihre Attributaktionen manuell oder mit einer annotierten Datenklasse definiert haben.

Der folgende Ausschnitt definiert eine DynamoDB-Tabellenverschlüsselungskonfiguration unter Verwendung des AWS Low-Level-Datenbankverschlüsselungs-SDK für DynamoDB-API und zulässige unsignierte Attribute, die durch ein eindeutiges Präfix definiert sind.

```
let table_config = DynamoDbTableEncryptionConfig::builder()
    .logical_table_name(ddb_table_name)
    .partition_key_name("partition_key")
    .sort_key_name("sort_key")
    .attribute_actions_on_encrypt(attribute_actions_on_encrypt)
    .keyring(kms_keyring)
    .allowed_unsigned_attribute_prefix(UNSIGNED_ATTR_PREFIX)
    // Specifying an algorithm suite is optional
    .algorithm_suite_id(
        DbeAlgorithmSuiteId::AlgAes256GcmHkdfSha512CommitKeyEcdsaP384SymsigHmacSha384,
    )
    .build()?;

let table_configs = DynamoDbTablesEncryptionConfig::builder()
    .table_encryption_configs(HashMap::from([(ddb_table_name.to_string(),
 table_config)]))
    .build()?;
```

Logischer Tabellenname

Ein logischer Tabellenname für Ihre DynamoDB-Tabelle.

Der logische Tabellenname ist kryptografisch an alle in der Tabelle gespeicherten Daten gebunden, um DynamoDB-Wiederherstellungsvorgänge zu vereinfachen. Es wird dringend empfohlen, Ihren DynamoDB-Tabellennamen als logischen Tabellennamen anzugeben, wenn Sie Ihre Verschlüsselungskonfiguration zum ersten Mal definieren. Sie müssen immer denselben logischen Tabellennamen angeben. Damit die Entschlüsselung erfolgreich ist, muss der Name der logischen Tabelle mit dem Namen übereinstimmen, der bei der Verschlüsselung angegeben wurde. Falls sich Ihr DynamoDB-Tabellenname nach dem Wiederherstellen Ihrer DynamoDB-Tabelle aus einer Sicherung ändert, stellt der logische Tabellenname sicher, dass der Entschlüsselungsvorgang die Tabelle weiterhin erkennt.

Zulässige Attribute ohne Vorzeichen

Die `DO_NOTHING` in Ihren Attributaktionen markierten Attribute.

Die zulässigen Attribute ohne Vorzeichen teilen dem Client mit, welche Attribute von den Signaturen ausgeschlossen sind. Der Client geht davon aus, dass alle anderen Attribute in der Signatur enthalten sind. Beim Entschlüsseln eines Datensatzes bestimmt der Client dann aus den von Ihnen angegebenen zulässigen unsignierten Attributen, welche er überprüfen muss und welche ignoriert werden sollen. Sie können kein Attribut aus Ihren zulässigen Attributen ohne Vorzeichen entfernen.

Sie können die zulässigen Attribute ohne Vorzeichen explizit definieren, indem Sie ein Array erstellen, das alle Ihre `DO_NOTHING` Attribute auflistet. Sie können bei der Benennung Ihrer `DO_NOTHING` Attribute auch ein eindeutiges Präfix angeben und das Präfix verwenden, um dem Client mitzuteilen, welche Attribute vorzeichenlos sind. Wir empfehlen dringend, ein eindeutiges Präfix anzugeben, da dies das Hinzufügen eines neuen `DO_NOTHING` Attributs in der future vereinfacht. Weitere Informationen finden Sie unter Aktualisierung Ihres Datenmodells.

Wenn Sie kein Präfix für alle `DO_NOTHING` Attribute angeben, können Sie ein `allowedUnsignedAttributes` Array konfigurieren, das explizit alle Attribute auflistet, von denen der Client erwarten sollte, dass sie nicht signiert sind, wenn er sie bei der Entschlüsselung findet. Sie sollten Ihre erlaubten vorzeichenlosen Attribute nur dann explizit definieren, wenn dies unbedingt erforderlich ist.

Suchkonfiguration (optional)

Das `SearchConfig` definiert die Beacon-Version.

Der `SearchConfig` muss angegeben werden, um durchsuchbare Verschlüsselung oder signierte Beacons verwenden zu können.

Algorithm Suite (optional)

Die `algorithmSuiteId` definiert, welche Algorithmus-Suite das AWS Database Encryption SDK verwendet.

Sofern Sie nicht explizit eine alternative Algorithmussuite angeben, verwendet das AWS Database Encryption SDK die Standard-Algorithmussuite. Die Standard-Algorithmussuite verwendet den AES-GCM-Algorithmus mit Schlüsselableitung, digitalen Signaturen und Schlüsselzusage. Obwohl die Standard-Algorithmus-Suite wahrscheinlich für die meisten Anwendungen geeignet ist, können Sie auch eine alternative Algorithmussuite wählen. Einige Vertrauensmodelle würden beispielsweise durch eine Algorithmus-Suite ohne digitale Signaturen erfüllt. Hinweise zu den Algorithmus-Suites, die das AWS Database Encryption SDK unterstützt, finden Sie unterUnterstützte Algorithmus-Suiten im AWS Database Encryption SDK.

Um die AES-GCM-Algorithmussuite ohne digitale ECDSA-Signaturen auszuwählen, nehmen Sie den folgenden Ausschnitt in Ihre Tabellenverschlüsselungskonfiguration auf.

```
.algorithm_suite_id(
    DbeAlgorithmSuiteId::AlgAes256GcmHkdfSha512CommitKeyEcdsaP384SymsigHmacSha384,
)
```

Elemente mit dem Database Encryption SDK aktualisieren AWS

Das AWS Database Encryption SDK unterstützt ddb: nicht UpdateItem für Elemente, die verschlüsselte oder signierte Attribute enthalten. Um ein verschlüsseltes oder signiertes Attribut zu aktualisieren, müssen Sie ddb: verwenden. PutItem Wenn Sie in Ihrer `PutItem` Anfrage denselben Primärschlüssel wie ein vorhandenes Element angeben, ersetzt das neue Element das vorhandene Element vollständig.

Legacy-DynamoDB-Verschlüsselungsclient

Am 9. Juni 2023 wurde unsere clientseitige Verschlüsselungsbibliothek in Database Encryption SDK umbenannt. AWS Das AWS Database Encryption SDK unterstützt weiterhin ältere Versionen des DynamoDB Encryption Client. Weitere Informationen zu den verschiedenen Teilen der clientseitigen Verschlüsselungsbibliothek, die sich mit der Umbenennung geändert haben, finden Sie unter. Amazon DynamoDB Encryption Client umbenennen

Informationen zur Migration zur neuesten Version der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB finden Sie unter. Migrieren Sie auf Version 4.x

Themen

AWS Database Encryption SDK für DynamoDB-Versionsunterstützung

Die Themen im Kapitel Legacy enthalten Informationen zu Versionen 1. x —2. x des DynamoDB Encryption Client für Java und Versionen 1. x —3. x des DynamoDB Encryption Client für Python.

In der folgenden Tabelle sind die Sprachen und Versionen aufgeführt, die die clientseitige Verschlüsselung in Amazon DynamoDB unterstützen.

Programmiersprache	Version	Lebenszyklusphase der SDK-Hauptversion
Java	Versionen 1. x	End-of-Support Phase, gültig ab Juli 2022
Java	Versionen 2. x	Wartungsphase, bis August 2026
Python	Versionen 1. x	End-of-Support Phase, gültig ab Juli 2022
Python	Versionen 2. x	End-of-Support Phase, gültig ab Juli 2022
Python	Versionen 3. x	Allgemeine Verfügbarkeit (GA)

So funktioniert der DynamoDB Encryption Client

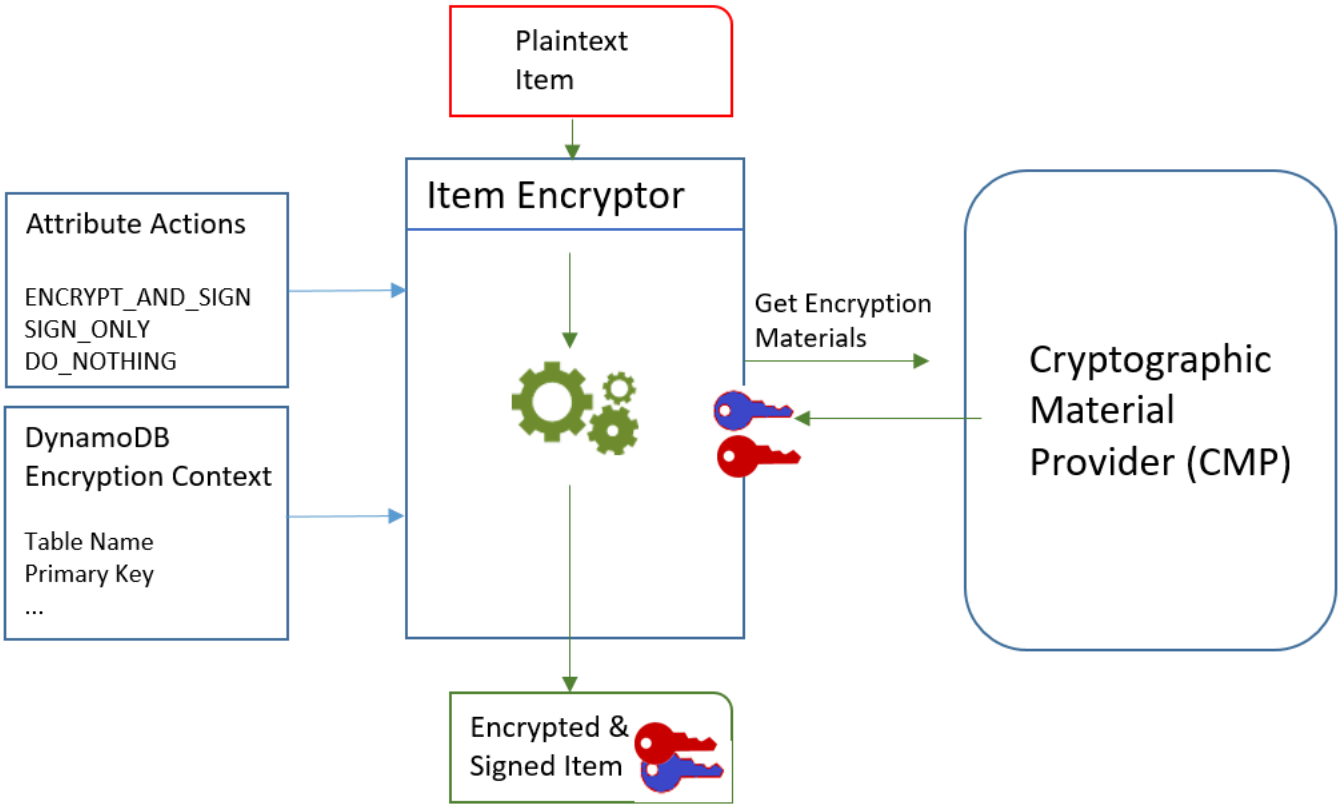
> **Note**
>
> Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt. AWS Das folgende Thema enthält Informationen zu Versionen 1. x —2. x des DynamoDB Encryption Client für Java und Versionen 1. x —3. x des DynamoDB Encryption Client für Python. Weitere Informationen finden Sie unter AWS Database Encryption SDK für DynamoDB-Versionsunterstützung.

Der DynamoDB Encryption Client wurde speziell für den Schutz der Daten entwickelt, die Sie in DynamoDB speichern. Die Bibliotheken enthalten sichere Implementierungen, die Sie erweitern oder unverändert verwenden können. Und die meisten Elemente werden durch abstrakte Elemente dargestellt, sodass Sie kompatible benutzerdefinierte Komponenten erstellen und verwenden können.

Verschlüsselung und Signieren von Tabellenelementen

Das Herzstück des DynamoDB Encryption Client ist ein Elementverschlüsseler, der Tabellenelemente verschlüsselt, signiert, verifiziert und entschlüsselt. Er nimmt Informationen über Ihre Tabellenelemente entgegen, sowie Anweisungen, welche Elemente zu verschlüsseln und zu signieren sind. Er ruft von einem Anbieter für kryptographisches Material, den Sie auswählen und konfigurieren, die Verschlüsselungsmaterialien ab, ebenso wie Anweisungen, wie man sie verwendet.

Das folgende Diagramm zeigt einen allgemeinen Überblick über dieses Verfahren.



Um ein Tabellenelement zu verschlüsseln und zu signieren, benötigt der DynamoDB Encryption Client:

- Informationen über die Tabelle. Es ruft Informationen über die Tabelle aus einem DynamoDB-Verschlüsselungskontext ab, den Sie angeben. Einige Helfer rufen die erforderlichen Informationen von DynamoDB ab und erstellen den DynamoDB-Verschlüsselungskontext für Sie.

  > **Note**
  >
  > Der DynamoDB-Verschlüsselungskontext im DynamoDB Encryption Client hat nichts mit dem Verschlüsselungskontext in AWS Key Management Service ()AWS KMS und dem zu tun. AWS Encryption SDK

- Welche Attribute zu verschlüsseln und zu signieren sind. Er erhält diese Informationen über die Tabelle von den Attribut-Aktionen, die Sie bereitstellen.
- Verschlüsselungsmaterialien, einschließlich Verschlüsselungs- und Signaturschlüssel. Er erhält diese von einem Anbieter kryptographischer Materialien (Cryptographic Materials Provider, CMP), den Sie auswählen und konfigurieren.

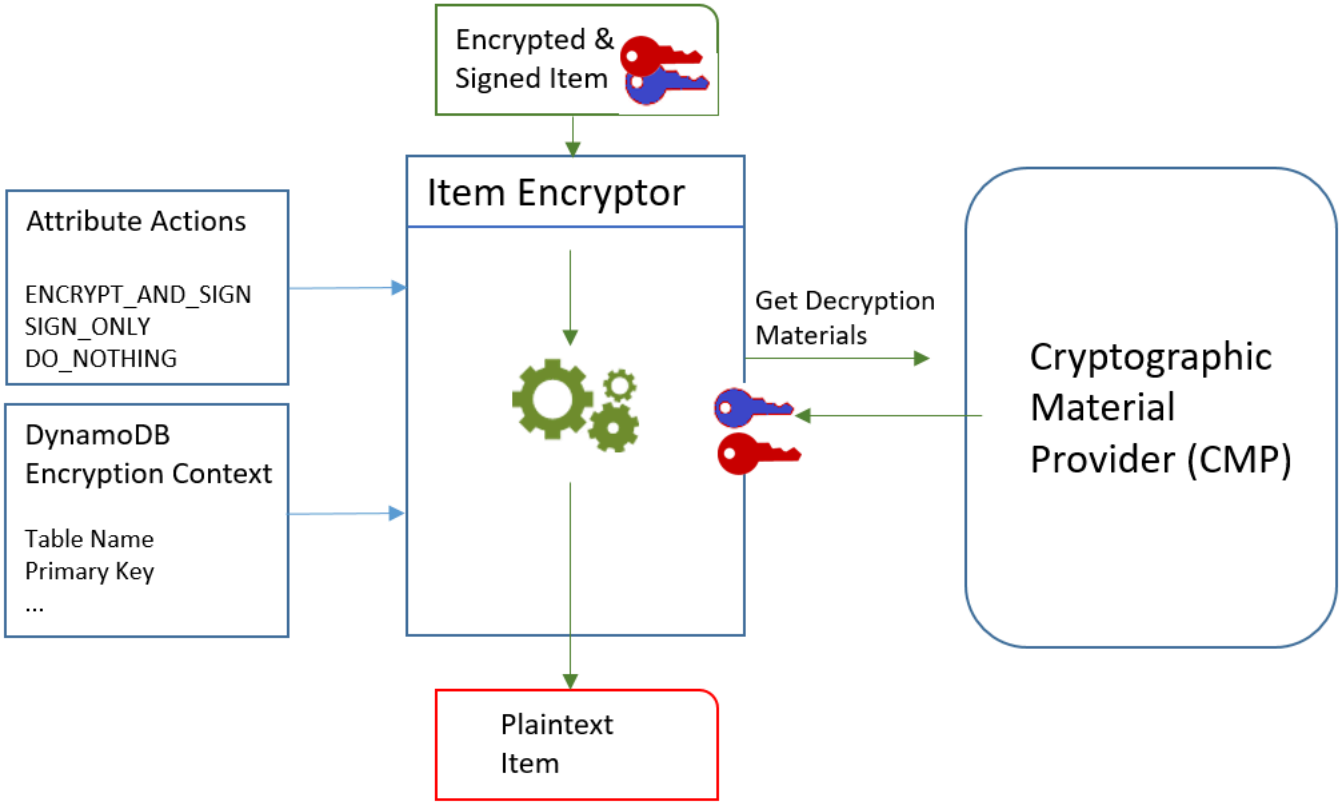
- Anweisungen für die Verschlüsselung und das Signieren des Elementes. Der CMP fügt der tatsächlichen Materialbeschreibung Anweisungen zur Verwendung der Verschlüsselungsmaterialien hinzu, einschließlich Verschlüsselungs- und Signaturalgorithmen.

Der Elementverschlüssler verwendet alle diese Dinge, um das Element zu verschlüsseln und zu signieren. Der Elementverschlüssler fügt dem Element außerdem zwei Attribute hinzu: ein Materialbeschreibungsattribut, das die Verschlüsselungs- und Signaturanweisungen (die tatsächliche Materialbeschreibung) enthält, und ein Attribut, das die Signatur enthält. Sie können direkt mit dem Elementverschlüsseler interagieren oder Helferfunktionen verwenden, die mit dem Elementverschlüssler interagieren, um ein sicheres Standardverhalten zu implementieren.

Das Ergebnis ist ein DynamoDB-Element mit verschlüsselten und signierten Daten.

Verifizieren und Entschlüsseln von Tabellenelementen

Diese Komponenten arbeiten auch zusammen, um Ihr Element zu verifizieren und zu entschlüsseln, wie in der folgenden Abbildung gezeigt.



Um ein Element zu verifizieren und zu entschlüsseln, benötigt der DynamoDB Encryption Client dieselben Komponenten, Komponenten mit derselben Konfiguration oder Komponenten, die speziell für die Entschlüsselung der Elemente entwickelt wurden, und zwar wie folgt:

- Informationen über die Tabelle aus dem DynamoDB-Verschlüsselungskontext.
- Welche Attribute zu überprüfen und zu entschlüsseln sind. Diese erhält er von den Attribut-Aktionen.
- Entschlüsselungsmaterialien, einschließlich Verifikations- und Entschlüsselungsschlüssel, von dem Anbieter kryptographischer Materialien (Cryptographic Materials Provider, CMP), den Sie auswählen und konfigurieren.

  Das verschlüsselte Element enthält keine Aufzeichnung des CMP, mit dem es verschlüsselt wurde. Sie müssen den gleichen CMP, einen CMP mit der gleichen Konfiguration oder einen CMP, der zum Entschlüsseln von Elementen vorgesehen ist, bereitstellen.
- Informationen darüber, wie das Element verschlüsselt und signiert wurde, einschließlich der Verschlüsselungs- und Signierungsalgorithmen. Der Client erhält sie vom Materialbeschreibungsattribut im Element.

Der Elementverschlüssler verwendet alle diese Dinge, um das Element zu verifizieren und zu entschlüsseln. Außerdem entfernt er die Materialbezeichnung und die Signaturattribute. Das Ergebnis ist ein DynamoDB-Element im Klartext-Format.

Konzepte des Amazon DynamoDB DynamoDB-Verschlüsselungsclients

> **Note**
>
> Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt. AWS Das folgende Thema enthält Informationen zu Versionen 1. x —2. x des DynamoDB Encryption Client für Java und Versionen 1. x —3. x des DynamoDB Encryption Client für Python. Weitere Informationen finden Sie unter AWS Database Encryption SDK für DynamoDB-Versionsunterstützung.

In diesem Thema werden die Konzepte und die Terminologie erklärt, die im Amazon DynamoDB Encryption Client verwendet werden.

Informationen zum Zusammenspiel der Komponenten des DynamoDB Encryption Client finden Sie unter. So funktioniert der DynamoDB Encryption Client

Themen

- Anbieter von kryptographischen Materialien (Cryptographic Materials Provider (CMP))
- Elementverschlüssler
- Attributaktionen
- Materialbeschreibung
- DynamoDB-Verschlüsselungsclient
- Provider-Store

Anbieter von kryptographischen Materialien (Cryptographic Materials Provider (CMP))

Bei der Implementierung des DynamoDB Encryption Client besteht eine Ihrer ersten Aufgaben darin, einen Anbieter für kryptografisches Material (CMP) (auch bekannt als Anbieter von Verschlüsselungsmaterialien) auszuwählen. Ihre Wahl bestimmt einen Großteil der restlichen Implementierung.

Ein Anbieter von kryptographischem Material (Cryptographic Materials Provider, CMP) erfasst und erstellt die kryptographischen Materialien und gibt sie zurück, die der Elementverschlüssler verwendet, um Ihre Tabellenelemente zu verschlüsseln und zu signieren. Der CMP bestimmt die zu verwendenden Verschlüsselungsalgorithmen, und wie Verschlüsselungs- und Signierschlüssel erzeugt und geschützt werden.

Der CMP interagiert mit dem Elementverschlüssler. Der Elementverschlüssler fordert vom CMP Ver- oder Entschlüsselungsmaterialien an, und der CMP gibt sie an den Elementverschlüsseler zurück. Dann verwendet der Elementverschlüssler die kryptographischen Materialien, um das Element zu verschlüsseln und zu signieren oder zu überprüfen und zu entschlüsseln.

Den CMP geben Sie bei der Konfiguration des Clients an. Sie können einen kompatiblen benutzerdefinierten CMP erstellen oder einen der vielen CMPs in der Bibliothek verwenden. Die meisten CMPs sind für mehrere Programmiersprachen verfügbar.

Elementverschlüssler

Der Elementverschlüsseler ist eine untergeordnete Komponente, die kryptografische Operationen für den DynamoDB Encryption Client ausführt. Er fordert kryptographisches Material von einem Anbieter

von kryptographischem Material (Cryptographic Materials Provider, CMP) an und verwendet dann die vom CMP zurückgegebenen Materialien, um Ihr Tabellenelement zu verschlüsseln und zu signieren oder zu verifizieren und zu entschlüsseln.

Sie können direkt mit dem Elementverschlüsseler interagieren oder die Helferklassen verwenden, die Ihre Bibliothek zur Verfügung stellt. Der DynamoDB Encryption Client für Java enthält beispielsweise eine `AttributeEncryptor` Hilfsklasse, die Sie mit dem verwenden können`DynamoDBMapper`, anstatt direkt mit dem `DynamoDBEncryptor` Elementverschlüsseler zu interagieren. Die Python-Bibliothek enthält die Helferklassen `EncryptedTable`, `EncryptedClient` und `EncryptedResource`, die für Sie mit dem Elementverschlüssler interagieren.

Attributaktionen

Attribut-Aktionen teilen dem Elementverschlüsseler mit, welche Aktionen er auf jedes Attribut des Elements anwenden soll.

Das Attribut-Aktionswerte können einer der folgenden sein:

- Verschlüsseln und signieren — Verschlüsselt den Attributwert. Nehmen Sie das Attribut (Name und Wert) in die Elementsignatur auf.
- Nur signieren — Schließt das Attribut in die Artikelsignatur ein.
- Nichts tun — Verschlüsseln oder signieren Sie das Attribut nicht.

Verwenden Sie für jedes Attribut, das vertrauliche Daten speichern kann, Verschlüsseln und signieren. Für Primärschlüsselattribute (Partitionsschlüssel und Sortierschlüssel) verwenden Sie Nur signieren. Das Materialverschlüsselungsattribut und das Signaturattribut werden wird nicht verschlüsselt oder signiert. Sie müssen für diese Attribute keine Attribut-Aktionen angeben.

Wählen Sie Ihre Attributaktionen sorgfältig aus. Verwenden Sie im Zweifelsfall Verschlüsseln und signieren. Sobald Sie den DynamoDB Encryption Client zum Schutz Ihrer Tabellenelemente verwendet haben, können Sie die Aktion für ein Attribut nicht mehr ändern, ohne einen Signaturvalidierungsfehler zu riskieren. Details hierzu finden Sie unter Ändern Ihres Datenmodells.

> **Warning**
>
> Verschlüsseln Sie die primären Schlüsselattribute nicht. Sie müssen im Klartext bleiben, damit DynamoDB das Element finden kann, ohne einen vollständigen Tabellenscan ausführen zu müssen.

Wenn der DynamoDB-Verschlüsselungskontext Ihre Primärschlüsselattribute identifiziert, gibt der Client einen Fehler aus, wenn Sie versuchen, sie zu verschlüsseln.

Die Technik, mit der Sie die Attribut-Aktionen festlegen, ist für jede Programmiersprache unterschiedlich. Sie kann auch spezifisch für Helferklassen sein, die Sie verwenden.

Weitere Informationen finden Sie in der Dokumentation Ihrer Programmiersprache.

- Python
- Java

Materialbeschreibung

Die Materialbeschreibung für ein verschlüsseltes Tabellenelement besteht aus Informationen, wie z. B. Verschlüsselungsalgorithmen, wie das Tabellenelement verschlüsselt und signiert wird. Ein Anbieter von kryptographischem Material (Cryptographic Materials Provider, CMP) zeichnet die Materialbeschreibung auf, wenn er die kryptographischen Materialien für die Verschlüsselung und die Signatur zusammenstellt. Später, wenn er kryptographische Materialien zusammenstellen muss, um das Element zu verifizieren und zu entschlüsseln, verwendet er die Materialbeschreibung als seinen Leitfaden.

Im DynamoDB Encryption Client bezieht sich die Materialbeschreibung auf drei verwandte Elemente:

Angeforderte Materialbeschreibung

Einige Anbieter kryptographischer Materialien (Cryptographic Materials Providers, CMPs) ermöglichen die Angabe erweiterter Optionen, wie beispielsweise eines Verschlüsselungsalgorithmus. Um Ihre Auswahlmöglichkeiten anzugeben, fügen Sie der Materialbeschreibungseigenschaft des DynamoDB-Verschlüsselungskontextes in Ihrer Anforderung zur Verschlüsselung eines Tabellenelements Name-Wert-Paare hinzu. Dieses Element wird als die angeforderte Materialbeschreibung bezeichnet. Die gültigen Werte in der angeforderten Materialbeschreibung werden durch den von Ihnen gewählten CMP definiert.

> **Note**
>
> Da die Materialbeschreibung sichere Standardwerte überschreiben kann, empfehlen wir Ihnen, die angeforderte Materialbeschreibung wegzulassen, es sei denn, Sie haben einen zwingenden Grund, sie zu verwenden.

Tatsächliche Materialbeschreibung

Die Materialbeschreibung auf, die der Anbieter kryptographischer Materialien (Cryptographic Materials Provider, CMP) zurückgibt, wird als tatsächliche Materialbeschreibung bezeichnet. Sie beschreibt die tatsächlichen Werte, die der CMP bei der Zusammenstellung der kryptographischen Materialien verwendet hat. Sie besteht in der Regel aus der angeforderten Materialbeschreibung, falls vorhanden, mit Ergänzungen und Änderungen.

Materialbeschreibungsattribut

Der Client speichert die tatsächliche Materialbeschreibung in dem Materialbeschreibungsattribut des verschlüsselten Elements. Der Name des Materialbeschreibungsattributs ist `amzn-ddb-map-desc`, der Wert ist die tatsächliche Materialbeschreibung. Der Client verwendet die Werte im Materialbeschreibungsattribut, um das Element zu überprüfen und zu entschlüsseln.

DynamoDB-Verschlüsselungsclient

Der DynamoDB-Verschlüsselungskontext liefert Informationen über die Tabelle und das Element an den Cryptographic Materials Provider (CMP). In fortgeschrittenen Implementierungen kann der DynamoDB-Verschlüsselungskontext eine angeforderte Materialbeschreibung enthalten.

Wenn Sie Tabellenelemente verschlüsseln, ist der DynamoDB-Verschlüsselungskontext kryptografisch an die verschlüsselten Attributwerte gebunden. Wenn beim Entschlüsseln der DynamoDB-Verschlüsselungskontext nicht exakt und unter Berücksichtigung der Groß- und Kleinschreibung mit dem DynamoDB-Verschlüsselungskontext übereinstimmt, der für die Verschlüsselung verwendet wurde, schlägt der Entschlüsselungsvorgang fehl. Wenn Sie direkt mit dem Elementverschlüsseler interagieren, müssen Sie beim Aufrufen einer Verschlüsselungs- oder Entschlüsselungsmethode einen DynamoDB-Verschlüsselungskontext angeben. Die meisten Helfer erstellen den DynamoDB-Verschlüsselungskontext für Sie.

> **Note**
>
> Der DynamoDB-Verschlüsselungskontext im DynamoDB Encryption Client hat nichts mit dem Verschlüsselungskontext in AWS Key Management Service ()AWS KMS und dem zu tun. AWS Encryption SDK

Der DynamoDB-Verschlüsselungskontext kann die folgenden Felder enthalten. Alle Felder und Werte sind optional.

- Name der Tabelle
- Partitionsschlüsselname
- Sortierschlüsselname
- Attribut-Namen-Wert-Paare
- Angeforderte Materialbeschreibung

Provider-Store

Ein Provider-Store ist eine Komponente, die Anbieter kryptographischer Materialien (Cryptographic Materials Providers, CMPs) zurückgibt. Der Provider-Store kann die CMPs erstellen oder von einer anderen Quelle beziehen, z. B. von einem anderen Provider-Store. Der Provider-Store speichert Versionen der von ihm erstellten CMPs in einem persistenten Speicher, in dem jeder gespeicherte CMP durch den Materialnamen des Anforderers und die Versionsnummer identifiziert wird.

Der neueste Anbieter im DynamoDB Encryption Client bezieht seine CMPs aus einem Provider-Speicher, aber Sie können den Provider-Speicher verwenden, um CMPs für jede Komponente bereitzustellen. Jeder Most Recent Provider ist einem Provider-Store zugeordnet, aber ein Provider-Store kann CMPs an mehrere Anforderer über mehrere Hosts liefern.

Der Provider-Store erstellt bei Bedarf neue Versionen von CMPs und gibt neue und bestehende Versionen zurück. Außerdem gibt er die neueste Versionsnummer für einen bestimmten Materialnamen zurück. Daran erkennt der Anforderer, dass der Provider-Store eine neue Version seines CMP hat, die er anfordern kann.

Der DynamoDB Encryption Client umfasst einen MetaStore, bei dem es sich um einen Provider-Speicher handelt, der verpackte CMPs mit Schlüsseln erstellt, die in DynamoDB gespeichert und mithilfe eines internen DynamoDB Encryption Clients verschlüsselt werden.

Weitere Informationen:

- Provider-Store: Java, Python
- MetaStore: Java, Python

Anbieter von kryptografischem Material

> **Note**
>
> Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt. AWS Das folgende Thema enthält Informationen zu Versionen 1. x —2. x des DynamoDB Encryption Client für Java und Versionen 1. x —3. x des DynamoDB Encryption Client für Python. Weitere Informationen finden Sie unter AWS Database Encryption SDK für DynamoDB-Versionsunterstützung.

Eine der wichtigsten Entscheidungen, die Sie bei der Verwendung des DynamoDB Encryption Client treffen, ist die Auswahl eines Anbieters für kryptografisches Material (CMP). Das CMP stellt kryptographisches Material zusammen und gibt es an den Elementverschlüssler zurück. Außerdem legt er fest, wie Verschlüsselungs- und Signierschlüssel generiert werden, ob für jedes Element neue Schlüsselmaterialien generiert oder wiederverwendet werden und welche Verschlüsselungs- und Signierungsalgorithmen verwendet werden.

Sie können eine CMP aus den Implementierungen auswählen, die in den DynamoDB Encryption Client-Bibliotheken bereitgestellt werden, oder eine kompatible benutzerdefinierte CMP erstellen. Welchen CMP Sie auswählen, hängt möglicherweise auch von der verwendeten Programmiersprache ab.

Dieses Thema beschreibt die gebräuchlichsten CMPs und bietet einige Ratschläge, um Ihnen bei der Auswahl des für Ihre Anwendung am besten geeigneten CMPs zu helfen.

Direct KMS Materials Provider

Der Direct KMS Materials Provider schützt Ihre Tabellenelemente unter einem That Never Leaves () AWS KMS keyunverschlüsselt. AWS Key Management ServiceAWS KMS Ihre Anwendung muss kein kryptographisches Material erzeugen oder verwalten. Da er die verwendet AWS KMS key , um eindeutige Verschlüsselungs- und Signaturschlüssel für jedes Element zu generieren, ruft dieser Anbieter AWS KMS jedes Mal auf, wenn er ein Element ver- oder entschlüsselt.

Wenn Sie verwenden AWS KMS und ein AWS KMS Aufruf pro Transaktion für Ihre Anwendung praktikabel ist, ist dieser Anbieter eine gute Wahl.

Details hierzu finden Sie unter Direct KMS Materials Provider.

Wrapped Materials Provider (Wrapped CMP)

Mit dem Wrapped Materials Provider (Wrapped CMP) können Sie Ihre Verpackungs- und Signaturschlüssel außerhalb des DynamoDB Encryption Client generieren und verwalten.

Der Wrapped CMP generiert einen eindeutigen Verschlüsselungsschlüssel für jedes Element. Dann verwendet er die von Ihnen bereitgestellten Wrapping- (oder Unwrapping-) und Signierschlüssel. Damit bestimmen Sie, wie die Wrapping- und Signierschlüssel erzeugt werden und ob sie für jedes Element eindeutig sind oder wiederverwendet werden. Der Wrapped CMP ist eine sichere Alternative zum Direct KMS Provider für Anwendungen, die kein kryptografisches Material verwenden AWS KMS und dieses sicher verwalten können.

Details hierzu finden Sie unter Wrapped Materials Provider.

Most Recent Provider

Der Most Recent Provider ist ein Anbieter kryptographischer Materialien (Cryptographic Materials Provider (CMP)), der auf die Arbeit mit einem Provider-Store ausgelegt ist. Es erhält CMPs aus dem Provider-Store und erhält die kryptographischen Materialien, die er zurückgibt, von den CMPs. Der Most Recent Provider verwendet in der Regel jeden CMP, um mehrere Anfragen nach kryptographischem Material zu erfüllen, aber Sie können die Funktionen des Provider-Stores verwenden, um zu steuern, wie oft Materialien wiederverwendet werden, um zu bestimmen, wie oft ein CMP rotiert wird, und sogar, um den Typ des CMP zu ändern, der verwendet wird, ohne den Most Recent Provider zu ändern.

Sie können den Most Recent Provider mit jedem kompatiblen Provider-Store verwenden. Der DynamoDB Encryption Client enthält einen MetaStore, bei dem es sich um einen Provider-Store handelt, der Wrapped CMPs zurückgibt.

Der Most Recent Provider ist eine gute Wahl für Anwendungen, die Aufrufe ihrer kryptographischen Quelle minimieren müssen, sowie für Anwendungen, die bestimmte kryptographische Materialien wiederverwenden können, ohne ihre Sicherheitsanforderungen zu verletzen. So können Sie beispielsweise Ihre kryptografischen Materialien mit einem AWS KMS keyin AWS Key Management Service(AWS KMS) schützen, ohne AWS KMS jedes Mal, wenn Sie ein Element ver- oder entschlüsseln, erneut aufrufen zu müssen.

Details hierzu finden Sie unter Most Recent Provider.

Static Materials Provider

Der Static Materials Provider ist auf Tests, Machbarkeitsnachweise und Abwärtskompatibilität ausgelegt. Er generiert keine eindeutigen kryptographischen Materialien für jedes Element. Er gibt

dieselben von Ihnen gelieferten Verschlüsselungs- und Signierschlüssel zurück, die direkt zum Verschlüsseln und Signieren Ihrer Tabellenelemente verwendet werden.

> **Note**
>
> Der Asymmetric Static Provider in der Java-Bibliothek ist kein statischer Anbieter. Er liefert nur alternative Konstruktoren für den Wrapped CMP. Er ist sicher für die Produktion, aber Sie sollten den Wrapped CMP nach Möglichkeit direkt verwenden.

Themen

- Direct KMS Materials Provider
- Wrapped Materials Provider
- Most Recent Provider
- Static Materials Provider

Direct KMS Materials Provider

> **Note**
>
> Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt. AWS Das folgende Thema enthält Informationen zu Versionen 1. x —2. x des DynamoDB Encryption Client für Java und Versionen 1. x —3. x des DynamoDB Encryption Client für Python. Weitere Informationen finden Sie unter AWS Database Encryption SDK für DynamoDB-Versionsunterstützung.

Der Direct KMS Materials Provider (Direct KMS Provider) schützt Ihre Tabellenelemente unter einem AWS KMS keyThat Never Leaves AWS Key Management Service(AWS KMS) unverschlüsselt. Dieser Anbieter kryptographischer Materialien (Cryptographic Materials Provider) gibt einen eindeutigen Verschlüsselungsschlüssel und einen Signierschlüssel für jedes Tabellenelement zurück. Zu diesem Zweck ruft er AWS KMS jedes Mal auf, wenn Sie ein Element ver- oder entschlüsseln.

Wenn Sie DynamoDB-Elemente mit hoher Frequenz und in großem Umfang verarbeiten, überschreiten Sie möglicherweise die Grenzwerte für AWS KMS Anfragen pro Sekunde, was zu Verarbeitungsverzögerungen führen kann. Wenn Sie ein Limit überschreiten müssen, erstellen Sie

einen Fall im Center.AWS Support Sie könnten auch erwägen, einen Anbieter für kryptografisches Material mit begrenzter Schlüsselwiederverwendung zu verwenden, z. B. den neuesten Anbieter.

Um den Direct KMS Provider verwenden zu können, benötigt der Anrufer mindestens einen AWS-Konto und die Berechtigung AWS KMS key, die Operationen GenerateDataKeyund Decrypt auf dem aufzurufen. AWS KMS key Das AWS KMS key muss ein symmetrischer Verschlüsselungsschlüssel sein. Der DynamoDB Encryption Client unterstützt keine asymmetrische Verschlüsselung. Wenn Sie eine globale DynamoDB-Tabelle verwenden, möchten Sie möglicherweise einen Schlüssel für AWS KMS mehrere Regionen angeben. Details hierzu finden Sie unter Verwendung.

> **Note**
>
> Wenn Sie den Direct KMS-Anbieter verwenden, werden die Namen und Werte Ihrer Primärschlüsselattribute im AWS KMS Verschlüsselungskontext und in den AWS CloudTrail Protokollen verwandter Operationen im Klartext angezeigt. AWS KMS Der DynamoDB Encryption Client macht jedoch niemals den Klartext verschlüsselter Attributwerte verfügbar.

Der Direct KMS Provider ist einer von mehreren Cryptographic Materials Providern (CMPs), die der DynamoDB Encryption Client unterstützt. Weitere Information zu den anderen CMPs finden Sie unter Anbieter von kryptografischem Material.

Beispielcode finden Sie unter:

- Java: AwsKmsEncryptedItem
- Python: aws-kms-encrypted-table, aws-kms-encrypted-item

Themen

- Verwendung
- Funktionsweise

Verwendung

Um einen Direct KMS-Anbieter zu erstellen, verwenden Sie den Schlüssel-ID-Parameter, um einen KMS-Schlüssel für die symmetrische Verschlüsselung in Ihrem Konto anzugeben. Der Wert des Schlüssel-ID-Parameters kann die Schlüssel-ID, der Schlüssel-ARN, der Aliasname oder der Alias-ARN von sein AWS KMS key. Einzelheiten zu den Schlüsselkennungen finden Sie unter Schlüsselkennungen im AWS Key Management Service Entwicklerhandbuch.

Der Direct KMS Provider benötigt einen KMS-Schlüssel mit symmetrischer Verschlüsselung. Sie können keinen asymmetrischen KMS-Schlüssel verwenden. Sie können jedoch einen KMS-Schlüssel für mehrere Regionen, einen KMS-Schlüssel mit importiertem Schlüsselmaterial oder einen KMS-Schlüssel in einem benutzerdefinierten Schlüsselspeicher verwenden. Sie benötigen die Berechtigungen kms: GenerateDataKey und kms:Decrypt für den KMS-Schlüssel. Daher müssen Sie einen vom Kunden verwalteten Schlüssel verwenden, keinen verwalteten oder AWS eigenen AWS KMS-Schlüssel.

Der DynamoDB Encryption Client für Python bestimmt die Region für den Aufruf AWS KMS aus der Region im Schlüssel-ID-Parameterwert, falls dieser einen enthält. Andernfalls verwendet er die Region im AWS KMS Client, falls Sie eine angeben, oder die Region, die Sie in der konfigurieren. AWS SDK für Python (Boto3) Informationen zur Regionsauswahl in Python finden Sie unter Konfiguration in der AWS API-Referenz zum SDK for Python (Boto3).

Der DynamoDB Encryption Client für Java bestimmt die Region für Anrufe AWS KMS von der Region im AWS KMS Client, wenn der von Ihnen angegebene Client eine Region enthält. Andernfalls verwendet er die Region, die Sie in der konfigurieren. AWS SDK für Java Informationen zur Regionsauswahl finden Sie unter AWS-Region Auswahl im AWS SDK für Java Developer Guide. AWS SDK für 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 KmsClient kms = KmsClient.builder().region(Region.of(region)).build();
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, keyArn);
```

Python

Im folgenden Beispiel wird der Schlüssel ARN verwendet, um den zu spezifizieren AWS KMS key. Wenn Ihre Schlüssel-ID keine enthält AWS-Region, ruft der DynamoDB Encryption Client die Region aus der konfigurierten Botocore-Sitzung, falls vorhanden, oder aus den Boto-Standardwerten ab.

```
# 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)
```

Wenn Sie globale Amazon DynamoDB-Tabellen verwenden, empfehlen wir Ihnen, Ihre Daten mit einem AWS KMS Schlüssel für mehrere Regionen zu verschlüsseln. Multi-Region Die Schlüssel sind AWS KMS keys unterschiedlich und können synonym verwendet werden AWS-Regionen , da sie dieselbe Schlüssel-ID und dasselbe Schlüsselmaterial haben. Einzelheiten finden Sie im Entwicklerhandbuch unter Verwenden von Schlüsseln für mehrere Regionen.AWS Key Management Service

> **Note**
>
> Wenn Sie die Version 2017.11.29 für globale Tabellen verwenden, müssen Sie Attributaktionen so einrichten, dass die reservierten Replikationsfelder nicht verschlüsselt oder signiert werden. Details hierzu finden Sie unter Probleme mit globalen Tabellen älterer Versionen.

Um einen Schlüssel für mehrere Regionen mit dem DynamoDB Encryption Client zu verwenden, erstellen Sie einen Schlüssel für mehrere Regionen und replizieren Sie ihn in die Regionen, in denen Ihre Anwendung ausgeführt wird. Konfigurieren Sie dann den Direct KMS Provider so, dass er den Schlüssel für mehrere Regionen in der Region verwendet, in der der DynamoDB Encryption Client anruft. AWS KMS

Im folgenden Beispiel wird der DynamoDB Encryption Client so konfiguriert, dass er Daten in der Region USA Ost (Nord-Virginia) (us-east-1) verschlüsselt und in der Region USA West (Oregon) (us-west-2) mit einem Schlüssel für mehrere Regionen entschlüsselt.

Java

In diesem Beispiel ruft der DynamoDB Encryption Client die Region für Anrufe AWS KMS von der Region im AWS KMS Client ab. Der `keyArn` Wert identifiziert einen Schlüssel mit mehreren Regionen in derselben Region.

```
// 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 KmsClient kms = KmsClient.builder().region(Region.of(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 KmsClient kms = KmsClient.builder().region(Region.of(region)).build();
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, usWestKey);
```

Python

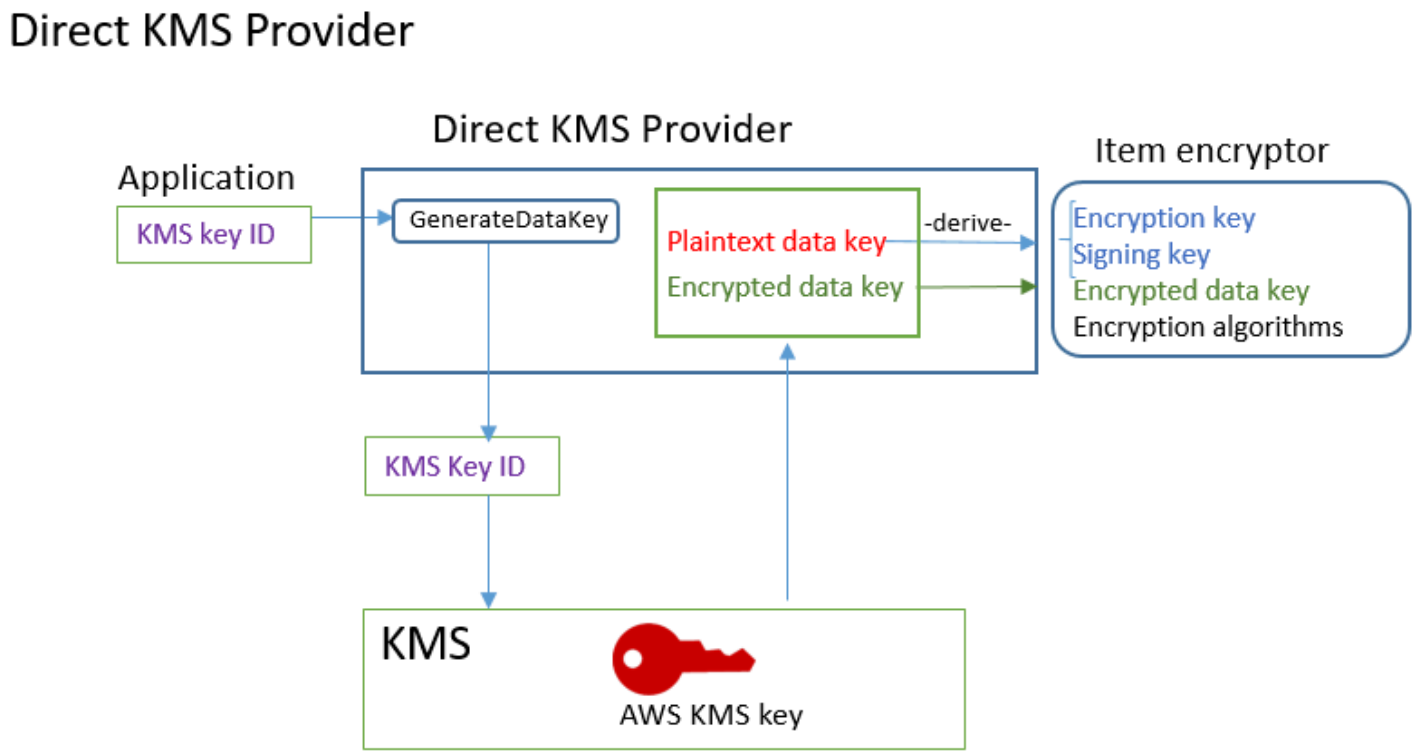
In diesem Beispiel ruft der DynamoDB Encryption Client die Region für Anrufe AWS KMS aus der Region im Schlüssel-ARN ab.

```
# 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)
```

Funktionsweise

Der Direct KMS Provider gibt Verschlüsselungs- und Signaturschlüssel zurück, die durch einen AWS KMS key von Ihnen angegebenen Wert geschützt sind, wie in der folgenden Abbildung dargestellt.



- Um Verschlüsselungsmaterial zu generieren, fordert der Direct KMS Provider AWS KMS auf, mithilfe eines von Ihnen angegebenen Datenschlüssels für jedes Element einen eindeutigen Datenschlüssel AWS KMS key zu generieren. Er leitet Verschlüsselungs- und Signierschlüssel für das Element aus der Klartextkopie des Datenschlüssels ab und gibt dann die Verschlüsselungs- und Signierschlüssel zusammen mit dem verschlüsselten Datenschlüssel zurück, der im Materialbeschreibungsattribut des Elements gespeichert wird.

  Der Elementverschlüssler verwendet die Verschlüsselungs- und Signierschlüssel und entfernt sie so schnell wie möglich aus dem Speicher. Nur die verschlüsselte Kopie des Datenschlüssels, von dem sie abgeleitet wurden, wird im verschlüsselten Element gespeichert.

- Um Entschlüsselungsmaterialien zu generieren, bittet der Direct KMS-Anbieter darum AWS KMS , den verschlüsselten Datenschlüssel zu entschlüsseln. Dann leitet es aus dem Klartextdatenschlüssel Verifizierungs- und Signierschlüssel ab und gibt sie an den Elementverschlüssler zurück.

  Der Elementverschlüssler verifiziert das Element und entschlüsselt bei erfolgreicher Verifikation die verschlüsselten Werte. Anschließend entfernt er die Schlüssel so schnell wie möglich aus dem Speicher.

Verschlüsselungsmaterialien abrufen

Dieser Abschnitt beschreibt detailliert die Ein- und Ausgänge und die Verarbeitung des Direct KMS Providers, wenn er eine Anfrage für Verschlüsselungsmaterialien vom Elementverschlüssler erhält.

Eingabe (von der Anwendung)

- Die Schlüssel-ID eines. AWS KMS key

Eingabe (vom Elementverschlüssler)

- DynamoDB-Verschlüsselungskontext

Ausgabe (an den Elementverschlüssler)

- Verschlüsselungsschlüssel (Klartext)
- Signierschlüssel
- In der tatsächlichen Materialbeschreibung: Diese Werte werden im Materialbeschreibungsattribut gespeichert, das der Client dem Element hinzufügt.
  - amzn-ddb-env-key: Datenschlüssel, verschlüsselt mit Base64-encoded AWS KMS key
  - amzn-ddb-env-alg: Verschlüsselungsalgorithmus, standardmäßig AES/256
  - amzn-ddb-sig-alg: Standardmäßig Signierungsalgorithmus HmacSHA256/256
  - amzn-ddb-wrap-alg: kms

Verarbeitung

1. Der Direct KMS-Anbieter sendet AWS KMS eine Anfrage, um mithilfe der angegebenen Daten einen eindeutigen Datenschlüssel für das Element AWS KMS key zu generieren. Die Operation gibt einen Klartextschlüssel und eine Kopie zurück, die unter dem AWS KMS key verschlüsselt ist. Dies wird als anfängliches Schlüsselmaterial bezeichnet.

   Die Anforderung enthält die folgenden Werte im Klartext im AWS KMS -Verschlüsselungskontext. Diese nicht geheimen Werte sind kryptographisch an das verschlüsselte Objekt gebunden, sodass beim Entschlüsseln der gleiche Verschlüsselungskontext benötigt wird. Sie können diese Werte verwenden, um den Anruf AWS KMS in AWS CloudTrail Protokollen zu identifizieren.

   - amzn-ddb-env-alg — Standardmäßig Verschlüsselungsalgorithmus AES/256

- amzn-ddb-sig-alg — Standardmäßig Signierungsalgorithmus HmacSHA256/256
- (Optional) aws-kms-Tabelle — *table name*
- (Optional) *partition key name* — (Binärwerte sind*partition key value*) Base64-encoded
- (Optional) *sort key name* — *sort key value* (Binärwerte sind Base64-encoded)

Der Direct KMS-Anbieter ruft die Werte für den AWS KMS Verschlüsselungskontext aus dem DynamoDB-Verschlüsselungskontext für das Element ab. Wenn der DynamoDB-Verschlüsselungskontext keinen Wert enthält, z. B. den Tabellennamen, wird dieses Name-Wert-Paar aus dem Verschlüsselungskontext weggelassen. AWS KMS

2. Der Direct KMS Provider leitet aus dem Datenschlüssel einen symmetrischen Verschlüsselungsschlüssel und einen Signierschlüssel ab. Standardmäßig verwendet es Secure Hash Algorithm (SHA) 256 und die HMAC-based RFC5869-Schlüsselableitungsfunktion, um einen symmetrischen 256-Bit-AES-Verschlüsselungsschlüssel und einen 256-Bit-Signaturschlüssel abzuleiten. HMAC-SHA-256
3. Der Direct KMS Provider gibt die Ausgabe an den Elementverschlüssler zurück.
4. Der Elementverschlüssler verwendet den Verschlüsselungsschlüssel, um die angegebenen Attribute zu verschlüsseln, und den Signierschlüssel, um sie mit den in der tatsächlichen Materialbeschreibung angegebenen Algorithmen zu signieren. Er entfernt die Klartextschlüssel so schnell wie möglich aus dem Speicher.

Entschlüsselungsmaterialien abrufen

Dieser Abschnitt beschreibt detailliert die Ein- und Ausgänge und die Verarbeitung des Direct KMS Providers, wenn er eine Anfrage für Entschlüsselungsmaterialien vom Elementverschlüssler erhält.

Eingabe (von der Anwendung)

- Die Schlüssel-ID eines. AWS KMS key

  Der Wert der Schlüssel-ID kann die Schlüssel-ID, der Schlüssel-ARN, der Aliasname oder der Alias-ARN von sein AWS KMS key. Alle Werte, die nicht in der Schlüssel-ID enthalten sind, z. B. die Region, müssen im AWS benannten Profil verfügbar sein. Der Schlüssel ARN liefert alle Werte, die AWS KMS benötigt werden.

Eingabe (vom Elementverschlüssler)

- Eine Kopie des DynamoDB-Verschlüsselungskontextes, der den Inhalt des Materialbeschreibungsattributs enthält.

Ausgabe (an den Elementverschlüssler)

- Verschlüsselungsschlüssel (Klartext)
- Signierschlüssel

Verarbeitung

1. Der Direct KMS-Anbieter ruft den verschlüsselten Datenschlüssel aus dem Materialbeschreibungsattribut im verschlüsselten Element ab.
2. Er fordert Sie AWS KMS auf, den angegebenen Schlüssel AWS KMS key zum Entschlüsseln des verschlüsselten Datenschlüssels zu verwenden. Die Operation gibt einen Klartextschlüssel zurück.

   Diese Anforderung muss denselben AWS KMS -Verschlüsselungskontext verwenden, in dem der Datenschlüssel generiert und verschlüsselt wurde.

   - aws-kms-Tabelle — *`table name`*
   - *`partition key name`*— *`partition key value`* (Binärwerte sind) Base64-encoded
   - (Optional) *`sort key name`* — *`sort key value`* (Binärwerte sind Base64-encoded)
   - amzn-ddb-env-alg — Standardmäßig Verschlüsselungsalgorithmus AES/256
   - amzn-ddb-sig-alg — Standardmäßig Signierungsalgorithmus HmacSHA256/256
3. Der Direct KMS Provider verwendet Secure Hash Algorithm (SHA) 256 und die HMAC-based RFC5869-Schlüsselableitungsfunktion, um einen symmetrischen 256-Bit-AES-Verschlüsselungsschlüssel und einen 256-Bit-Signaturschlüssel aus dem Datenschlüssel abzuleiten. HMAC-SHA-256
4. Der Direct KMS Provider gibt die Ausgabe an den Elementverschlüssler zurück.
5. Der Elementverschlüssler verwendet den Signierschlüssel, um das Element zu verifizieren. Wenn er erfolgreich ist, verwendet er den symmetrischen Verschlüsselungsschlüssel, um die verschlüsselten Attributwerte zu entschlüsseln. Diese Operationen verwenden die in der tatsächlichen Materialbeschreibung angegebenen Verschlüsselungs- und Signieralgorithmen. Der Elementverschlüssler entfernt die Klartextschlüssel so schnell wie möglich aus dem Speicher.

Wrapped Materials Provider

> **Note**
>
> Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt. AWS Das folgende Thema enthält Informationen zu Versionen 1. x —2. x des DynamoDB Encryption Client für Java und Versionen 1. x —3. x des DynamoDB Encryption Client für Python. Weitere Informationen finden Sie unter AWS Database Encryption SDK für DynamoDB-Versionsunterstützung.

Mit dem Wrapped Materials Provider (Wrapped CMP) können Sie Schlüssel aus beliebigen Quellen mit dem DynamoDB Encryption Client verpacken und signieren. Das Wrapped CMP ist von keinem Dienst abhängig. AWS Sie müssen jedoch Ihre Wrapping- und Signierschlüssel außerhalb des Clients generieren und verwalten, einschließlich der Bereitstellung der richtigen Schlüssel zur Verifizierung und Entschlüsselung des Elements.

Der Wrapped CMP generiert einen eindeutigen Verschlüsselungsschlüssel für jedes Element. Er verpackt den Verschlüsselungsschlüssel des Elements mit dem von Ihnen bereitgestellten Wrapping-Schlüssel und speichert den verpackten Elementverschlüsselungsschlüssel im Materialbeschreibungsattribut des Elements. Da Sie die Wrapping- und Signierschlüssel bereitstellen, bestimmen Sie, wie die Wrapping- und Signierschlüssel erzeugt werden und ob sie für jedes Element eindeutig sind oder wiederverwendet werden.

Der Wrapped CMP ist eine sichere Implementierung und eine gute Wahl für Anwendungen, die kryptographische Materialien verwalten können.

Der Wrapped CMP ist einer von mehreren Anbietern von kryptografischen Materialien (CMPs), die der DynamoDB Encryption Client unterstützt. Weitere Information zu den anderen CMPs finden Sie unter Anbieter von kryptografischem Material.

Beispielcode finden Sie unter:

- Java: AsymmetricEncryptedItem
- Python: wrapped-rsa-encrypted-table, wrapped-symmetric-encrypted-table

Themen

- Verwendung

- Funktionsweise

Verwendung

Um einen Wrapped CMP zu erstellen, geben Sie einen Wrapping-Schlüssel (beim Verschlüsseln erforderlich), einen Unwrapping-Schlüssel (beim Entschlüsseln erforderlich) und einen Signierschlüssel an. Sie müssen beim Ver- und Entschlüsseln von Elementen Schlüssel bereitstellen.

Die Wrapping-, Unwrapping- und Signierschlüssel können symmetrische Schlüssel oder asymmetrische Schlüsselpaare sein.

Java

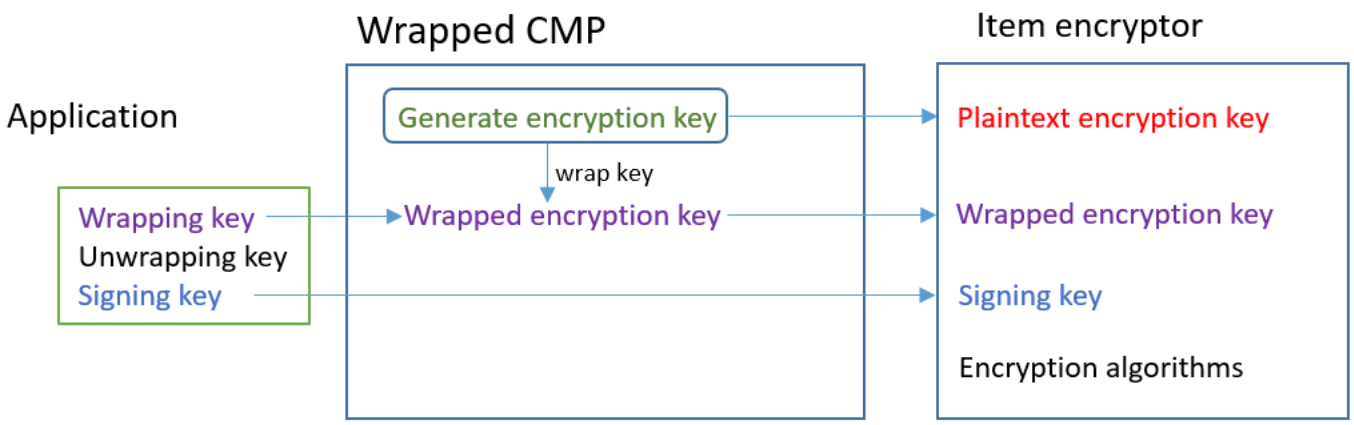
```
// This example uses asymmetric wrapping and signing key pairs
final KeyPair wrappingKeys = ...
final KeyPair signingKeys = ...

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
)
```

Funktionsweise

Der Wrapped CMP generiert einen neuen Verschlüsselungsschlüssel für jedes Element. Es verwendet die von Ihnen bereitgestellten Wrapping-, Unwrapping- und Signierschlüssel, wie in der folgenden Abbildung gezeigt.



Verschlüsselungsmaterialien abrufen

Dieser Abschnitt beschreibt detailliert die Ein- und Ausgänge und die Verarbeitung des Wrapped Materials Providers (Wrapped CMP), wenn er eine Anfrage für Verschlüsselungsmaterialien erhält.

Eingabe (von der Anwendung)

- Wrapping-Schlüssel: Ein symmetrischer Advanced Encryption Standard (AES)-Schlüssel oder ein öffentlicher RSA-Schlüssel. Erforderlich, wenn Attributwerte verschlüsselt sind. Andernfalls ist er optional und wird ignoriert.
- Unwrapping-Schlüssel: Optional und wird ignoriert.
- Signierschlüssel

Eingabe (vom Elementverschlüssler)

- DynamoDB-Verschlüsselungskontext

Ausgabe (an den Elementverschlüssler):

- Klartext-Element-Verschlüsselungsschlüssel
- Signierschlüssel (unverändert)
- Tatsächliche Materialbeschreibung: Diese Werte werden im Materialbeschreibungsattribut gespeichert, das der Client dem Element hinzufügt.
  - `amzn-ddb-env-key`: Base64-encoded Verschlüsselungsschlüssel für verpackte Artikel
  - `amzn-ddb-env-alg`: Verschlüsselungsalgorithmus, der zur Verschlüsselung des Elements verwendet wird. Der Standardwert ist AES-256-CBC.

- `amzn-ddb-wrap-alg`: Der Wrapping-Algorithmus, den der Wrapped CMP verwendet hat, um den Elementverschlüsselungsschlüssel zu verpacken. Wenn es sich bei dem Wrapping-Schlüssel um einen AES-Schlüssel handelt, wird der Schlüssel mit nicht aufgefülltem `AES-Keywrap` verpackt, wie in RFC 3394 definiert. Handelt es sich bei dem Wrapping-Schlüssel um einen RSA-Schlüssel, wird dieser mit RSA OAEP mit MGF1-Auffüllung verschlüsselt.

Verarbeitung

Wenn Sie ein Element verschlüsseln, übergeben Sie einen Wrapping-Schlüssel und einen Signierschlüssel. Ein Unwrapping-Schlüssel ist optional und wird ignoriert.

1. Der Wrapped CMP generiert einen symmetrischen Elementverschlüsselungsschlüssel für das Tabellenelement.
2. Er verwendet den Wrapping-Schlüssel, den Sie angeben, um den Elementverschlüsselungsschlüssel zu verpacken. Anschließend entfernt er ihn so schnell wie möglich aus dem Speicher.
3. Es gibt den Klartextelementverschlüsselungsschlüssel zurück, den von Ihnen angegebenen Signierschlüssel und eine tatsächliche Materialbeschreibung, die den Verschlüsselungsschlüssel des verpackten Elements und die Verschlüsselungs- und Wrapping-Algorithmen enthält.
4. Der Elementverschlüssler verwendet den Klartext-Verschlüsselungsschlüssel, um das Element zu verschlüsseln. Es verwendet den von Ihnen bereitgestellten Signierschlüssel, um das Element zu signieren. Anschließend entfernt er die Klartextschlüssel so schnell wie möglich aus dem Speicher. Es kopiert die Felder in der tatsächlichen Materialbeschreibung, einschließlich des verpackten Verschlüsselungsschlüssels (`amzn-ddb-env-key`), in das Materialbeschreibungsattribut des Elements.

Entschlüsselungsmaterialien abrufen

Dieser Abschnitt beschreibt detailliert die Ein- und Ausgänge und die Verarbeitung des Wrapped Materials Providers (Wrapped CMP), wenn er eine Anfrage für Entschlüsselungsmaterialien erhält.

Eingabe (von der Anwendung)

- Wrapping-Schlüssel: Optional und wird ignoriert.
- Unwrapping-Schlüssel: Derselbe symmetrische Advanced Encryption Standard (AES)-Schlüssel oder private RSA-Schlüssel, der dem zum Verschlüsseln verwendeten öffentlichen RSA-Schlüssel

entspricht. Erforderlich, wenn Attributwerte verschlüsselt sind. Andernfalls ist er optional und wird ignoriert.

- Signierschlüssel

Eingabe (vom Elementverschlüssler)

- Eine Kopie des DynamoDB-Verschlüsselungskontextes, der den Inhalt des Materialbeschreibungsattributs enthält.

Ausgabe (an den Elementverschlüssler)

- Klartext-Element-Verschlüsselungsschlüssel
- Signierschlüssel (unverändert)

Verarbeitung

Wenn Sie ein Element entschlüsseln, übergeben Sie einen Unwrapping-Schlüssel und einen Signierschlüssel. Ein Wrapping-Schlüssel ist optional und wird ignoriert.

1. Der Wrapped CMP erhält den Verschlüsselungsschlüssel des verpackten Elements aus dem Materialbeschreibungsattribut des Elements.
2. Er verwendet den Unwrapping-Schlüssel und den Algorithmus, um den Verschlüsselungsschlüssel des Elements zu entpacken.
3. Es gibt den Klartextelementverschlüsselungsschlüssel, den Signierschlüssel sowie Verschlüsselungs- und Signieralgorithmen an den Elementverschlüsseler zurück.
4. Der Elementverschlüssler verwendet den Signierschlüssel, um das Element zu verifizieren. Wenn dies erfolgreich ist, verwendet er den Verschlüsselungsschlüssel des Elements, um das Element zu entschlüsseln. Anschließend entfernt er die Klartextschlüssel so schnell wie möglich aus dem Speicher.

Most Recent Provider

> **Note**
>
> Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt. AWS Das folgende Thema enthält Informationen zu Versionen 1. x —2. x des

DynamoDB Encryption Client für Java und Versionen 1. x —3. x des DynamoDB Encryption Client für Python. Weitere Informationen finden Sie unter AWS Database Encryption SDK für DynamoDB-Versionsunterstützung.

Der Most Recent Provider ist ein Anbieter kryptographischer Materialien (Cryptographic Materials Provider (CMP)), der auf die Arbeit mit einem Provider-Store ausgelegt ist. Es erhält CMPs aus dem Provider-Store und erhält die kryptographischen Materialien, die er zurückgibt, von den CMPs. Typischerweise wird jeder CMP verwendet, um mehrere Anfragen nach kryptographischem Material zu befriedigen. Sie können jedoch mit den Funktionen des Provider-Stores steuern, inwieweit Materialien wiederverwendet werden, wie oft der CMP gewechselt wird und sogar den Typ des CMP ändern, ohne den Most Recent Provider zu ändern.

Note

Der Code, der dem `MostRecentProvider` Symbol für den neuesten Anbieter zugeordnet ist, speichert möglicherweise kryptografisches Material für die gesamte Lebensdauer des Prozesses im Speicher. Es könnte einem Anrufer ermöglichen, Schlüssel zu verwenden, zu deren Verwendung er nicht mehr berechtigt ist.
Das `MostRecentProvider` Symbol ist in älteren unterstützten Versionen des DynamoDB Encryption Client veraltet und wurde aus Version 2.0.0 entfernt. Es wird durch das Symbol ersetzt. `CachingMostRecentProvider` Details hierzu finden Sie unter Aktualisierungen für den neuesten Anbieter.

Der Most Recent Provider ist eine gute Wahl für Anwendungen, die Aufrufe des Provider-Stores und seiner kryptographischen Quelle minimieren müssen, sowie für Anwendungen, die bestimmte kryptographische Materialien wiederverwenden können, ohne ihre Sicherheitsanforderungen zu verletzen. So können Sie beispielsweise Ihr kryptografisches Material mit einem AWS KMS keyin AWS Key Management Service(AWS KMS) schützen, ohne AWS KMS jedes Mal, wenn Sie ein Objekt ver- oder entschlüsseln, erneut aufrufen zu müssen.

Der von Ihnen gewählte Provider-Store bestimmt den Typ der CMPs, die der Most Recent Provider verwendet, und wie oft er einen neuen CMP erhält. Sie können jeden kompatiblen Provider-Store für den Most Recent Provider verwenden, einschließlich benutzerdefinierter Provider-Stores, die Sie entwerfen.

Der DynamoDB Encryption Client enthält einen MetaStore, der Wrapped Materials Providers (Wrapped CMPs) erstellt und zurückgibt. Das MetaStore speichert mehrere Versionen der Wrapped CMPs, die es generiert, in einer internen DynamoDB-Tabelle und schützt sie mit clientseitiger Verschlüsselung durch eine interne Instanz des DynamoDB Encryption Client.

Sie können den MetaStore so konfigurieren, dass er jede Art von internem CMP verwendet, um die Materialien in der Tabelle zu schützen, einschließlich eines direkten KMS-Anbieters, der von Ihnen geschütztes kryptografisches Material generiert AWS KMS key, eines Wrapped CMP, das von Ihnen bereitgestellte Wrapping- und Signierschlüssel verwendet, oder eines kompatiblen benutzerdefinierten CMP, das Sie entwerfen.

Beispielcode finden Sie unter:

- Java: MostRecentEncryptedItem
- Python: most_recent_provider_encrypted_table

Themen

- Verwendung
- Funktionsweise
- Aktualisierungen für den neuesten Anbieter

Verwendung

Um einen Most Recent Provider zu erstellen, müssen Sie einen Provider-Store erstellen und konfigurieren und dann einen Most Recent Provider erstellen, der den Provider-Store verwendet.

Die folgenden Beispiele zeigen, wie Sie einen Aktuellsten Anbieter erstellen, der einen verwendet MetaStore und die Versionen in seiner internen DynamoDB-Tabelle mit kryptografischem Material von einem Direct KMS-Anbieter schützt. In diesen Beispielen wird das Symbol verwendet. `CachingMostRecentProvider`

Jeder aktuelle Anbieter hat einen Namen, der seine CMPs in der MetaStore Tabelle identifiziert, eine Einstellung für die Gültigkeitsdauer (TTL) und eine Einstellung für die Cachegröße, die bestimmt, wie viele Einträge der Cache aufnehmen kann. In diesen Beispielen wird die Cachegröße auf 1000 Einträge und eine TTL von 60 Sekunden festgelegt.

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 DynamoDbClient ddb =
 DynamoDbClient.builder().region(Region.of(region)).build();

// Create an internal Direct KMS Provider for the MetaStore
final KmsClient kms = KmsClient.builder().region(Region.of(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
)
```

Funktionsweise

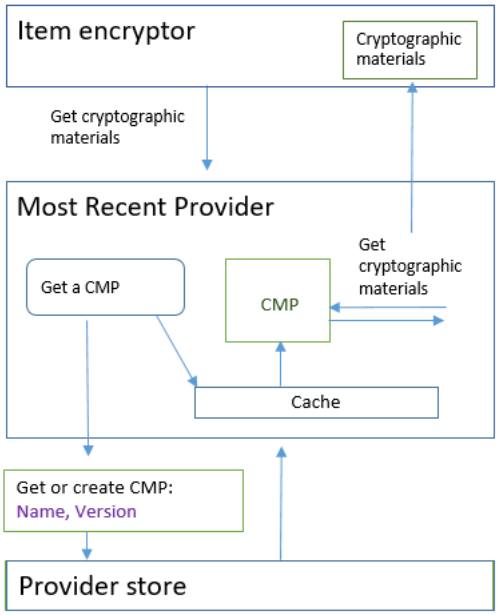
Der Most Recent Provider erhält CMPs von einem Provider-Store. Dann verwendet er den CMP, um die kryptographischen Materialien zu generieren, die er an den Elementverschlüssler zurückgibt.

Informationen über den Most Recent Provider

Der Most Recent Provider erhält einen Anbieter kryptographischer Materialien (Cryptographic Materials Provider (CMP)) aus einem Provider-Store. Dann verwendet er den CMP, um die kryptographischen Materialien zu generieren, die er zurückgibt. Jeder Most Recent Provider ist einem Provider-Store zugeordnet, aber ein Provider-Store kann CMPs an mehrere Provider über mehrere Hosts liefern.

Der Most Recent Provider kann mit jedem kompatiblen CMP aus einem beliebigen Provider-Store arbeiten. Es fordert Verschlüsselungs- oder Entschlüsselungsmaterial vom CMP an und gibt die Ausgabe an den Elementverschlüsseler zurück. Er führt keine kryptografischen Operationen durch.

Um einen CMP von seinem Provider-Store anzufordern, gibt der Most Recent Provider seinen Materialnamen und die Version eines bestehenden CMP an, den er verwenden möchte. Bei Verschlüsselungsmaterialien fordert der Most Recent Provider immer die maximale („neueste“) Version an. Bei Entschlüsselungsmaterialien wird die Version des CMP angefordert, die für die Erstellung der Verschlüsselungsmaterialien verwendet wurde, wie in der folgenden Abbildung dargestellt.



Der Most Recent Provider speichert Versionen der CMPs, die der Provider-Store zurückgibt, in einem lokalen Least Recently Used (LRU)-Cache im Speicher. Der Cache ermöglicht es dem Most Recent Provider, die benötigten CMPs zu erhalten, ohne für jedes Element den Provider-Store aufzurufen. Sie können den Cache bei Bedarf leeren.

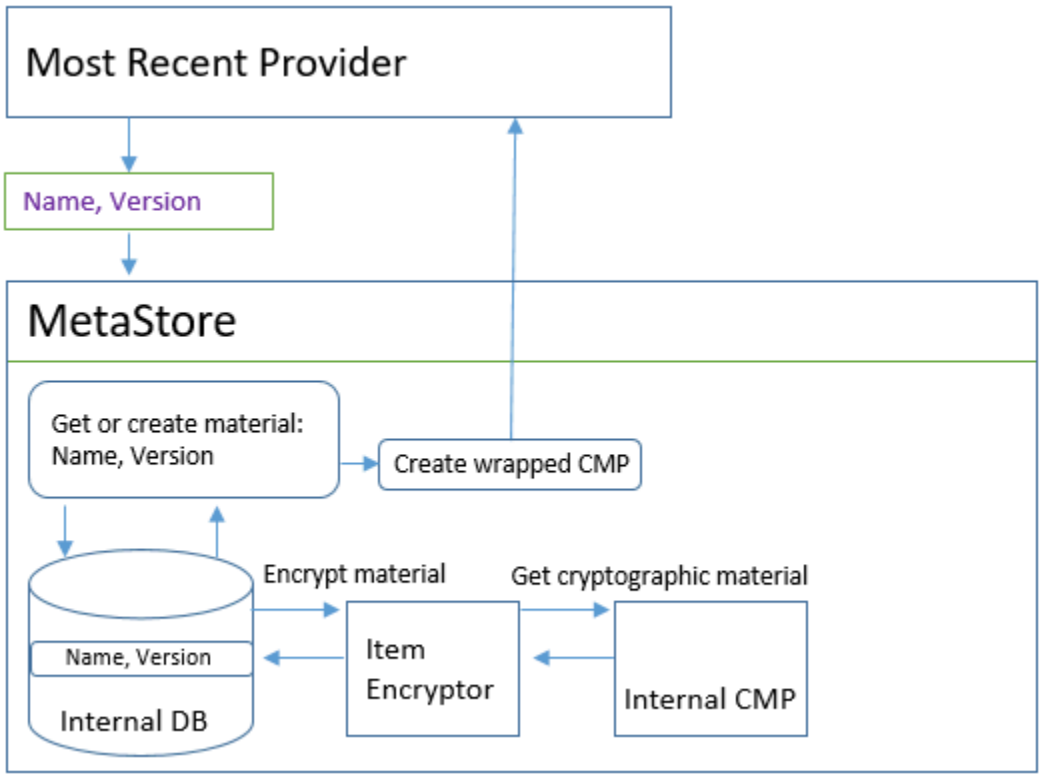
Der neueste Anbieter verwendet einen konfigurierbaren Wert für die Gültigkeitsdauer, den Sie an die Eigenschaften Ihrer Anwendung anpassen können.

Über den MetaStore

Sie können einen Most Recent Provider mit einem beliebigen Provider-Store verwenden, einschließlich eines kompatiblen benutzerdefinierten Provider-Stores. Der DynamoDB Encryption Client umfasst eine MetaStore, eine sichere Implementierung, die Sie konfigurieren und anpassen können.

A MetaStoreist ein Provider-Store, der Wrapped CMPs erstellt und zurückgibt, die mit dem Wrapped Key, dem Unwrapping Key und dem Signaturschlüssel konfiguriert sind, die Wrapped CMPs benötigen. A MetaStore ist eine sichere Option für den neuesten Anbieter, da Wrapped CMPs immer eindeutige Elementverschlüsselungsschlüssel für jedes Element generieren. Nur der Wrapping-Schlüssel, der den Elementverschlüsselungsschlüssel und die Signierschlüssel schützt, wird wiederverwendet.

Das folgende Diagramm zeigt die Komponenten von MetaStore und wie es mit dem neuesten Anbieter interagiert.



Der MetaStore generiert die Wrapped CMPs und speichert sie dann (in verschlüsselter Form) in einer internen DynamoDB-Tabelle. Der Partitionsschlüssel ist der Name des aktuellsten Provider-Materials, der Sortierschlüssel die Versionsnummer. Die Materialien in der Tabelle sind durch einen internen DynamoDB Encryption Client geschützt, einschließlich eines Elementverschlüsselers und eines internen Anbieters für kryptografische Materialien (CMP).

Sie können jede Art von internem CMP in Ihrem verwenden MetaStore, einschließlich eines direkten KMS-Anbieters, eines verpackten CMP mit von Ihnen bereitgestellten kryptografischen Materialien oder eines kompatiblen benutzerdefinierten CMP. Wenn es sich bei Ihrem internen CMP um einen Direct KMS-Anbieter MetaStore handelt, sind Ihre wiederverwendbaren Wrapping- und Signaturschlüssel mit einem in () geschützt. AWS KMS keyAWS Key Management ServiceAWS KMS

Der MetaStore ruft AWS KMS jedes Mal auf, wenn er seiner internen Tabelle eine neue CMP-Version hinzufügt oder eine CMP-Version aus seiner internen Tabelle abruft.

Es wird ein Wert für die Gültigkeitsdauer festgelegt

Sie können für jeden neuesten Anbieter, den Sie erstellen, einen Wert für die Gültigkeitsdauer (TTL) festlegen. Verwenden Sie im Allgemeinen den niedrigsten TTL-Wert, der für Ihre Anwendung praktikabel ist.

Die Verwendung des TTL-Werts wird im `CachingMostRecentProvider` Symbol für den neuesten Anbieter geändert.

> **Note**
>
> Das `MostRecentProvider` Symbol für den neuesten Anbieter ist in älteren unterstützten Versionen des DynamoDB Encryption Client veraltet und wurde aus Version 2.0.0 entfernt. Es wird durch das Symbol ersetzt. `CachingMostRecentProvider` Wir empfehlen Ihnen, Ihren Code so schnell wie möglich zu aktualisieren. Details hierzu finden Sie unter Aktualisierungen für den neuesten Anbieter.

`CachingMostRecentProvider`

Der `CachingMostRecentProvider` verwendet den TTL-Wert auf zwei verschiedene Arten.

- Die TTL bestimmt, wie oft der neueste Anbieter im Anbieterspeicher nach einer neuen Version der CMP sucht. Wenn eine neue Version verfügbar ist, ersetzt der neueste Anbieter seinen CMP und aktualisiert sein kryptografisches Material. Andernfalls verwendet er weiterhin seine aktuellen CMP- und kryptografischen Materialien.
- Die TTL bestimmt, wie lange CMPs im Cache verwendet werden können. Bevor ein zwischengespeichertes CMP für die Verschlüsselung verwendet wird, bewertet der aktuelle Anbieter die Zeit im Cache. Wenn die CMP-Cachezeit die TTL überschreitet, wird die CMP aus dem Cache entfernt und der neueste Anbieter erhält eine neue CMP der neuesten Version aus seinem Provider-Speicher.

`MostRecentProvider`

In der bestimmt die TTL`MostRecentProvider`, wie oft der neueste Anbieter im Anbieterspeicher nach einer neuen Version des CMP sucht. Wenn eine neue Version verfügbar ist, ersetzt der

neueste Anbieter seinen CMP und aktualisiert sein kryptografisches Material. Andernfalls verwendet er weiterhin seine aktuellen CMP- und kryptografischen Materialien.

Die TTL bestimmt nicht, wie oft eine neue CMP-Version erstellt wird. Sie erstellen neue CMP-Versionen, indem Sie die kryptografischen Materialien rotieren.

Ein idealer TTL-Wert hängt von der Anwendung und ihren Latenz- und Verfügbarkeitszielen ab. Eine niedrigere TTL verbessert Ihr Sicherheitsprofil, da sie die Zeit reduziert, in der kryptografisches Material im Speicher gespeichert wird. Außerdem aktualisiert eine niedrigere TTL wichtige Informationen häufiger. Wenn es sich bei Ihrem internen CMP beispielsweise um einen Direct KMS-Anbieter handelt, überprüft er häufiger, ob der Anrufer weiterhin berechtigt ist, einen zu verwenden. AWS KMS key

Wenn die TTL jedoch zu kurz ist, können die häufigen Anrufe beim Provider-Store Ihre Kosten in die Höhe treiben und dazu führen, dass Ihr Provider-Store Anfragen von Ihrer Anwendung und anderen Anwendungen, die Ihr Dienstkonto gemeinsam nutzen, drosselt. Sie könnten auch davon profitieren, die TTL mit der Geschwindigkeit zu koordinieren, mit der Sie kryptografisches Material wechseln.

Variieren Sie beim Testen die TTL und die Cachegröße je nach Arbeitslast, bis Sie eine Konfiguration gefunden haben, die für Ihre Anwendung und Ihre Sicherheits- und Leistungsstandards geeignet ist.

Rotieren von kryptografischen Materialien

Wenn ein neuester Anbieter Verschlüsselungsmaterial benötigt, verwendet er immer die neueste Version seiner CMP, die ihm bekannt ist. Die Häufigkeit, mit der nach einer neueren Version gesucht wird, wird durch den Time-to-Live-Wert (TTL) bestimmt, den Sie bei der Konfiguration des neuesten Anbieters festgelegt haben.

Wenn die TTL abläuft, sucht der neueste Anbieter im Anbieterspeicher nach einer neueren Version der CMP. Wenn eine verfügbar ist, wird sie vom neuesten Anbieter abgerufen und die CMP in seinem Cache ersetzt. Er verwendet dieses CMP und seine kryptografischen Materialien, bis es feststellt, dass es im Provider-Store eine neuere Version gibt.

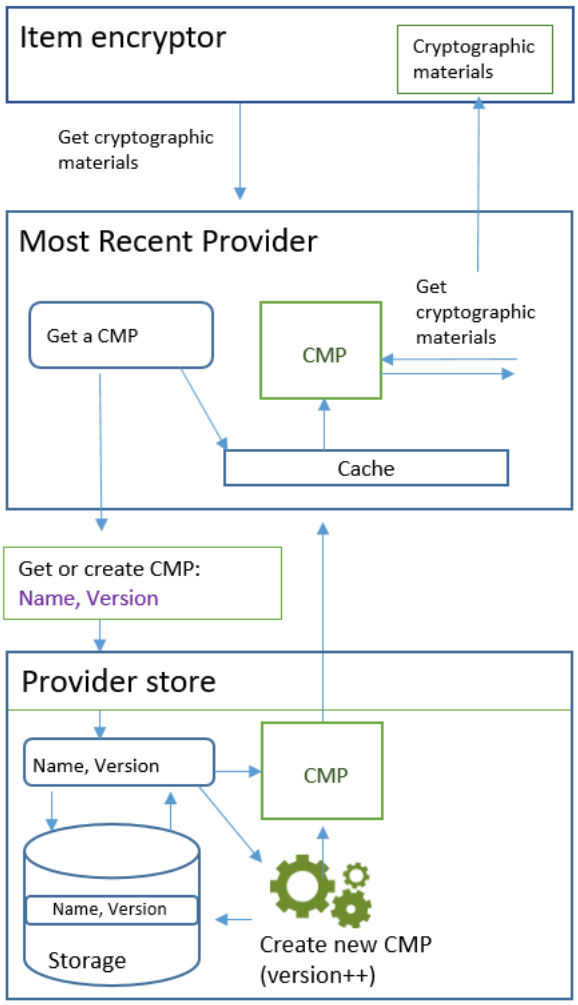
Um den Provider-Store anzuweisen, eine neue Version eines CMP für einen Most Recent Provider zu erstellen, rufen Sie die Operation „Create New Provider“ (Neuen Provider erstellen) des Provider-Stores mit dem Materialnamen des Most Recent Providers auf. Der Provider-Store erstellt einen neuen CMP und speichert eine verschlüsselte Kopie in seinem internen Speicher mit einer höheren Versionsnummer. (Es gibt auch einen CMP zurück, aber Sie können ihn verwerfen.) Wenn der neueste Anbieter das nächste Mal den Provider-Speicher nach der maximalen Versionsnummer

seiner CMPs abfragt, erhält er die neue höhere Versionsnummer und verwendet sie in nachfolgenden Anfragen an den Store, um festzustellen, ob eine neue Version der CMP erstellt wurde.

Sie können Ihre „Create New Provider“-Aufrufe (Neuen Provider erstellen) abhängig von der Zeit, der Anzahl der verarbeiteten Elemente oder Attribute oder einer anderen für Ihre Anwendung sinnvollen Kennzahl planen.

Verschlüsselungsmaterialien abrufen

Der Most Recent Provider verwendet den folgenden Prozess, wie in dieser Abbildung gezeigt, um die Verschlüsselungsmaterialien zu erhalten, die er an den Elementverschlüssler zurückgibt. Die Ausgabe hängt vom Typ des CMP ab, den der Provider-Store zurückgibt. Der neueste Anbieter kann jeden kompatiblen Anbieterspeicher verwenden, einschließlich des Speichers MetaStore , der im DynamoDB Encryption Client enthalten ist.



Wenn Sie mithilfe des `CachingMostRecentProviderSymbols` einen aktuellen Anbieter erstellen, geben Sie einen Provider-Speicher, einen Namen für den neuesten Anbieter und einen Time-to-

Live-Wert (TTL) an. Sie können optional auch eine Cachegröße angeben, die die maximale Anzahl kryptografischer Materialien bestimmt, die im Cache vorhanden sein können.

Wenn der Elementverschlüssler den Most Recent Provider nach Verschlüsselungsmaterialien fragt, sucht der Most Recent Provider zunächst in seinem Cache nach der neuesten Version seines CMP.

- Wenn er die neueste Version von CMP in seinem Cache findet und der CMP den TTL-Wert nicht überschritten hat, verwendet der neueste Anbieter den CMP, um Verschlüsselungsmaterial zu generieren. Anschließend gibt er die Verschlüsselungsmaterialien an den Elementverschlüssler zurück. Für diese Operation muss der Provider-Store nicht aufgerufen werden.
- Wenn sich die neueste Version der CMP nicht in seinem Cache befindet oder wenn sie sich im Cache befindet, aber ihren TTL-Wert überschritten hat, fordert der neueste Anbieter eine CMP aus seinem Provider-Speicher an. Die Anfrage enthält den Materialnamen des Most Recent Providers und die maximale Versionsnummer, die ihm bekannt ist.

  1. Der Provider-Store gibt einen CMP aus seinem persistenten Speicher zurück. Wenn es sich bei dem Provider-Speicher um einen handelt MetaStore, ruft er eine verschlüsselte Wrapped CMP aus seiner internen DynamoDB-Tabelle ab, indem er den Materialnamen des neuesten Anbieters als Partitionsschlüssel und die Versionsnummer als Sortierschlüssel verwendet. Der MetaStore verwendet seinen internen Elementverschlüsseler und sein internes CMP, um das Wrapped CMP zu entschlüsseln. Dann gibt er den Klartext-CMP an den Most Recent Provider zurück. Wenn der interne CMP ein Direct KMS Provider ist, beinhaltet dieser Schritt einen Aufruf von AWS Key Management Service (AWS KMS).
  2. Der CMP fügt das `amzn-ddb-meta-id`-Feld der aktuellen Materialbeschreibung hinzu. Sein Wert sind der Materialname und die Version des CMP in seiner internen Tabelle. Der Provider-Store gibt den CMP an den Most Recent Provider zurück.
  3. Der Most Recent Provider speichert den CMP im Speicher zwischen.
  4. Der Most Recent Provider verwendet den CMP, um Verschlüsselungsmaterialien zu generieren. Anschließend gibt er die Verschlüsselungsmaterialien an den Elementverschlüssler zurück.

Entschlüsselungsmaterialien abrufen

Wenn der Elementverschlüssler den Most Recent Provider nach Entschlüsselungsmaterialien abfragt, verwendet der Most Recent Provider den folgenden Prozess, um diese abzurufen und zurückzugeben.

1. Der Most Recent Provider fragt den Provider-Store nach der Versionsnummer des kryptographischen Materials, das zur Verschlüsselung des Elements verwendet wurde. Er übergibt die aktuelle Materialbeschreibung aus dem Materialbeschreibungsattribut des Elements.
2. Der Provider Store ruft die verschlüsselnde CMP-Versionsnummer aus dem Feld `amzn-ddb-meta-id` in der aktuellen Materialbeschreibung ab und gibt sie an den Most Recent Provider zurück.
3. Der Most Recent Provider durchsucht seinen Cache nach der Version des CMP, mit der das Element verschlüsselt und signiert wurde.

- Wenn er feststellt, dass sich die passende Version des CMP in seinem Cache befindet und der CMP den Time-to-Live-Wert (TTL) nicht überschritten hat, verwendet der neueste Anbieter den CMP, um Entschlüsselungsmaterialien zu generieren. Anschließend gibt er die Entschlüsselungsmaterialien an den Elementverschlüssler zurück. Für diese Operation muss der Provider-Store nicht aufgerufen werden, und auch kein anderer CMP.
- Wenn sich die passende Version der CMP nicht im Cache befindet oder wenn die zwischengespeicherte Version ihren TTL-Wert überschritten AWS KMS key hat, fordert der neueste Anbieter eine CMP aus seinem Anbieterspeicher an. Er sendet seinen Materialnamen und die Versionsnummer des verschlüsselnden CMP in der Anfrage.

  1. Der Provider-Store durchsucht seinen persistenten Speicher nach dem CMP, indem er den Namen des Most Recent Providers als Partitionsschlüssel und die Versionsnummer als Sortierschlüssel verwendet.

     - Wenn sich der Name und die Versionsnummer nicht in seinem persistenten Speicher befinden, wirft der Provider-Store eine Ausnahme auf. Wenn der Provider-Store zur Generierung des CMP verwendet wurde, sollte der CMP in seinem persistenten Speicher abgelegt sein, es sei denn, er wurde absichtlich gelöscht.
     - Wenn sich der CMP mit dem übereinstimmenden Namen und der Versionsnummer im persistenten Speicher des Provider-Stores befindet, gibt der Provider-Store den angegebenen CMP an den Most Recent Provider zurück.

       Wenn es sich bei dem Provider-Speicher um einen handelt MetaStore, ruft er die verschlüsselte CMP aus seiner DynamoDB-Tabelle ab. Dann verwendet er kryptografische Materialien aus seinem internen CMP, um den verschlüsselten CMP zu entschlüsseln, bevor es den CMP an den Most Recent Provider zurückgibt. Wenn der interne CMP ein Direct KMS Provider ist, beinhaltet dieser Schritt einen Aufruf von AWS Key Management Service (AWS KMS).

2. Der Most Recent Provider speichert den CMP im Speicher zwischen.
3. Der Most Recent Provider verwendet den CMP, um Entschlüsselungsmaterialien zu generieren. Anschließend gibt er die Entschlüsselungsmaterialien an den Elementverschlüssler zurück.

Aktualisierungen für den neuesten Anbieter

Das Symbol für den neuesten Anbieter wurde von `MostRecentProvider` zu geändert`CachingMostRecentProvider`.

> **Note**
>
> Das `MostRecentProvider` Symbol, das den neuesten Anbieter darstellt, ist in Version 1.15 des DynamoDB Encryption Client für Java und Version 1.3 des DynamoDB Encryption Client für Python veraltet und wurde aus den Versionen 2.0.0 des DynamoDB Encryption Client in beiden Sprachimplementierungen entfernt. Verwenden Sie stattdessen die. `CachingMostRecentProvider`

Der `CachingMostRecentProvider` implementiert die folgenden Änderungen:

- Die entfernt in `CachingMostRecentProvider` regelmäßigen Abständen kryptografisches Material aus dem Speicher, wenn ihre Speicherdauer den konfigurierten Wert für die Gültigkeitsdauer (TTL) überschreitet.

  Sie speichern `MostRecentProvider` möglicherweise kryptografisches Material für die gesamte Lebensdauer des Prozesses im Speicher. Dies hat zur Folge, dass der neueste Anbieter von Autorisierungsänderungen möglicherweise nichts weiß. Möglicherweise werden Verschlüsselungsschlüssel verwendet, nachdem dem Anrufer die Berechtigungen zu deren Verwendung entzogen wurden.

  Wenn Sie nicht auf diese neue Version aktualisieren können, können Sie einen ähnlichen Effekt erzielen, indem Sie die `clear()` Methode regelmäßig im Cache aufrufen. Diese Methode löscht den Cache-Inhalt manuell und erfordert, dass der neueste Anbieter eine neue CMP und neue kryptografische Materialien anfordert.
- Dazu gehört `CachingMostRecentProvider` auch eine Einstellung für die Cachegröße, mit der Sie mehr Kontrolle über den Cache haben.

Um auf die zu aktualisieren`CachingMostRecentProvider`, müssen Sie den Symbolnamen in Ihrem Code ändern. In jeder anderen Hinsicht `CachingMostRecentProvider` ist das vollständig abwärtskompatibel mit dem`MostRecentProvider`. Sie müssen keine Tabellenelemente erneut verschlüsseln.

Das `CachingMostRecentProvider` generiert jedoch mehr Aufrufe an die zugrunde liegende Schlüsselinfrastruktur. Es ruft den Provider-Speicher in jedem Time-to-Live-Intervall (TTL) mindestens einmal auf. Anwendungen mit zahlreichen aktiven CMPs (aufgrund häufiger Rotation) oder Anwendungen mit großen Flotten reagieren höchstwahrscheinlich empfindlich auf diese Änderung.

Bevor Sie Ihren aktualisierten Code veröffentlichen, testen Sie ihn gründlich, um sicherzustellen, dass die häufigeren Aufrufe Ihre Anwendung nicht beeinträchtigen oder zu Drosselungen durch Dienste führen, von denen Ihr Anbieter abhängig ist, wie AWS Key Management Service (AWS KMS) oder Amazon DynamoDB. Um Leistungsprobleme zu vermeiden, passen Sie die Cachegröße und die Gültigkeitsdauer des Caches an die von Ihnen beobachteten Leistungsmerkmale an`CachingMostRecentProvider`. Anleitungen finden Sie unter Es wird ein Wert für die Gültigkeitsdauer festgelegt.

Static Materials Provider

> **Note**
>
> Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt. AWS Das folgende Thema enthält Informationen zu Versionen 1. x —2. x des DynamoDB Encryption Client für Java und Versionen 1. x —3. x des DynamoDB Encryption Client für Python. Weitere Informationen finden Sie unter AWS Database Encryption SDK für DynamoDB-Versionsunterstützung.

Der Static Materials Provider (Static CMP) ist ein sehr einfacher Anbieter für kryptografisches Material (CMP), der für Tests, Machbarkeitsnachweise und Kompatibilität mit älteren Versionen vorgesehen ist.

Um mit dem Static CMP ein Tabellenelement zu verschlüsseln, geben Sie einen symmetrischen Advanced Encryption Standard (AES)-Verschlüsselungsschlüssel und einen Signierschlüssel oder ein Schlüsselpaar an. Sie müssen die gleichen Schlüssel angeben, um das verschlüsselte Element zu entschlüsseln. Der Static CMP führt keine kryptografischen Vorgänge durch. Stattdessen

übergibt es die Verschlüsselungsschlüssel, die Sie dem Elementverschlüssler zur Verfügung stellen, unverändert. Der Elementverschlüsseler verschlüsselt die Elemente direkt unter dem Verschlüsselungsschlüssel. Anschließend verwendet er den Signierschlüssel, um sie direkt zu signieren.

Da der Static CMP keine eindeutigen kryptographischen Materialien erzeugt, werden alle Tabellenelemente, die Sie verarbeiten, mit demselben Verschlüsselungsschlüssel verschlüsselt und mit demselben Signierschlüssel signiert. Wenn Sie den gleichen Schlüssel verwenden, um die Attributwerte in verschiedenen Elementen zu verschlüsseln, oder wenn Sie den gleichen Schlüssel oder das gleiche Schlüsselpaar verwenden, um alle Elemente zu signieren, laufen Sie Gefahr, die kryptographischen Grenzen der Schlüssel zu überschreiten.

> **Note**
>
> Der Asymmetric Static Provider in der Java-Bibliothek ist kein statischer Anbieter. Er liefert nur alternative Konstruktoren für den Wrapped CMP. Er ist sicher für die Produktion, aber Sie sollten den Wrapped CMP nach Möglichkeit direkt verwenden.

Static CMP ist einer von mehreren Anbietern kryptografischer Materialien (CMPs), die der DynamoDB Encryption Client unterstützt. Weitere Information zu den anderen CMPs finden Sie unter Anbieter von kryptografischem Material.

Beispielcode finden Sie unter:

- Java: SymmetricEncryptedItem

Themen

- Verwendung
- Funktionsweise

Verwendung

Um einen statischen Anbieter zu erstellen, geben Sie einen Verschlüsselungsschlüssel oder ein Schlüsselpaar und einen Signierschlüssel oder ein Schlüsselpaar an. Sie müssen Schlüsselmaterial zum Ver- und Entschlüsseln von Tabellenelementen bereitstellen.

Java

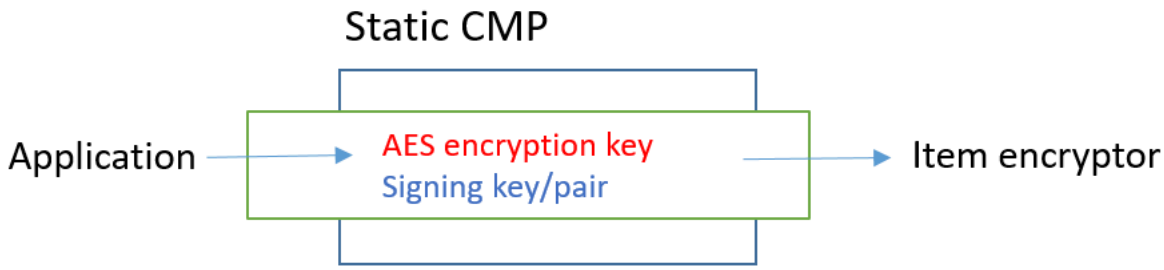
```
// To encrypt
SecretKey cek = ...;        // Encryption key
SecretKey macKey =  ...;    // Signing key
EncryptionMaterialsProvider provider = new SymmetricStaticProvider(cek, macKey);

// 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
)
```

Funktionsweise

Der Statische Provider übergibt die von Ihnen gelieferten Verschlüsselungs- und Signierschlüssel an den Elementverschlüssler, wo sie direkt zum Verschlüsseln und Signieren Ihrer Tabellenelemente verwendet werden. Wenn Sie nicht für jedes Element unterschiedliche Schlüssel angeben, werden für jedes Element die gleichen Schlüssel verwendet.



Verschlüsselungsmaterialien abrufen

Dieser Abschnitt beschreibt detailliert die Ein- und Ausgänge und die Verarbeitung des Static Materials Providers (Static CMP), wenn er eine Anfrage für Verschlüsselungsmaterialien erhält.

Eingabe (von der Anwendung)

- Ein Verschlüsselungsschlüssel — Dies muss ein symmetrischer Schlüssel sein, z. B. ein AES-Schlüssel (Advanced Encryption Standard).
- Ein Signaturschlüssel — Dies kann ein symmetrischer Schlüssel oder ein asymmetrisches key pair sein.

Eingabe (vom Elementverschlüssler)

- DynamoDB-Verschlüsselungskontext

Ausgabe (an den Elementverschlüssler)

- Der als Eingabe übergebene Verschlüsselungsschlüssel.
- Der als Eingabe übergebene Signierschlüssel.
- Tatsächliche Materialbeschreibung: Die angeforderte Materialbeschreibung, falls vorhanden, unverändert.

Entschlüsselungsmaterialien abrufen

Dieser Abschnitt beschreibt detailliert die Ein- und Ausgänge und die Verarbeitung des Static Materials Providers (Static CMP), wenn er eine Anfrage für Entschlüsselungsmaterialien erhält.

Obwohl er getrennte Methoden zum Abrufen von Verschlüsselungsmaterialien und Entschlüsselungsmaterialien enthält, ist das Verhalten das gleiche.

Eingabe (von der Anwendung)

- Ein Verschlüsselungsschlüssel — Dies muss ein symmetrischer Schlüssel sein, z. B. ein AES-Schlüssel (Advanced Encryption Standard).
- Ein Signaturschlüssel — Dies kann ein symmetrischer Schlüssel oder ein asymmetrisches key pair sein.

Eingabe (vom Elementverschlüssler)

- DynamoDB-Verschlüsselungskontext (nicht verwendet)

Ausgabe (an den Elementverschlüssler)

- Der als Eingabe übergebene Verschlüsselungsschlüssel.
- Der als Eingabe übergebene Signierschlüssel.

Verfügbare Programmiersprachen für Amazon DynamoDB Encryption Client

> **Note**
>
> Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt. AWS Das folgende Thema enthält Informationen zu Versionen 1. x —2. x des DynamoDB Encryption Client für Java und Versionen 1. x —3. x des DynamoDB Encryption Client für Python. Weitere Informationen finden Sie unter AWS Database Encryption SDK für DynamoDB-Versionsunterstützung.

Der Amazon DynamoDB Encryption Client ist für die folgenden Programmiersprachen verfügbar. Die sprachspezifischen Bibliotheken sind unterschiedlich, aber die daraus resultierenden Implementierungen sind interoperabel. Beispielsweise können Sie ein Element mit dem Java-Client verschlüsseln (und signieren) und das Element mit dem Python-Client entschlüsseln.

Weitere Informationen finden Sie unter den entsprechenden Themen.

Themen

- Amazon DynamoDB DynamoDB-Verschlüsselungsclient für Java
- DynamoDB-Verschlüsselungsclient für Python

Amazon DynamoDB DynamoDB-Verschlüsselungsclient für Java

> **Note**
>
> Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt. AWS Das folgende Thema enthält Informationen zu Versionen 1. x —2. x des DynamoDB Encryption Client für Java und Versionen 1. x —3. x des DynamoDB Encryption Client für Python. Weitere Informationen finden Sie unter AWS Database Encryption SDK für DynamoDB-Versionsunterstützung.

In diesem Thema wird erklärt, wie Sie den Amazon DynamoDB Encryption Client für Java installieren und verwenden. Einzelheiten zur Programmierung mit dem DynamoDB Encryption Client finden Sie in den Java-Beispielen, den Beispielen im aws-dynamodb-encryption-java-Repository auf GitHub und im Javadoc für den DynamoDB Encryption Client.

> **Note**
>
> Versionen 1. x. x der DynamoDB Encryption Client für Java befinden sich mit Wirkung zum Juli 2022 in der Endphase des Supports. Führen Sie so bald wie möglich ein Upgrade auf eine neuere Version durch.

Themen

- Voraussetzungen
- Installation
- Verwenden des DynamoDB Encryption Client für Java
- Beispielcode für den DynamoDB Encryption Client für Java

Voraussetzungen

Bevor Sie den Amazon DynamoDB Encryption Client für Java installieren, stellen Sie sicher, dass Sie die folgenden Voraussetzungen erfüllen.

Eine Java-Entwicklungsumgebung

Sie benötigen Java 8 oder höher. Klicken Sie auf der Oracle-Website auf Java SE Downloads und laden und installieren Sie anschließend das Java SE Development Kit (JDK).

Wenn Sie das Oracle JDK verwenden, müssen Sie auch die Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files herunterladen und installieren.

AWS SDK für Java

Der DynamoDB Encryption Client benötigt das DynamoDB-Modul von, AWS SDK für Java auch wenn Ihre Anwendung nicht mit DynamoDB interagiert. Sie können das gesamte SDK oder nur dieses Modul installieren. Wenn Sie Maven verwenden, fügen Sie `aws-java-sdk-dynamodb` Ihrer `pom.xml`-Datei hinzu.

Weitere Informationen zur Installation und Konfiguration von finden Sie unter. AWS SDK für JavaAWS SDK für Java

Installation

Sie können den Amazon DynamoDB Encryption Client für Java auf folgende Weise installieren.

manuell

Um den Amazon DynamoDB Encryption Client für Java zu installieren, klonen Sie das GitHub aws-dynamodb-encryption-java-Repository oder laden Sie es herunter.

Verwenden von Apache Maven

Der Amazon DynamoDB Encryption Client für Java ist über Apache Maven mit der folgenden Abhängigkeitsdefinition verfügbar.

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-dynamodb-encryption-java</artifactId>
  <version>version-number</version>
</dependency>
```

Nachdem Sie das SDK installiert haben, schauen Sie sich zunächst den Beispielcode in diesem Handbuch und den DynamoDB Encryption Client Javadoc an. GitHub

Verwenden des DynamoDB Encryption Client für Java

> **Note**
>
> Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt. AWS Das folgende Thema enthält Informationen zu Versionen 1. x —2. x des DynamoDB Encryption Client für Java und Versionen 1. x —3. x des DynamoDB Encryption Client für Python. Weitere Informationen finden Sie unter AWS Database Encryption SDK für DynamoDB-Versionsunterstützung.

In diesem Thema werden einige Funktionen des DynamoDB Encryption Client in Java erklärt, die in anderen Programmiersprachenimplementierungen möglicherweise nicht zu finden sind.

Einzelheiten zur Programmierung mit dem DynamoDB Encryption Client finden Sie in den Java-Beispielen, den Beispielen im `aws-dynamodb-encryption-java repository` on GitHub und im Javadoc für den DynamoDB Encryption Client.

Themen

- Artikelverschlüsseler: DynamoDBEncryptor
- Attributaktionen in Java
- Überschreiben von Tabellennamen

Artikelverschlüsseler: DynamoDBEncryptor

Der DynamoDB Encryption Client in Java hat einen Elementverschlüsseler: den DynamoDBEncryptor auf niedrigerer Ebene

Attributaktionen in Java

Attribut-Aktionen bestimmen, welche Attributwerte verschlüsselt und signiert, welche nur signiert und welche ignoriert werden.

> **Important**
>
> Nachdem Sie die Attributaktionen zum Verschlüsseln der Tabellenelemente verwendet haben, kann das Hinzufügen oder Entfernen von Attributen zu oder aus Ihrem Datenmodell

einen Signaturvalidierungsfehler verursachen, der ein Entschlüsseln Ihrer Daten verhindert. Eine detaillierte Beschreibung finden Sie unter Ändern Ihres Datenmodells.

Attributaktionen für den DynamoDBEncryptor

Um Attributaktionen anzugeben, wenn Sie den DynamoDBEncryptor direkt verwenden, erstellen Sie ein `HashMap`-Objekt, in dem Attributnamen und die angegebenen Aktionen durch die Name-Wert-Paare repräsentiert werden.

Die gültigen Werte für die Attribut-Aktionen sind unter im Aufzählungstyp `EncryptionFlags` definiert. Sie können `ENCRYPT` und `SIGN` gemeinsam oder `SIGN` alleine verwenden, oder beides weglassen. Wenn Sie den DynamoDB Encryption Client jedoch `ENCRYPT` alleine verwenden, gibt er einen Fehler aus. Ein Attribut, das Sie nicht signieren, können Sie nicht verschlüsseln.

```
ENCRYPT
SIGN
```

> **Warning**
>
> Verschlüsseln Sie die primären Schlüsselattribute nicht. Sie müssen im Klartext bleiben, damit DynamoDB das Element finden kann, ohne einen vollständigen Tabellenscan ausführen zu müssen.

Wenn Sie einen Primärschlüssel im Verschlüsselungskontext angeben und dann `ENCRYPT` in der Attributaktion für eines der Primärschlüsselattribute angeben, löst der DynamoDB Encryption Client eine Ausnahme aus.

Der folgende Java-Code erstellt beispielsweise eine `actions` HashMap , die alle Attribute im Element verschlüsselt und signiert. `record` Ausnahmen sind die Partitionsschlüssel- und Sortierschlüsselattribute, die zwar signiert, aber nicht verschlüsselt sind, sowie das `test`-Attribut, das weder signiert noch verschlüsselt ist.

```
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;
  }
}
```

Wenn Sie dann die encryptRecord-Methode des `DynamoDBEncryptor` aufrufen, geben Sie die Map als Wert des Parameters `attributeFlags` an. Der folgende Aufruf von `encryptRecord` beispielsweise verwendet die `actions`-Map.

```
// Encrypt the plaintext record
final Map<String, AttributeValue> encrypted_record = encryptor.encryptRecord(record,
 actions, encryptionContext);
```

Überschreiben von Tabellennamen

Im DynamoDB Encryption Client ist der Name der DynamoDB-Tabelle ein Element des DynamoDB-Verschlüsselungskontextes, das an die Verschlüsselungs - und Entschlüsselungsmethoden übergeben wird. Wenn Sie Tabellenelemente verschlüsseln oder signieren, ist der DynamoDB-Verschlüsselungskontext, einschließlich des Tabellennamens, kryptografisch an den Chiffretext gebunden. Wenn der DynamoDB-Verschlüsselungskontext, der an die Entschlüsselungsmethode übergeben wird, nicht mit dem DynamoDB-Verschlüsselungskontext übereinstimmt, der an die Verschlüsselungsmethode übergeben wurde, schlägt der Entschlüsselungsvorgang fehl.

Gelegentlich ändert sich der Name einer Tabelle, z. B. wenn Sie eine Tabelle sichern oder eine zeitpunktbezogene Wiederherstellung durchführen. Wenn Sie die Signatur dieser Elemente entschlüsseln oder überprüfen, müssen Sie denselben DynamoDB-Verschlüsselungskontext übergeben, der zum Verschlüsseln und Signieren der Elemente verwendet wurde, einschließlich des ursprünglichen Tabellennamens. Der aktuelle Tabellenname wird nicht benötigt.

Wenn Sie den verwenden`DynamoDBEncryptor`, stellen Sie den DynamoDB-Verschlüsselungskontext manuell zusammen. Verwenden Sie daher nicht die Operatoren zum Überschreiben von Tabellennamen, wenn Sie die verwenden. `DynamoDBEncryptor` Erstellen Sie stattdessen einen Verschlüsselungskontext mit dem ursprünglichen Tabellennamen und senden Sie ihn an die Entschlüsselungsmethode.

Beispielcode für den DynamoDB Encryption Client für Java

> **Note**
>
> Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt. AWS Das folgende Thema enthält Informationen zu Versionen 1. x —2. x des DynamoDB Encryption Client für Java und Versionen 1. x —3. x des DynamoDB Encryption Client für Python. Weitere Informationen finden Sie unter AWS Database Encryption SDK für DynamoDB-Versionsunterstützung.

Die folgenden Beispiele zeigen Ihnen, wie Sie den DynamoDB Encryption Client für Java verwenden, um DynamoDB-Tabellenelemente in Ihrer Anwendung zu schützen. Weitere Beispiele (und eigene Beispiele) finden Sie im Beispielverzeichnis des aws-database-encryption-sdk-dynamodb-Repositorys unter. GitHub

Themen

- Verwenden des DynamoDBEncryptor

Verwenden des DynamoDBEncryptor

Dieses Beispiel zeigt Ihnen, wie Sie den untergeordneten DynamoDBEncryptor mit dem Direct KMS Provider verwenden. Der Direct KMS-Anbieter generiert und schützt seine kryptografischen Materialien unter einem von Ihnen angegebenen Wert in (). AWS KMS key AWS Key Management Service AWS KMS

Sie können jeden kompatiblen Anbieter für kryptografisches Material (CMP) mit dem verwenden. `DynamoDBEncryptor`

Sehen Sie sich das vollständige Codebeispiel an: AwsKmsEncryptedItem.java

Schritt 1: Erstellen Sie den Direct KMS Provider

Erstellen Sie eine Instanz des AWS KMS Clients mit der angegebenen Region. Verwenden Sie dann die Client-Instanz, um eine Instanz des Direct KMS Providers mit Ihrem bevorzugten zu erstellen AWS KMS key.

In diesem Beispiel wird der Amazon-Ressourcenname (ARN) verwendet, um den zu identifizieren AWS KMS key, aber Sie können jeden gültigen Schlüsselbezeichner verwenden.

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

final KmsClient kms = KmsClient.builder().region(Region.of(region)).build();
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, keyArn);
```

Schritt 2: Erstellen Sie ein Element

In diesem Beispiel wird a definiert `record` HashMap , das ein Beispieltabellenelement darstellt.

```
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"));
```

Schritt 3: Erstellen Sie einen DynamoDBEncryptor

Erstellen Sie eine Instance von `DynamoDBEncryptor` mit dem Direct KMS Provider.

```
final DynamoDBEncryptor encryptor = DynamoDBEncryptor.getInstance(cmp);
```

Schritt 4: Erstellen Sie einen DynamoDB-Verschlüsselungskontext

Der DynamoDB-Verschlüsselungskontext enthält Informationen über die Tabellenstruktur und wie sie verschlüsselt und signiert ist.

```
final String tableName = "testTable";

final EncryptionContext encryptionContext = new EncryptionContext.Builder()
    .withTableName(tableName)
    .withHashKeyName(partitionKeyName)
    .withRangeKeyName(sortKeyName)
    .build();
```

Schritt 5: Erstellen Sie das Attribut-Aktionen-Objekt

Attribut-Aktionen bestimmen, welche Attribute des Elements verschlüsselt und signiert sind, welche nur signiert und welche nicht verschlüsselt oder signiert sind.

In Java erstellen Sie zur Angabe von Attributaktionen eine Kombination HashMap aus Attributnamen und `EncryptionFlags` Wertepaaren.

Der folgende Java-Code erstellt beispielsweise eine, `actions` HashMap die alle Attribute im `record` Element verschlüsselt und signiert, mit Ausnahme der Partitionsschlüssel- und Sortierschlüsselattribute, die signiert, aber nicht verschlüsselt sind, und des `test` Attributs, das nicht signiert oder verschlüsselt ist.

```
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;
  }
}
```

Schritt 6: Verschlüsseln und signieren Sie das Element

Um das Tabellenelement zu verschlüsseln und zu signieren, rufen Sie die Methode `encryptRecord` für die Instance des `DynamoDBEncryptor` auf. Geben Sie das Tabellenelement (`record`), die Attribut-Aktionen (`actions`) und den Verschlüsselungskontext (`encryptionContext`) an.

```
final Map<String, AttributeValue> encrypted_record = encryptor.encryptRecord(record,
 actions, encryptionContext);
```

Schritt 7: Fügen Sie das Element in die DynamoDB-Tabelle ein

Fügen Sie abschließend das verschlüsselte und signierte Element in die DynamoDB-Tabelle ein.

```
final DynamoDbClient ddb =
 DynamoDbClient.builder().region(Region.of(region)).build();
ddb.putItem(tableName, encrypted_record);
```

DynamoDB-Verschlüsselungsclient für Python

> **Note**
>
> Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt. AWS Das folgende Thema enthält Informationen zu Versionen 1. x —2. x des DynamoDB Encryption Client für Java und Versionen 1. x —3. x des DynamoDB Encryption Client für Python. Weitere Informationen finden Sie unter AWS Database Encryption SDK für DynamoDB-Versionsunterstützung.

In diesem Thema wird erklärt, wie Sie den DynamoDB Encryption Client für Python installieren und verwenden. Sie finden den Code im aws-dynamodb-encryption-python-Repository unter GitHub, einschließlich des vollständigen und getesteten Beispielcodes, der Ihnen den Einstieg erleichtert.

> **Note**
>
> Versionen 1. x. x und 2. x. x des DynamoDB Encryption Client für Python befinden sich mit Wirkung zum Juli 2022 in der Endphase des Supports. Führen Sie so bald wie möglich ein Upgrade auf eine neuere Version durch.

Themen

Voraussetzungen

Bevor Sie den Amazon DynamoDB Encryption Client für Python installieren, stellen Sie sicher, dass Sie die folgenden Voraussetzungen erfüllen.

Eine unterstützte Version von Python

Python 3.8 oder höher ist für den Amazon DynamoDB Encryption Client für Python-Versionen 3.3.0 und höher erforderlich. Weitere Informationen zum Download von Python finden Sie unter Python-Downloads.

Frühere Versionen des Amazon DynamoDB Encryption Client für Python unterstützen Python 2.7 und Python 3.4 und höher, wir empfehlen jedoch, die neueste Version des DynamoDB Encryption Client zu verwenden.

Das pip-Installationstool for Python

Python 3.6 und höher enthalten Pip, obwohl Sie es vielleicht aktualisieren möchten. Weitere Informationen zum Aktualisieren oder Installieren von pip finden Sie unter Installation in der Dokumentation zu pip.

Installation

Verwenden Sie pip, um den Amazon DynamoDB Encryption Client für Python zu installieren, wie in den folgenden Beispielen gezeigt.

Installieren der neuesten Version

```
pip install dynamodb-encryption-sdk
```

Weitere Informationen zur Verwendung von pip für die Installation und die Aktualisierung von Paketen finden Sie unter Pakete installieren.

Der DynamoDB Encryption Client benötigt die Kryptografiebibliothek auf allen Plattformen. Alle Versionen von pip installieren und erstellen die Kryptographie-Bibliothek unter Windows. pip 8.1 und höher installiert und erstellt cryptography auf Linux. Wenn Sie eine frühere Version von pip verwenden und Ihre Linux-Umgebung nicht über die erforderlichen Tools zum Erstellen der Kryptographie-Bibliothek verfügt, müssen Sie sie installieren. Weitere Informationen finden Sie unter Kryptographie unter Linux.

Sie können die neueste Entwicklungsversion des DynamoDB Encryption Client aus dem aws-dynamodb-encryption-python Repository unter herunterladen. GitHub

Nachdem Sie den DynamoDB Encryption Client installiert haben, schauen Sie sich zunächst den Python-Beispielcode in diesem Handbuch an.

Den DynamoDB Encryption Client für Python verwenden

> **Note**
>
> Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt. AWS Das folgende Thema enthält Informationen zu Versionen 1. x —2. x des DynamoDB Encryption Client für Java und Versionen 1. x —3. x des DynamoDB Encryption Client für Python. Weitere Informationen finden Sie unter AWS Database Encryption SDK für DynamoDB-Versionsunterstützung.

In diesem Thema werden einige Funktionen des DynamoDB Encryption Client für Python erläutert, die in anderen Programmiersprachenimplementierungen möglicherweise nicht zu finden sind. Diese Funktionen sollen es einfacher machen, den DynamoDB Encryption Client auf die sicherste Art und Weise zu verwenden. Wenn Sie keinen ungewöhnlichen Anwendungsfall haben, empfehlen wir Ihnen, sie zu verwenden.

Einzelheiten zur Programmierung mit dem DynamoDB Encryption Client finden Sie in den Python-Beispielen in diesem Handbuch, in den Beispielen im aws-dynamodb-encryption-python Repository auf GitHub und in der Python-Dokumentation für den DynamoDB Encryption Client.

Themen

- Client-Helferklassen
- TableInfo Klasse
- Attributaktionen in Python

Client-Helferklassen

Der DynamoDB Encryption Client für Python umfasst mehrere Client-Hilfsklassen, die die Boto 3-Klassen für DynamoDB widerspiegeln. Diese Hilfsklassen sollen das Hinzufügen von Verschlüsselung und Signierung zu Ihrer vorhandenen DynamoDB-Anwendung vereinfachen und die häufigsten Probleme wie folgt vermeiden:

- Verhindern Sie, dass Sie den Primärschlüssel in Ihrem Element verschlüsseln, indem Sie dem AttributeActionsObjekt entweder eine Aktion zum Überschreiben des Primärschlüssels hinzufügen oder indem Sie eine Ausnahme auslösen, wenn Ihr `AttributeActions` Objekt den Client ausdrücklich auffordert, den Primärschlüssel zu verschlüsseln. Wenn die Standardaktion in Ihrem `AttributeActions`-Objekt `DO_NOTHING` ist, verwenden die Client-Helferklassen diese Aktion für den Primärschlüssel. Andernfalls verwenden sie `SIGN_ONLY`.
- Erstellen Sie ein TableInfo Objekt und füllen Sie den DynamoDB-Verschlüsselungskontext auf der Grundlage eines DynamoDB-Aufrufs auf. Auf diese Weise können Sie sicherstellen, dass Ihr DynamoDB-Verschlüsselungskontext korrekt ist und der Client den Primärschlüssel identifizieren kann.
- Support Sie Methoden wie `put_item` und`get_item`, die Ihre Tabellenelemente transparent ver- und entschlüsseln, wenn Sie in eine DynamoDB-Tabelle schreiben oder aus einer DynamoDB-Tabelle lesen. Nur die Methode `update_item` wird nicht unterstützt.

Sie können die Client-Helferklassen verwenden, anstatt direkt mit dem untergeordneten Elementverschlüssler zu interagieren. Verwenden Sie diese Klassen, es sei denn, Sie müssen erweiterte Optionen im Elementverschlüssler festlegen.

Zu den Client-Helferklassen gehören:

- EncryptedTablefür Anwendungen, die die Tabellenressource in DynamoDB verwenden, um jeweils eine Tabelle zu verarbeiten.
- EncryptedResourcefür Anwendungen, die die Service Resource-Klasse in DynamoDB für die Batchverarbeitung verwenden.
- EncryptedClientfür Anwendungen, die den Lower-Level-Client in DynamoDB verwenden.

Um die Client-Hilfsklassen verwenden zu können, muss der Aufrufer die Berechtigung haben, den DescribeTableDynamoDB-Vorgang in der Zieltabelle aufzurufen.

TableInfo Klasse

Die TableInfoKlasse ist eine Hilfsklasse, die eine DynamoDB-Tabelle darstellt, komplett mit Feldern für den Primärschlüssel und die Sekundärindizes. Sie hilft Ihnen, genaue Informationen über die Tabelle in Echtzeit zu erhalten.

Wenn Sie eine Client-Helferklasse verwenden, erstellt und verwendet sie ein `TableInfo`-Objekt für Sie. Ansonsten können Sie explizit ein solches anlegen. Ein Beispiel finden Sie unter Verwendung des Elementverschlüsslers.

Wenn Sie die `refresh_indexed_attributes` Methode für ein `TableInfo` Objekt aufrufen, füllt sie die Eigenschaftswerte des Objekts auf, indem sie den DescribeTableDynamoDB-Vorgang aufruft. Die Abfrage der Tabelle ist wesentlich zuverlässiger als eine feste Codierung von Indexnamen. Die `TableInfo` Klasse enthält auch eine `encryption_context_values` Eigenschaft, die die erforderlichen Werte für den DynamoDB-Verschlüsselungskontext bereitstellt.

Um die `refresh_indexed_attributes` Methode verwenden zu können, muss der Aufrufer über die Berechtigung verfügen, den DescribeTableDynamoDB-Vorgang in der Zieltabelle aufzurufen.

Attributaktionen in Python

Attribut-Aktionen teilen dem Elementverschlüsseler mit, welche Aktionen er auf jedes Attribut des Elements anwenden soll. Um Attribut-Aktionen in Python anzugeben, legen Sie ein `AttributeActions`-Objekt mit einer Standardaktion und Ausnahmen für bestimmte Attribute an. Die gültigen Werte sind im Aufzählungstyp `CryptoAction` definiert.

> **Important**
>
> Nachdem Sie die Attributaktionen zum Verschlüsseln der Tabellenelemente verwendet haben, kann das Hinzufügen oder Entfernen von Attributen zu oder aus Ihrem Datenmodell einen Signaturvalidierungsfehler verursachen, der ein Entschlüsseln Ihrer Daten verhindert. Eine detaillierte Beschreibung finden Sie unter Ändern Ihres Datenmodells.

```
DO_NOTHING = 0
SIGN_ONLY = 1
ENCRYPT_AND_SIGN = 2
```

Beispielsweise richtet dieses `AttributeActions`-Objekt `ENCRYPT_AND_SIGN` als Standard für alle Attribute ein und gibt Ausnahmen für die Attribute `ISBN` und `PublicationYear` an.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={
        'ISBN': CryptoAction.DO_NOTHING,
        'PublicationYear': CryptoAction.SIGN_ONLY
    }
)
```

Wenn Sie eine Client-Helferklasse verwenden, müssen Sie keine Attribut-Aktion für die Primärschlüsselattribute angeben. Die Client-Helferklassen verhindern, dass Sie Ihren Primärschlüssel verschlüsseln.

Wenn Sie keine Client-Helferklasse verwenden und die Standardaktion `ENCRYPT_AND_SIGN` ist, müssen Sie eine Aktion für den Primärschlüssel angeben. Die empfohlene Aktion für Primärschlüssel ist `SIGN_ONLY`. Um dies zu vereinfachen, verwenden Sie die Methode `set_index_keys`, die SIGN_ONLY für Primärschlüssel verwendet, oder DO_NOTHING, wenn dies die Standardaktion ist.

> **Warning**
>
> Verschlüsseln Sie die primären Schlüsselattribute nicht. Sie müssen im Klartext bleiben, damit DynamoDB das Element finden kann, ohne einen vollständigen Tabellenscan ausführen zu müssen.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
)
actions.set_index_keys(*table_info.protected_index_keys())
```

Beispielcode für den DynamoDB Encryption Client für Python

> **Note**
>
> Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt. AWS Das folgende Thema enthält Informationen zu Versionen 1. x —2. x des DynamoDB Encryption Client für Java und Versionen 1. x —3. x des DynamoDB Encryption Client für Python. Weitere Informationen finden Sie unter AWS Database Encryption SDK für DynamoDB-Versionsunterstützung.

Die folgenden Beispiele zeigen Ihnen, wie Sie den DynamoDB Encryption Client für Python verwenden, um DynamoDB-Daten in Ihrer Anwendung zu schützen. Weitere Beispiele (und eigene Beispiele) finden Sie im Beispielverzeichnis des aws-dynamodb-encryption-python-Repositorys unter. GitHub

Themen

- Verwenden EncryptedTable Sie die Client-Helper-Klasse
- Verwendung des Elementverschlüsslers

Verwenden EncryptedTable Sie die Client-Helper-Klasse

Das folgende Beispiel zeigt Ihnen, wie Sie den Direct KMS Provider mit der `EncryptedTable`-Client-Helferklasse verwenden. Dieses Beispiel verwendet denselben Anbieter von Verschlüsselungsdaten wie das nachfolgende Beispiel Verwendung des Elementverschlüsslers. Es verwendet jedoch die Klasse `EncryptedTable`, statt direkt mit dem untergeordneten Elementverschlüssler zusammenzuarbeiten.

Durch den Vergleich dieser Beispiele erkennen Sie, was die Client-Helferklasse für Sie erledigt. Dazu gehört die Erstellung des DynamoDB-Verschlüsselungskontextes und die Sicherstellung, dass die Primärschlüsselattribute immer signiert, aber niemals verschlüsselt sind. Um den Verschlüsselungskontext zu erstellen und den Primärschlüssel zu ermitteln, rufen die Client-Hilfsklassen den DynamoDB-Vorgang DescribeTableauf. Um diesen Code ausführen zu können, müssen Sie die Berechtigung haben, diese Operation aufzurufen.

Das vollständige Codebeispiel finden Sie unter: aws_kms_encrypted_table.py

Schritt 1: Erstellen der Tabelle

Erstellen Sie zunächst eine Instanz einer DynamoDB-Standardtabelle mit dem Tabellennamen.

```
table_name='test-table'
table = boto3.resource('dynamodb').Table(table_name)
```

Schritt 2: Erstellen Sie einen Anbieter für Verschlüsselungsdaten

Erstellen Sie eine Instance des Anbieters für Verschlüsselungsdaten (CMP, Cryptographic Materials Provider), den Sie ausgewählt haben.

Dieses Beispiel verwendet den Direct KMS Provider, Sie können aber jeden beliebigen kompatiblen CMP verwenden. Um einen Direct KMS-Anbieter zu erstellen, geben Sie einen an.

AWS KMS key In diesem Beispiel wird der Amazon-Ressourcenname (ARN) von verwendet AWS KMS key, Sie können jedoch jeden gültigen Schlüsselbezeichner verwenden.

```
kms_key_id='arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key_id)
```

Schritt 3: Erstellen Sie das Attribut-Aktionen-Objekt

Attribut-Aktionen teilen dem Elementverschlüsseler mit, welche Aktionen er auf jedes Attribut des Elements anwenden soll. Das `AttributeActions`-Objekt in diesem Beispiel verschlüsselt und signiert alle Elemente außer dem Attribut `test`, das ignoriert wird.

Geben Sie keine Attribut-Aktionen für die Primärschlüsselattribute an, wenn Sie eine Client-Helferklasse verwenden. Die `EncryptedTable`-Klassen signiert die Primärschlüsselattribute, verschlüsselt sie aber nie.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={'test': CryptoAction.DO_NOTHING}
)
```

Schritt 4: Erstellen Sie die verschlüsselte Tabelle

Legen Sie die verschlüsselte Tabelle mit der Standardtabelle, dem Direct KMS Provider und den Attribut-Aktionen an. Dieser Schritt schließt die Konfiguration ab.

```
encrypted_table = EncryptedTable(
    table=table,
    materials_provider=kms_cmp,
    attribute_actions=actions
)
```

Schritt 5: Legen Sie das Klartext-Element in der Tabelle ab

Wenn Sie die `put_item` Methode für aufrufen`encrypted_table`, werden Ihre Tabellenelemente transparent verschlüsselt, signiert und zu Ihrer DynamoDB-Tabelle hinzugefügt.

Definieren Sie zunächst das Tabellenelement.

```
plaintext_item = {
```

```
    'partition_attribute': 'value1',
    'sort_attribute': 55
    'example': 'data',
    'numbers': 99,
    'binary': Binary(b'\x00\x01\x02'),
    'test': 'test-value'
}
```

Anschließend legen Sie es in der Tabelle ab.

```
encrypted_table.put_item(Item=plaintext_item)
```

Um das Element in verschlüsselter Form aus der DynamoDB-Tabelle abzurufen, rufen Sie die `get_item` Methode für das Objekt auf. `table` Um das verschlüsselte Element abzurufen, rufen Sie die Methode `get_item` für das Objekt `encrypted_table` auf.

Verwendung des Elementverschlüsslers

Dieses Beispiel zeigt Ihnen, wie Sie beim Verschlüsseln von Tabellenelementen direkt mit dem Elementverschlüsseler im DynamoDB Encryption Client interagieren können, anstatt die Client-Hilfsklassen zu verwenden, die für Sie mit dem Elementverschlüsseler interagieren.

Wenn Sie diese Technik verwenden, erstellen Sie den DynamoDB-Verschlüsselungskontext und das Konfigurationsobjekt (`CryptoConfig`) manuell. Außerdem verschlüsseln Sie die Elemente in einem Aufruf und fügen sie in einem separaten Aufruf in Ihre DynamoDB-Tabelle ein. Auf diese Weise können Sie Ihre `put_item` Aufrufe anpassen und den DynamoDB Encryption Client verwenden, um strukturierte Daten zu verschlüsseln und zu signieren, die niemals an DynamoDB gesendet werden.

Dieses Beispiel verwendet den Direct KMS Provider, Sie können aber jeden beliebigen kompatiblen CMP verwenden.

Das vollständige Codebeispiel finden Sie unter: aws_kms_encrypted_item.py

Schritt 1: Erstellen der Tabelle

Erstellen Sie zunächst eine Instanz einer standardmäßigen DynamoDB-Tabellenressource mit dem Tabellennamen.

```
table_name='test-table'
table = boto3.resource('dynamodb').Table(table_name)
```

Schritt 2: Erstellen Sie einen Anbieter für Verschlüsselungsdaten

Erstellen Sie eine Instance des Anbieters für Verschlüsselungsdaten (CMP, Cryptographic Materials Provider), den Sie ausgewählt haben.

Dieses Beispiel verwendet den Direct KMS Provider, Sie können aber jeden beliebigen kompatiblen CMP verwenden. Um einen Direct KMS-Anbieter zu erstellen, geben Sie einen an. AWS KMS key In diesem Beispiel wird der Amazon-Ressourcenname (ARN) von verwendet AWS KMS key, Sie können jedoch jeden gültigen Schlüsselbezeichner verwenden.

```
kms_key_id='arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key_id)
```

Schritt 3: Verwenden Sie die TableInfo Helper-Klasse

Um Informationen über die Tabelle von DynamoDB zu erhalten, erstellen Sie eine Instanz der TableInfoHelper-Klasse. Wenn Sie direkt mit dem Elementverschlüssler arbeiten, müssen Sie eine `TableInfo`-Instance erstellen und deren Methoden aufrufen. Dies erledigt die Client-Helferklasse für Sie.

Die `refresh_indexed_attributes` Methode von `TableInfo` verwendet den DescribeTableDynamoDB-Vorgang, um in Echtzeit genaue Informationen über die Tabelle abzurufen. Dazu gehören ihre Primärschlüssel und ihre lokalen und globalen Sekundärindizes. Der Aufrufer benötigt die Berechtigung, `DescribeTable` aufzurufen.

```
table_info = TableInfo(name=table_name)
table_info.refresh_indexed_attributes(table.meta.client)
```

Schritt 4: DynamoDB-Verschlüsselungskontext erstellen

Der DynamoDB-Verschlüsselungskontext enthält Informationen über die Tabellenstruktur und wie sie verschlüsselt und signiert ist. In diesem Beispiel wird explizit ein DynamoDB-Verschlüsselungskontext erstellt, da er mit dem Elementverschlüsseler interagiert. Die Client-Hilfsklassen erstellen den DynamoDB-Verschlüsselungskontext für Sie.

Um den Partitionsschlüssel und den Sortierschlüssel abzurufen, können Sie die Eigenschaften der TableInfoHilfsklasse verwenden.

```
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)
)
```

Schritt 5: Erstellen Sie das Attribut-Aktionen-Objekt

Attribut-Aktionen teilen dem Elementverschlüsseler mit, welche Aktionen er auf jedes Attribut des Elements anwenden soll. Das `AttributeActions`-Objekt in diesem Beispiel verschlüsselt und signiert alle Elemente mit Ausnahme der Primärschlüsselattribute, die signiert, aber nicht verschlüsselt sind, und des Attributs `test`, das ignoriert wird.

Wenn Sie direkt mit dem Elementverschlüsseler interagieren und Ihre Standardaktion `ENCRYPT_AND_SIGN` ist, müssen Sie eine alternative Aktion für den Primärschlüssel angeben. Sie können die `set_index_keys`-Methode verwenden, die `SIGN_ONLY` für den Primärschlüssel verwendet, oder `DO_NOTHING`, wenn es die Standardaktion ist.

Um den Primärschlüssel anzugeben, verwendet dieses Beispiel die Indexschlüssel im TableInfoObjekt, das durch einen Aufruf von DynamoDB aufgefüllt wird. Diese Technik ist sicherer als die Verwendung fest codierter Primärschlüsselnamen.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={'test': CryptoAction.DO_NOTHING}
)
actions.set_index_keys(*table_info.protected_index_keys())
```

Schritt 6: Erstellen Sie die Konfiguration für das Element

Um den DynamoDB Encryption Client zu konfigurieren, verwenden Sie die Objekte, die Sie gerade in einer CryptoConfigKonfiguration für das Tabellenelement erstellt haben. Die Client-Helper-Klassen erstellen das CryptoConfig für Sie.

```
crypto_config = CryptoConfig(
    materials_provider=kms_cmp,
```

```
    encryption_context=encryption_context,
    attribute_actions=actions
)
```

Schritt 7: Verschlüsseln Sie das Element

In diesem Schritt wird das Element verschlüsselt und signiert, es wird jedoch nicht in die DynamoDB-Tabelle aufgenommen.

Wenn Sie eine Client-Hilfsklasse verwenden, werden Ihre Elemente transparent verschlüsselt und signiert und dann zu Ihrer DynamoDB-Tabelle hinzugefügt, wenn Sie die `put_item` Methode der Hilfsklasse aufrufen. Wenn Sie den Elementverschlüsseler direkt verwenden, sind die Verschlüsselungs- und Put-Aktionen voneinander unabhängig.

Erstellen Sie zunächst ein Klartext-Element.

```
plaintext_item = {
    'partition_attribute': 'value1',
    'sort_key': 55,
    'example': 'data',
    'numbers': 99,
    'binary': Binary(b'\x00\x01\x02'),
    'test': 'test-value'
}
```

Anschließend verschlüsseln und signieren Sie es. Für die `encrypt_python_item`-Methode ist das `CryptoConfig`-Konfigurationsobjekt erforderlich.

```
encrypted_item = encrypt_python_item(plaintext_item, crypto_config)
```

Schritt 8: Legen Sie das Element in der Tabelle ab

In diesem Schritt wird das verschlüsselte und signierte Element in die DynamoDB-Tabelle eingefügt.

```
table.put_item(Item=encrypted_item)
```

Um das verschlüsselte Element anzuzeigen, rufen Sie die `get_item`-Methode für das ursprünglichen `table`-Objekt statt für das `encrypted_table`-Objekt auf. Sie ruft das Element aus der DynamoDB-Tabelle ab, ohne es zu verifizieren und zu entschlüsseln.

```
encrypted_item = table.get_item(Key=partition_key)['Item']
```

Das folgende Bild zeigt einen Teil eines verschlüsselten und signierten Beispiel-Tabellenelements.

Die verschlüsselten Attributwerte sind Binärdaten. Die Namen und Werte der Primärschlüsselattribute (`partition_attribute` und `sort_attribute`) und das `test`-Attribut verbleiben im Klartext. Die Ausgabe zeigt auch das Attribut, das die Signatur (`*amzn-ddb-map-sig*`) und das Materialbeschreibungsattribut (`*amzn-ddb-map-desc*`) enthält.

```
{
    '*amzn-ddb-map-desc*': Binary(b'\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'
}
```

Ändern Ihres Datenmodells

> **Note**
>
> Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt. AWS Das folgende Thema enthält Informationen zu Versionen 1. x —2. x des DynamoDB Encryption Client für Java und Versionen 1. x —3. x des DynamoDB Encryption Client für Python. Weitere Informationen finden Sie unter AWS Database Encryption SDK für DynamoDB-Versionsunterstützung.

Jedes Mal, wenn Sie ein Element ver- oder entschlüsseln, müssen Sie Attributaktionen angeben, die dem DynamoDB Encryption Client mitteilen, welche Attribute verschlüsselt und signiert, welche Attribute signiert (aber nicht verschlüsselt) und welche ignoriert werden sollen. Attributaktionen werden nicht im verschlüsselten Element gespeichert und der DynamoDB Encryption Client aktualisiert Ihre Attributaktionen nicht automatisch.

> **Important**
>
> Der DynamoDB Encryption Client unterstützt nicht die Verschlüsselung vorhandener, unverschlüsselter DynamoDB-Tabellendaten.

Wenn Sie Ihr Datenmodell ändern, d. h. wenn Sie Attribute zu Ihren Tabellenelementen hinzufügen oder von ihnen entfernen, riskieren Sie einen Fehler. Wenn die von Ihnen angegebenen Attribut-Aktionen nicht alle Attribute im Element berücksichtigen, wird das Element möglicherweise nicht so verschlüsselt und signiert, wie Sie es beabsichtigen. Wenn die Attributaktionen, die Sie beim Entschlüsseln eines Elements angeben, von den Attributaktionen abweichen, die Sie beim Verschlüsseln des Elements angegeben haben, kann zudem die Signaturprüfung fehlschlagen.

Wenn zum Beispiel die Attribut-Aktionen, die zum Verschlüsseln des Elements verwendet werden, es anweisen, das Attribut `test` zu signieren, wird die Signatur im Element das Attribut `test` enthalten. Aber wenn die Attribut-Aktionen, die zum Entschlüsseln des Elements verwendet werden, das Attribut `test` nicht berücksichtigen, schlägt die Überprüfung fehl, weil der Client versucht, eine Signatur zu verifizieren, die das Attribut `test` nicht enthält.

Dies ist ein besonderes Problem, wenn mehrere Anwendungen dieselben DynamoDB-Elemente lesen und schreiben, da der DynamoDB Encryption Client dieselbe Signatur für Elemente in allen Anwendungen berechnen muss. Dies ist auch ein Problem für jede verteilte Anwendung, da Änderungen an Attributaktionen auf alle Hosts übertragen werden müssen. Selbst wenn ein Host in einem Prozess auf Ihre DynamoDB-Tabellen zugreift, hilft die Einrichtung eines Best-Practice-Prozesses dabei, Fehler zu vermeiden, falls das Projekt einmal komplexer wird.

Verwenden Sie die folgenden Anleitungen, um Signaturvalidierungsfehler zu vermeiden, die das Lesen der Tabellenelemente verhindern.

- Hinzufügen eines Attributs — Wenn das neue Attribut Ihre Attributaktionen ändert, implementieren Sie die Änderung der Attributaktion vollständig, bevor Sie das neue Attribut in ein Element aufnehmen.
- Ein Attribut entfernen — Wenn Sie ein Attribut nicht mehr in Ihren Artikeln verwenden, ändern Sie Ihre Attributaktionen nicht.
- Aktion ändern — Nachdem Sie eine Konfiguration für Attributaktionen zum Verschlüsseln Ihrer Tabellenelemente verwendet haben, können Sie die Standardaktion oder die Aktion für ein vorhandenes Attribut nicht sicher ändern, ohne jedes Element in Ihrer Tabelle erneut zu verschlüsseln.

Signaturvalidierungsfehler können extrem schwierig zu beheben sein. Daher ist es am besten, diese zu verhindern.

Themen

Hinzufügen eines Attributs

Wenn Sie ein neues Attribut zu Tabellenelementen hinzufügen, müssen Sie möglicherweise die Attributaktionen ändern. Um Signaturvalidierungsfehler zu vermeiden, empfehlen wir, diese Änderung in einem zweistufigen Prozess zu implementieren. Stellen Sie sicher, dass die erste Stufe abgeschlossen ist, bevor Sie mit der zweiten Stufe beginnen.

1. Ändern Sie die Attributaktionen in allen Anwendungen, die aus der Tabelle lesen oder in sie schreiben. Stellen Sie diese Änderungen bereit und bestätigen Sie, dass das Update an alle Zielhosts weitergegeben wurde.
2. Schreiben Sie Werte in das neue Attribut in Ihren Tabellenelementen.

Dieser zweistufige Ansatz stellt sicher, dass alle Anwendungen und Hosts dieselben Attributaktionen haben, und berechnet die gleiche Signatur, bevor das neue Attribut auftritt. Dies ist auch dann wichtig, wenn die Aktion für das Attribut Nichts tun (nicht verschlüsseln oder signieren) lautet, da die Standardeinstellung für einige Verschlüssler das Verschlüsseln und Signieren ist.

Die folgenden Beispiele zeigen den Code für die erste Stufe in diesem Prozess. Sie fügen ein neues Elementattribut, `link`, hinzu, das einen Link zu einem anderen Tabellenelement speichert. Da dieser Link als Klartext verbleiben muss, wird ihm im Beispiel die Aktion nur zum Signieren zugewiesen. Nachdem Sie diese Änderung vollständig bereitgestellt und anschließend überprüft haben, ob alle Anwendungen und Hosts über die neuen Attributaktionen verfügen, können Sie mit der Verwendung des `link`-Attributs in Ihren Tabellenelementen beginnen.

Java DynamoDB Mapper

> Wenn der `DynamoDB Mapper` und `AttributeEncryptor` verwendet werden, sind standardmäßig alle Attribute verschlüsselt und signiert, mit Ausnahme der Primärschlüssel, die zwar signiert, aber nicht verschlüsselt sind. Verwenden Sie die Annotation `@DoNotEncrypt`, um eine Aktion nur mit Signierung anzugeben.

In diesem Beispiel wird die Annotation `@DoNotEncrypt` für das neue `link`-Attribut verwendet.

```
@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

Im DynamoDB-Verschlüsseler auf niedrigerer Ebene müssen Sie Aktionen für jedes Attribut festlegen. In diesem Beispiel wird eine Switch-Anweisung verwendet, bei der der Standardwert `encryptAndSign` lautet und Ausnahmen für den Partitionsschlüssel, den Sortierschlüssel und das neue `link`-Attribut angegeben werden. Wenn in diesem Beispiel der Linkattributcode nicht vollständig bereitgestellt wurde, bevor er verwendet wird, wird das Linkattribut von einigen Anwendungen verschlüsselt und signiert, von anderen aber nur signiert.

```
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

Im DynamoDB Encryption Client für Python können Sie eine Standardaktion für alle Attribute und dann Ausnahmen angeben.

Wenn Sie eine Python-Client-Helferklasse verwenden, müssen Sie keine Attributaktion für die Primärschlüsselattribute angeben. Die Client-Helferklassen verhindern, dass Sie Ihren Primärschlüssel verschlüsseln. Wenn Sie jedoch keine Client-Hilfsklasse verwenden, müssen Sie die Aktion SIGN_ONLY für Ihren Partitions- und Sortierschlüssel festlegen. Wenn Sie versehentlich Ihren Partitions- oder Sortierschlüssel verschlüsseln, können Sie Ihre Daten nur mit einem vollständigen Tabellenscan wiederherstellen.

In diesem Beispiel wird eine Ausnahme für das neue `link`-Attribut angegeben, das die Aktion `SIGN_ONLY` abruft.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={
      'example': CryptoAction.DO_NOTHING,
      'link': CryptoAction.SIGN_ONLY
    }
)
```

Entfernen eines Attributs

Wenn Sie ein Attribut in Elementen, die mit dem DynamoDB Encryption Client verschlüsselt wurden, nicht mehr benötigen, können Sie die Verwendung des Attributs beenden. Löschen oder ändern Sie die Aktion für dieses Attribut jedoch nicht. Wenn Sie in diesem Fall auf ein Element mit diesem Attribut stoßen, stimmt die für das Element berechnete Signatur nicht mit der ursprünglichen Signatur überein, und die Signaturvalidierung schlägt fehl.

Obwohl Sie möglicherweise versucht sind, alle Spuren des Attributs aus Ihrem Code zu entfernen, fügen Sie einen Kommentar hinzu, dass das Element nicht mehr verwendet wird, anstatt es zu löschen. Selbst wenn Sie einen vollständigen Tabellenscan durchführen, um alle Instances des Attributs zu löschen, wird möglicherweise ein verschlüsseltes Element mit diesem Attribut zwischengespeichert oder befindet sich irgendwo in Ihrer Konfiguration in Bearbeitung.

Behebung von Problemen in Ihrer DynamoDB Encryption Client-Anwendung

> **Note**
>
> Unsere clientseitige Verschlüsselungsbibliothek wurde in Database Encryption SDK umbenannt. AWS Das folgende Thema enthält Informationen zu Versionen 1. x —2. x des DynamoDB Encryption Client für Java und Versionen 1. x —3. x des DynamoDB Encryption Client für Python. Weitere Informationen finden Sie unter AWS Database Encryption SDK für DynamoDB-Versionsunterstützung.

In diesem Abschnitt werden Probleme beschrieben, die bei der Verwendung des DynamoDB Encryption Client auftreten können, und es werden Lösungsvorschläge gegeben.

Um Feedback zum DynamoDB Encryption Client zu geben, melden Sie ein Problem im aws-dynamodb-encryption-javaaws-dynamodb-encryption-python GitHub OR-Repository.

Über den auf jeder Seite angezeigten Feedback-Link können Sie Feedback zu dieser Dokumentation bereitstellen.

Themen

Zugriff verweigert

Problem: Ihrer Anwendung wird der Zugriff auf eine benötigte Ressource verweigert.

Vorschlag: Informieren Sie sich über die erforderlichen Berechtigungen und fügen Sie sie dem Sicherheitskontext hinzu, in dem Ihre Anwendung ausgeführt wird.

Details

Um eine Anwendung auszuführen, die die DynamoDB Encryption Client-Bibliothek verwendet, muss der Aufrufer berechtigt sein, ihre Komponenten zu verwenden. Andernfalls wird ihnen der Zugriff auf die benötigten Elemente verweigert.

- Der DynamoDB Encryption Client benötigt kein Amazon Web Services (AWS) -Konto und ist auch nicht von einem Service abhängig. AWS Wenn Ihre Anwendung jedoch verwendet AWS, benötigen Sie ein AWS-Konto und Benutzer, die berechtigt sind, das Konto zu verwenden.
- Der DynamoDB Encryption Client benötigt Amazon DynamoDB nicht. Wenn die Anwendung, die den Client verwendet, jedoch DynamoDB-Tabellen erstellt, Elemente in eine Tabelle einfügt oder Elemente aus einer Tabelle abruft, muss der Aufrufer über die Berechtigung verfügen, die erforderlichen DynamoDB-Operationen in Ihrer zu verwenden. AWS-Konto Einzelheiten finden Sie in den Themen zur Zugriffskontrolle im Amazon DynamoDB Developer Guide.
- Wenn Ihre Anwendung eine Client-Hilfsklasse im DynamoDB Encryption Client für Python verwendet, muss der Aufrufer berechtigt sein, den DynamoDB-Vorgang aufzurufen. DescribeTable

- Der DynamoDB Encryption Client benötigt AWS Key Management Service ()AWS KMS nicht. Wenn Ihre Anwendung jedoch einen Direct KMS Materials Provider oder einen Aktuellsten Anbieter mit einem Provider-Store verwendet, der diese verwendet AWS KMS, benötigt der Aufrufer die Erlaubnis, die Operationen AWS KMSGenerateDataKeyund Decrypt zu verwenden.

Signaturverifizierung schlägt fehl

Problem: Ein Element kann nicht entschlüsselt werden, da die Signaturprüfung fehlschlägt. Das Element ist möglicherweise auch nicht so verschlüsselt und signiert, wie von Ihnen beabsichtigt.

Vorschlag: Stellen Sie sicher, dass die Attribut-Aktionen, die Sie zur Verfügung stellen, alle Attribute des Elements berücksichtigen. Wenn Sie ein Element entschlüsseln, stellen Sie sicher, dass die Attributaktionen mit den Aktionen übereinstimmen, die zum Verschlüsseln des Elements verwendet wurden.

Details

Die von Ihnen bereitgestellten Attributaktionen teilen dem DynamoDB Encryption Client mit, welche Attribute verschlüsselt und signiert, welche Attribute signiert (aber nicht verschlüsselt) und welche ignoriert werden sollen.

Wenn die von Ihnen angegebenen Attribut-Aktionen nicht alle Attribute im Element berücksichtigen, wird das Element möglicherweise nicht so verschlüsselt und signiert, wie Sie es beabsichtigen. Wenn die Attribut-Aktionen, die Sie beim Entschlüsseln eines Elements angeben, von den Attribut-Aktionen abweichen, die Sie beim Verschlüsseln des Elements angegeben haben, kann die Signaturprüfung fehlschlagen. Dies ist speziell ein Problem für verteilte Anwendungen, bei denen sich neue Attribut-Aktionen nicht auf alle Hosts ausgebreitet haben.

Signaturvalidierungsfehler sind schwer zu beheben. Um sie zu verhindern, sollten Sie zusätzliche Vorsichtsmaßnahmen ergreifen, wenn Sie Ihr Datenmodell ändern. Details hierzu finden Sie unter Ändern Ihres Datenmodells.

Probleme mit globalen Tabellen älterer Versionen

Problem: Elemente in einer älteren Version der globalen Amazon DynamoDB-Tabelle können nicht entschlüsselt werden, da die Signaturüberprüfung fehlschlägt.

Vorschlag: Richten Sie Attributaktionen so ein, dass die reservierten Replikationsfelder nicht verschlüsselt oder signiert werden.

Details

Sie können den DynamoDB Encryption Client mit globalen DynamoDB-Tabellen verwenden. Wir empfehlen, globale Tabellen mit einem KMS-Schlüssel für mehrere Regionen zu verwenden und den KMS-Schlüssel in alle Bereiche zu replizieren, in AWS-Regionen denen die globale Tabelle repliziert wird.

Ab Version 2019.11.21 für globale Tabellen können Sie globale Tabellen mit dem DynamoDB Encryption Client ohne spezielle Konfiguration verwenden. Wenn Sie jedoch die Version 2017.11.29 für globale Tabellen verwenden, müssen Sie sicherstellen, dass reservierte Replikationsfelder nicht verschlüsselt oder signiert sind.

Wenn Sie die globale Tabellenversion 2017.11.29 verwenden, müssen Sie die Attributaktionen für die folgenden Attribute `DO_NOTHING` in Java oder `@DoNotTouch` Python auf setzen.

- `aws:rep:deleting`
- `aws:rep:updatetime`
- `aws:rep:updateregion`

Wenn Sie eine andere Version von globalen Tabellen verwenden, ist keine Aktion erforderlich.

Schlechte Leistung des neuesten Anbieters

Problem: Ihre Anwendung reagiert weniger, insbesondere nach einem Update auf eine neuere Version des DynamoDB Encryption Client.

Vorschlag: Passen Sie den time-to-live Wert und die Cachegröße an.

Details

The Most Recent Provider wurde entwickelt, um die Leistung von Anwendungen zu verbessern, die den DynamoDB Encryption Client verwenden, indem eine eingeschränkte Wiederverwendung von kryptografischem Material ermöglicht wird. Wenn Sie den neuesten Anbieter für Ihre Anwendung konfigurieren, müssen Sie die verbesserte Leistung mit den Sicherheitsbedenken abwägen, die sich aus dem Zwischenspeichern und der Wiederverwendung ergeben.

In neueren Versionen des DynamoDB Encryption Client bestimmt der time-to-live (TTL) -Wert, wie lange Anbieter von zwischengespeichertem kryptografischem Material (CMPs) verwendet werden können. Die TTL bestimmt auch, wie oft der neueste Anbieter nach einer neuen Version der CMP sucht.

Wenn Ihre TTL zu lang ist, verstößt Ihre Anwendung möglicherweise gegen Ihre Geschäftsregeln oder Sicherheitsstandards. Wenn Ihre TTL zu kurz ist, können häufige Anrufe beim Provider Store dazu führen, dass Ihr Provider Store Anfragen von Ihrer Anwendung und anderen Anwendungen, die Ihr Dienstkonto gemeinsam nutzen, drosselt. Um dieses Problem zu beheben, passen Sie TTL und Cachegröße auf einen Wert an, der Ihren Latenz- und Verfügbarkeitszielen entspricht und Ihren Sicherheitsstandards entspricht. Details hierzu finden Sie unter Es wird ein Wert für die Gültigkeitsdauer festgelegt.

Amazon DynamoDB Encryption Client umbenennen

Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Am 9. Juni 2023 wurde unsere clientseitige Verschlüsselungsbibliothek in Database Encryption SDK umbenannt. AWS Das AWS Database Encryption SDK ist mit Amazon DynamoDB kompatibel. Es kann Elemente entschlüsseln und lesen, die mit dem älteren DynamoDB Encryption Client verschlüsselt wurden. Weitere Informationen zu den älteren Versionen des DynamoDB Encryption Client finden Sie unter. AWS Database Encryption SDK für DynamoDB-Versionsunterstützung

Das AWS Database Encryption SDK bietet Version 3. x und höher der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB. Version 3. x ist eine umfassende Neufassung des DynamoDB Encryption Client für Java. Er beinhaltet viele Updates, wie z. B. ein neues strukturiertes Datenformat, verbesserte Mehrmandantenunterstützung, nahtlose Schemaänderungen und Unterstützung für durchsuchbare Verschlüsselung.

Weitere Informationen zu den neuen Funktionen, die mit dem AWS Database Encryption SDK eingeführt wurden, finden Sie in den folgenden Themen.

Durchsuchbare Verschlüsselung

Sie können Datenbanken entwerfen, die verschlüsselte Datensätze durchsuchen können, ohne die gesamte Datenbank zu entschlüsseln. Abhängig von Ihrem Bedrohungsmodell und Ihren Abfrageanforderungen können Sie eine durchsuchbare Verschlüsselung verwenden, um nach exakten Treffern oder individuellere komplexe Abfragen in Ihren verschlüsselten Datensätzen durchzuführen.

Schlüsselanhänger

Das AWS Database Encryption SDK verwendet Schlüsselringe, um die Envelope-Verschlüsselung durchzuführen. Schlüsselringe generieren, verschlüsseln und entschlüsseln die Datenschlüssel, die Ihre Daten schützen. Das AWS Database Encryption SDK unterstützt AWS KMS Schlüsselbunde, die symmetrische Verschlüsselung oder asymmetrische RSA verwenden, um Ihre Datenschlüssel AWS KMS keyszu schützen, sowie AWS KMS hierarchische Schlüsselringe, mit denen Sie Ihre kryptografischen Materialien mit einem symmetrischen

Verschlüsselungs-KMS-Schlüssel schützen können, ohne jedes Mal, wenn Sie einen Datensatz verschlüsseln oder entschlüsseln, erneut aufrufen zu müssen. AWS KMS Sie können mit Raw AES Keyrings und Raw RSA Keyrings auch Ihr eigenes Schlüsselmaterial angeben.

Reibungslose Schemaänderungen

Wenn Sie das AWS Database Encryption SDK konfigurieren, stellen Sie kryptografische Aktionen bereit, die dem Client mitteilen, welche Felder verschlüsselt und signiert, welche Felder signiert (aber nicht verschlüsselt) und welche ignoriert werden sollen. Nachdem Sie das AWS Database Encryption SDK zum Schutz Ihrer Datensätze verwendet haben, können Sie immer noch Änderungen an Ihrem Datenmodell vornehmen. Sie können Ihre kryptografischen Aktionen, wie das Hinzufügen oder Entfernen verschlüsselter Felder, in einer einzigen Bereitstellung aktualisieren.

Konfiguration vorhandener DynamoDB-Tabellen für clientseitige Verschlüsselung

Ältere Versionen des DynamoDB Encryption Client wurden für die Implementierung in neuen, nicht aufgefüllten Tabellen konzipiert. Mit dem AWS Database Encryption SDK für DynamoDB können Sie Ihre vorhandenen Amazon DynamoDB-Tabellen auf Version 3 migrieren. x oder 4. x der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB.

Referenz

Unsere clientseitige Verschlüsselungsbibliothek wurde in AWS Database Encryption SDK umbenannt. Dieses Entwicklerhandbuch enthält weiterhin Informationen zum DynamoDB Encryption Client.

Die folgenden Themen enthalten technische Details zum AWS Database Encryption SDK.

Format der Materialbeschreibung

Die Materialbeschreibung dient als Header für einen verschlüsselten Datensatz. Wenn Sie Felder mit dem AWS Database Encryption SDK verschlüsseln und signieren, zeichnet der Verschlüsseler die Materialbeschreibung auf, während er die kryptografischen Materialien zusammenstellt, und speichert die Materialbeschreibung in einem neuen Feld (`aws_dbe_head`), das der Verschlüsseler Ihrem Datensatz hinzufügt. Die Materialbeschreibung ist eine übertragbare, formatierte Datenstruktur, die den verschlüsselten Datenschlüssel und Informationen darüber enthält, wie der Datensatz verschlüsselt und signiert wurde. In der folgenden Tabelle werden die Werte beschrieben, aus denen sich die Materialbeschreibung zusammensetzt. Die Byte werden in der angegebenen Reihenfolge angehängt.

Wert	Länge in Byte
Version	1
Signatures Enabled	1
Record ID	32
Encrypt Legend	Variable
Encryption Context Length	2
???	Variable
Encrypted Data Key Count	1
Encrypted Data Keys	Variable

Wert	Länge in Byte
Record Commitment	1

Version

Die Version des Formats dieses `aws_dbe_head` Felds.

Signaturen aktiviert

Kodiert, ob digitale ECDSA-Signaturen für diesen Datensatz aktiviert sind.

Byte-Wert	Bedeutung
`0x01`	Digitale ECDSA-Signaturen sind aktiviert (Standard)
`0x00`	Digitale ECDSA-Signaturen sind deaktiviert

Datensatz-ID

Ein zufällig generierter 256-Bit-Wert, der den Datensatz identifiziert. Die Datensatz-ID:

- Identifiziert den verschlüsselten Datensatz eindeutig.
- Bindet die Materialbeschreibung an den verschlüsselten Datensatz.

Legende verschlüsseln

Eine serialisierte Beschreibung, welche authentifizierten Felder verschlüsselt wurden. Die Verschlüsselungslegende wird verwendet, um zu bestimmen, welche Felder die Entschlüsselungsmethode zu entschlüsseln versuchen soll.

Byte-Wert	Bedeutung
`0x65`	`ENCRYPT_AND_SIGN`
`0x73`	`SIGN_ONLY`

Die Encrypt-Legende ist wie folgt serialisiert:

1. Lexikographisch nach der Bytefolge, die ihren kanonischen Pfad darstellt.
2. Hängen Sie für jedes Feld der Reihe nach einen der oben angegebenen Bytewerte an, um anzugeben, ob das Feld verschlüsselt werden soll.

Länge des Verschlüsselungskontextes

Die Länge des Verschlüsselungskontextes. Dies ist ein 2-Byte-Wert, interpretiert als vorzeichenlose 16-Bit-Ganzzahl. Die maximale Länge beträgt 65.535 Byte.

Verschlüsselungskontext

Ein Satz von Name-Wert-Paaren, die beliebige, nicht geheime zusätzliche authentifizierte Daten enthalten.

Wenn digitale ECDSA-Signaturen aktiviert sind, enthält der Verschlüsselungskontext das Schlüssel-Wert-Paar. `{"aws-crypto-footer-ecdsa-key": Qtxt}` `Qtxt`stellt den elliptischen Kurvenpunkt dar, der gemäß SEC 1 Version 0 2.0 komprimiert und dann Base64-kodiert wurde.

Anzahl verschlüsselter Datenschlüssel

Die Anzahl der verschlüsselten Datenschlüssel. Es handelt sich um einen 1-Byte-Wert, der als 8-Bit-Ganzzahl ohne Vorzeichen interpretiert wird und die Anzahl der verschlüsselten Datenschlüssel angibt. Die maximale Anzahl verschlüsselter Datenschlüssel in jedem Datensatz beträgt 255.

Verschlüsselte Datenschlüssel

Eine Folge von verschlüsselten Datenschlüsseln. Die Länge der Folge wird durch die Anzahl der verschlüsselten Datenschlüssel und ihre jeweilige Länge bestimmt. Die Folge enthält mindestens einen verschlüsselten Datenschlüssel.

In der folgenden Tabelle sind die Felder beschrieben, die die verschlüsselten Datenschlüssel bilden. Die Byte werden in der angegebenen Reihenfolge angehängt.

Struktur der verschlüsselten Datenschlüssel

Feld	Länge in Byte
Key Provider ID Length	2

Feld	Länge in Byte
Key Provider ID	Variable. Gleich dem Wert, der in den vorherigen 2 Bytes angegeben ist (Länge der Schlüsselanbieter-ID).
Key Provider Information Length	2
Key Provider Information	Variable. Gleich dem Wert, der in den vorherigen 2 Bytes angegeben ist (Länge der Schlüsselanbieterinformation).
Encrypted Data Key Length	2
Encrypted Data Key	Variable. Gleich dem Wert, der in den vorherigen 2 Bytes angegeben ist (Länge des verschlüsselten Datenschlüssels).

Länge der Schlüsselanbieter-ID

Die Länge der Schlüsselanbieter-ID. Es handelt sich um einen 2-Byte-Wert, interpretiert als vorzeichenlose 16-Bit-Ganzzahl, die die Anzahl der Bytes angibt, die die Schlüsselanbieter-ID enthalten.

ID des Schlüsselanbieters

Die Schlüsselanbieter-ID. Wird verwendet, um den Anbieter des verschlüsselten Datenschlüssels anzugeben, und ist auf Erweiterbarkeit ausgelegt.

Länge der Informationen zum Schlüsselanbieter

Die Länge der Schlüsselanbieterinformation. Es handelt sich um einen 2-Byte-Wert, interpretiert als vorzeichenlose 16-Bit-Ganzzahl, die die Anzahl der Bytes angibt, die die Schlüsselanbieterinformation enthalten.

Informationen zu den wichtigsten Anbietern

Die Schlüsselanbieterinformation. Wird durch den Schlüsselanbieter bestimmt.

Wenn Sie einen AWS KMS Schlüsselbund verwenden, enthält dieser Wert den Amazon-Ressourcennamen (ARN) von. AWS KMS key

Länge des verschlüsselten Datenschlüssels

Die Länge des verschlüsselten Datenschlüssels. Es handelt sich um einen 2-Byte-Wert, interpretiert als vorzeichenlose 16-Bit-Ganzzahl, die die Anzahl der Bytes angibt, die den verschlüsselten Datenschlüssel enthalten.

Verschlüsselter Datenschlüssel

Der verschlüsselte Datenschlüssel. Es ist der vom Schlüsselanbieter verschlüsselte Datenschlüssel.

Engagement in Rekordhöhe

Ein eindeutiger 256-Bit-HMAC-Hash (Hash-Based Message Authentication Code), der mithilfe des Commit-Schlüssels für alle vorherigen Materialbeschreibungs-Bytes berechnet wurde.

AWS KMS Technische Details zum hierarchischen Schlüsselbund

Der AWS KMS hierarchische Schlüsselbund verwendet einen eindeutigen Datenschlüssel, um jedes Feld zu verschlüsseln, und verschlüsselt jeden Datenschlüssel mit einem eindeutigen Umschließungsschlüssel, der von einem aktiven Zweigschlüssel abgeleitet wird. Er verwendet eine Schlüsselableitung im Zählermodus mit einer Pseudozufallsfunktion mit HMAC SHA-256, um den 32-Byte-Wrapping-Schlüssel mit den folgenden Eingaben abzuleiten.

- Ein zufälliges 16-Byte-Salz
- Der aktive Zweigschlüssel
- Der UTF-8-kodierte Wert für die Schlüsselanbieter-ID "“ aws-kms-hierarchy

Der hierarchische Schlüsselbund verwendet den abgeleiteten Wrapping-Schlüssel, um eine Kopie des Klartext-Datenschlüssels mithilfe von AES-GCM-256 mit einem 16-Byte-Authentifizierungs-Tag und den folgenden Eingaben zu verschlüsseln.

- Der abgeleitete Wrapping-Schlüssel wird als AES-GCM-Verschlüsselungsschlüssel verwendet
- Der Datenschlüssel wird als AES-GCM-Nachricht verwendet
- Ein zufälliger 12-Byte-Initialisierungsvektor (IV) wird als AES-GCM IV verwendet
- Zusätzliche authentifizierte Daten (AAD), die die folgenden serialisierten Werte enthalten.

Wert	Länge in Byte	Interpretiert als
"aws-kms-hierarchy"	17	UTF-8-Kodierung
Die Kennung des Zweigschlüssels	Variable	UTF-8-Kodierung
Die Version des Zweigschlüssels	16	UTF-8-Kodierung
Verschlüsselungskontext	Variable	UTF-8-kodierte Schlüssel-Wert-Paare

Dokumentenhistorie für AWS Entwicklerhandbuch für das Database Encryption SDK

In der folgenden Tabelle werden wichtige Änderungen an dieser Dokumentation beschrieben. Neben diesen hier aufgelisteten größeren Änderungen aktualisieren wir die Dokumentation regelmäßig überarbeitet, um Beschreibungen und Beispiele zu verbessern und Ihre Rückmeldungen zu berücksichtigen. Wenn Sie über wichtige Änderungen benachrichtigt werden möchten, abonnieren Sie den RSS-Feed.

Änderung	Beschreibung	Datum
Version für allgemeine Verfügbarkeit (GA)	Einführung der 4.x-Version der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB, die das AWS SDK for Java 1.x durch das AWS SDK for Java 2.x in der eingebetteten clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB-Version 2.x-Code ersetzt.	26. März 2026
Neues Feature	Dokumentation für den AWS KMS ECDH-Keyring und den Raw ECDH-Keyring hinzugefügt.	17. Juni 2024
Version für allgemeine Verfügbarkeit (GA)	Einführung in die Unterstützung der clientseitigen .NET-Verschlüsselungsbibliothek für DynamoDB.	17. Januar 2024
Version für allgemeine Verfügbarkeit (GA)	Die Dokumentation für die GA-Version von Version 3 wurde aktualisiert. x der clientseitigen	24. Juli 2023

	Java-Verschlüsselungsbiblio thek für DynamoDB. **Warning** Verzweigungsschlüs sel, die während der Developer Preview-V ersion erstellt wurden, werden nicht mehr unterstützt.	
Umbenennung von DynamoDB Encryption Client	Die clientseitige Verschlüs selungsbibliothek wurde in Database Encryption SDK umbenannt. AWS	9. Juni 2023
Vorschauversion	Dokumentation für Version 3 hinzugefügt und aktualisi ert. x der clientseitigen Java-Verschlüsselungsbibliothek für DynamoDB, die ein neues strukturiertes Datenformat, verbesserte Mehrmanda ntenunterstützung, nahtlose Schemaänderungen und Unterstützung für durchsuch bare Verschlüsselung umfasst.	9. Juni 2023
Änderung der Dokumentation	Ersetzen Sie den AWS Key Management Service Begriff Customer Master Key (CMK) durch einen KMS-Schlüssel. AWS KMS key	30. August 2021

Neue Funktion	Unterstützung für AWS Key Management Service (AWS KMS) Schlüssel mit mehreren Regionen hinzugefügt. Multi-Region Schlüssel sind unterschiedliche AWS KMS Schlüssel AWS-Regionen , die synonym verwendet werden können, da sie dieselbe Schlüssel-ID und dasselbe Schlüsselmaterial haben.	8. Juni 2021
Neues Beispiel	Beispiel für die Verwendung des DynamoDBMapper in Java hinzugefügt.	6. September 2018
Python-Unterstützung	Unterstützung für Python zusätzlich zu Java hinzugefügt.	2. Mai 2018
Erstversion	Erste Veröffentlichung dieser Dokumentation.	2. Mai 2018

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.