aws

Guida per gli sviluppatori

AWS SDK per la crittografia del database

AWS SDK per la crittografia del database: Guida per gli sviluppatori

Table of Contents

Cos'è il AWS Database Encryption SDK?

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Il AWS Database Encryption SDK è un insieme di librerie software che consentono di includere la crittografia lato client nella progettazione del database. Il AWS Database Encryption SDK fornisce soluzioni di crittografia a livello di record. Specifica quali campi sono crittografati e quali sono inclusi nelle firme che garantiscono l'autenticità dei tuoi dati. La crittografia dei dati sensibili in transito e a riposo aiuta a garantire che i dati in testo non crittografato non siano disponibili a terzi, ad esempio. AWS Il AWS Database Encryption SDK è fornito gratuitamente con la licenza Apache 2.0.

Questa guida per gli sviluppatori fornisce una panoramica concettuale del AWS Database Encryption SDK, inclusa un'introduzione alla sua architettura, dettagli su come protegge i dati, su come si differenzia dalla crittografia lato server e indicazioni sulla selezione dei componenti critici per l'applicazione per aiutarti a iniziare.

L'SDK per la crittografia del AWS database supporta Amazon DynamoDB con crittografia a livello di attributo. Versione 3. x della libreria di crittografia lato client Java per DynamoDB è un'importante riscrittura del client di crittografia DynamoDB per Java. Include molti aggiornamenti, come un nuovo formato di dati strutturati, supporto multitenancy migliorato, crittografia ricercabile e supporto per modifiche allo schema senza interruzioni.

Il AWS Database Encryption SDK offre i seguenti vantaggi:

Progettato appositamente per le applicazioni di database

- Non è necessario essere un esperto di crittografia per utilizzare il AWS Database Encryption SDK. Le implementazioni includono metodi di supporto progettati per funzionare con le applicazioni esistenti.

- Dopo aver creato e configurato i componenti richiesti, il client di crittografia crittografa e firma in modo trasparente i record quando li aggiungi a un database e li verifica e li decrittografa quando li recuperi.

Include la funzione di firma e crittografia sicure

Il AWS Database Encryption SDK include implementazioni sicure che crittografano i valori dei campi in ogni record utilizzando una chiave di crittografia dei dati univoca e quindi firmano il record per proteggerlo da modifiche non autorizzate, come l'aggiunta o l'eliminazione di campi o lo scambio di valori crittografati.

Utilizza i materiali crittografici provenienti da qualsiasi origine

Il AWS Database Encryption SDK utilizza i portachiavi per generare, crittografare e decrittografare la chiave di crittografia dei dati univoca che protegge il tuo record. I portachiavi determinano le chiavi di avvolgimento che crittografano quella chiave dati.

Puoi utilizzare chiavi di wrapping da qualsiasi fonte, inclusi i servizi di crittografia, come AWS Key Management Service() o. AWS KMS AWS CloudHSM Il AWS Database Encryption SDK non richiede Account AWS alcun AWS servizio.

Supporto per la memorizzazione nella cache di materiali crittografici

Il portachiavi AWS KMS gerarchico è una soluzione di memorizzazione nella cache dei materiali crittografici che riduce il numero di AWS KMS chiamate utilizzando chiavi di derivazione AWS KMS protette conservate in una tabella Amazon DynamoDB e quindi memorizzando nella cache locale i materiali delle chiavi di diramazione utilizzati nelle operazioni di crittografia e decrittografia. Ti consente di proteggere i tuoi materiali crittografici con una chiave KMS di crittografia simmetrica senza chiamare AWS KMS ogni volta che crittografi o decrittografi un record. Il portachiavi AWS KMS gerarchico è una buona scelta per le applicazioni che devono ridurre al minimo le chiamate a. AWS KMS

Crittografia ricercabile

È possibile progettare database in grado di cercare record crittografati senza decrittografare l'intero database. A seconda del modello di minaccia e dei requisiti delle query, puoi utilizzare la crittografia ricercabile per eseguire ricerche con corrispondenza esatta o query complesse più personalizzate sul tuo database crittografato.

Supporto per schemi di database multitenant

Il AWS Database Encryption SDK consente di proteggere i dati archiviati nei database con uno schema condiviso isolando ogni tenant con materiali di crittografia distinti. Se hai più utenti che eseguono operazioni di crittografia all'interno del tuo database, utilizza uno dei AWS KMS portachiavi per fornire a ciascun utente una chiave distinta da utilizzare nelle proprie operazioni crittografiche. Per ulteriori informazioni, consulta Lavorare con database multitenant.

Supporto per aggiornamenti dello schema senza interruzioni

Quando configuri AWS Database Encryption SDK, fornisci azioni crittografiche che indicano al client quali campi crittografare e firmare, quali firmare (ma non crittografare) e quali ignorare. Dopo aver utilizzato AWS Database Encryption SDK per proteggere i tuoi record, puoi comunque apportare modifiche al tuo modello di dati. Puoi aggiornare le tue azioni crittografiche, come l'aggiunta o la rimozione di campi crittografati, in un'unica distribuzione.

Sviluppato in repository open source

Il AWS Database Encryption SDK è sviluppato in repository open source su. GitHub Puoi utilizzare questi repository per visualizzare il codice, leggere e inviare problemi e trovare informazioni specifiche per la tua implementazione.

L'SDK per la crittografia del AWS database per DynamoDB

- libreria di crittografia Java lato client per DynamoDB — -dynamodb-java aws-database-encryption-sdk

  Versione 3. x della libreria di crittografia lato client Java per DynamoDB è un prodotto del AWS Database Encryption SDK in Dafny, un linguaggio che supporta la verifica in cui si scrivono le specifiche, il codice per implementarle e le prove per testarle. Il risultato è una libreria che implementa le funzionalità del AWS Database Encryption SDK per DynamoDB in un framework che garantisce la correttezza funzionale.

Supporto e manutenzione

AWSDatabase Encryption SDK utilizza la stessa politica di manutenzione utilizzata da AWS SDK e Tools, comprese le fasi di controllo delle versioni e del ciclo di vita. Come best practice, ti consigliamo di utilizzare l'ultima versione disponibile di AWS Database Encryption SDK per l'implementazione del database e di effettuare l'aggiornamento man mano che vengono rilasciate nuove versioni.

Per ulteriori informazioni, consulta la politica di manutenzione degli AWS SDK e degli strumenti nella Guida di riferimento agli AWS SDK e agli strumenti.

Invio di feedback

Apprezziamo il tuo feedback. Se hai una domanda, un commento o un problema da segnalare, utilizza le seguenti risorse.

Se scopri una potenziale vulnerabilità di sicurezza nel AWS Database Encryption SDK, invia una notifica alla AWS sicurezza. Non creare un GitHub problema pubblico.

Per fornire feedback su questa documentazione, utilizzare il link di feedback in qualsiasi pagina.

AWS Concetti dell'SDK per la crittografia dei database

> La nostra libreria di crittografia lato client è stata rinominata Database Encryption SDK. AWS Questa guida per sviluppatori fornisce ancora informazioni sul DynamoDB Encryption Client.

Questo argomento spiega i concetti e la terminologia utilizzati nel AWS Database Encryption SDK.

Per informazioni su come interagiscono i componenti del AWS Database Encryption SDK, consulta. Come funziona il AWS Database Encryption SDK

Per ulteriori informazioni su AWS Database Encryption SDK, consulta i seguenti argomenti.

- Scopri come AWS Database Encryption SDK utilizza la crittografia a busta per proteggere i tuoi dati.
- Scopri gli elementi della crittografia in busta: le chiavi dati che proteggono i tuoi record e le chiavi di avvolgimento che proteggono le tue chiavi dati.
- Scopri i portachiavi che determinano le chiavi di avvolgimento da utilizzare.
- Scopri il contesto di crittografia che aggiunge integrità al tuo processo di crittografia.
- Scopri la descrizione del materiale che i metodi di crittografia aggiungono al tuo record.
- Scopri le azioni crittografiche che indicano al AWS Database Encryption SDK quali campi crittografare e firmare.

Argomenti

- Crittografia envelope
- Chiave di dati
- Chiave di avvolgimento

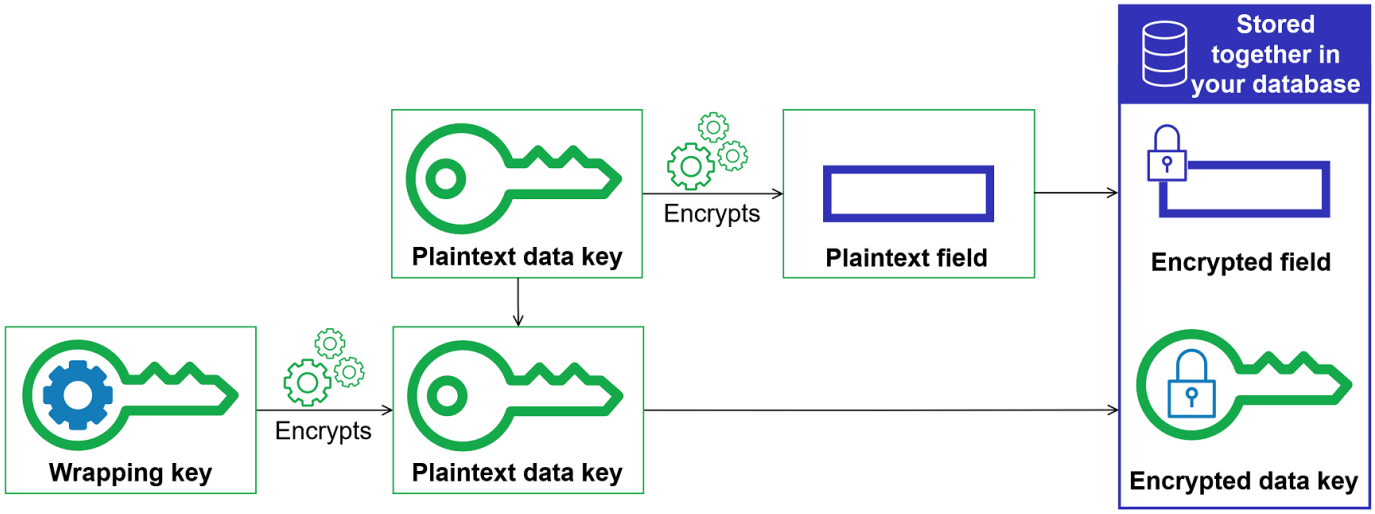
- Portachiavi
- Azioni crittografiche
- Descrizione dei materiali
- Contesto di crittografia
- Responsabile di materiali crittografici
- Crittografia simmetrica e asimmetrica
- Impegno chiave
- Firme digitali

Crittografia envelope

La sicurezza dei dati crittografati dipende in parte dalla protezione della chiave di dati che può decrittarli. Una best practice accettata per la protezione della chiave di dati è crittografarla. A tale scopo, è necessaria un'altra chiave di crittografia, nota come chiave di crittografia a chiave o chiave di wrapping. La pratica di utilizzare una chiave di wrapping per crittografare le chiavi di dati è nota come crittografia a busta.

Protezione delle chiavi dei dati

> Il AWS Database Encryption SDK crittografa ogni campo con una chiave dati unica. Quindi crittografa ogni chiave di dati sotto la chiave di wrapping specificata. Memorizza le chiavi di dati crittografate nella descrizione del materiale.
>
> Per specificare la chiave di imballaggio, si utilizza un portachiavi.



Crittografia degli stessi dati con più chiavi di wrapping

È possibile crittografare la chiave dati con più chiavi di wrapping. Potresti voler fornire chiavi di avvolgimento diverse per utenti diversi, oppure chiavi di avvolgimento di tipi diversi o in posizioni diverse. Ciascuna delle chiavi di wrapping crittografa la stessa chiave di dati. Il AWS Database Encryption SDK memorizza tutte le chiavi di dati crittografate insieme ai campi crittografati nella descrizione del materiale.

Per decrittografare i dati, è necessario fornire almeno una chiave di wrapping in grado di decrittografare le chiavi di dati crittografate.

Abbinare i punti di forza di più algoritmi

Per crittografare i dati, per impostazione predefinita, AWS Database Encryption SDK utilizza una suite di algoritmi con crittografia simmetrica AES-GCM, una funzione di derivazione delle chiavi basata su HMAC (HKDF) e firma ECDSA. Per crittografare la chiave dati, puoi specificare un algoritmo di crittografia simmetrico o asimmetrico appropriato alla tua chiave di wrapping.

In generale, gli algoritmi di crittografia di chiavi simmetriche sono più rapidi e producono testi cifrati di dimensioni minori rispetto alla crittografia della chiave pubblica o asimmetrica. Ma gli algoritmi a chiave pubblica forniscono una separazione intrinseca dei ruoli. Per combinare i punti di forza di ciascuno, puoi crittografare la chiave dati con la crittografia a chiave pubblica.

Ti consigliamo di utilizzare uno dei AWS KMS portachiavi ogni volta che è possibile. Quando usi il AWS KMS portachiavi, puoi scegliere di combinare i punti di forza di più algoritmi specificando un RSA asimmetrico come chiave di avvolgimento. AWS KMS key Puoi anche utilizzare una chiave KMS di crittografia simmetrica.

Chiave di dati

Una chiave dati è una chiave di crittografia che AWS Database Encryption SDK utilizza per crittografare i campi di un record contrassegnati `ENCRYPT_AND_SIGN` nelle azioni crittografiche. Ogni chiave di dati corrisponde a un array di byte conforme ai requisiti per le chiavi di crittografia. Il AWS Database Encryption SDK utilizza una chiave dati unica per crittografare ogni attributo.

Non è necessario specificare, generare, implementare, estendere, proteggere o utilizzare chiavi dati. Il AWS Database Encryption SDK funziona per te quando richiami le operazioni di crittografia e decrittografia.

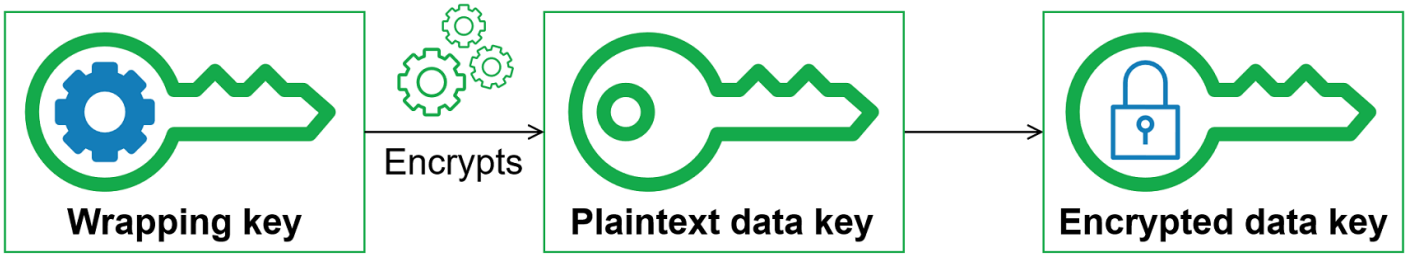
Per proteggere le chiavi dei dati, AWS Database Encryption SDK le crittografa utilizzando una o più chiavi di crittografia a chiave note come chiavi di wrapping. Dopo che AWS Database Encryption SDK ha utilizzato le chiavi di dati in testo semplice per crittografare i dati, le rimuove dalla memoria il prima possibile. Quindi memorizza la chiave di dati crittografata nella descrizione del materiale. Per informazioni dettagliate, vedi Come funziona il AWS Database Encryption SDK.

> **Tip**
>
> Nel AWS Database Encryption SDK, distinguiamo le chiavi di dati dalle chiavi di crittografia dei dati. Come best practice, tutte le suite di algoritmi supportate utilizzano una funzione di derivazione delle chiavi. La funzione di derivazione delle chiavi accetta una chiave di dati come input e restituisce le chiavi di crittografia dei dati che vengono effettivamente utilizzate per crittografare i record. Per questo motivo, abbiamo spesso detto che i dati sono crittografati "in" una chiave dei dati anziché "da" una chiave di dati.

Ogni chiave di dati crittografata include metadati, incluso l'identificatore della chiave di avvolgimento che l'ha crittografata. Questi metadati consentono al AWS Database Encryption SDK di identificare chiavi di wrapping valide durante la decrittografia.

Chiave di avvolgimento

Una chiave di wrapping è una chiave di crittografia a chiave che AWS Database Encryption SDK utilizza per crittografare la chiave dati che crittografa i record. Ogni chiave di dati può essere crittografata con una o più chiavi di wrapping. Sei tu a determinare quali chiavi di wrapping vengono utilizzate per proteggere i tuoi dati quando configuri un portachiavi.



AWS Database Encryption SDK supporta diverse chiavi di wrapping di uso comune, come AWS Key Management Service(AWS KMS) chiavi KMS con crittografia simmetrica (incluse chiavi multiregionali AWS KMS) e chiavi RSA KMS asimmetriche, chiavi AES-GCM (Advanced Encryption Standard/ Galois Counter Mode) non elaborate e chiavi RSA non elaborate. Ti consigliamo di utilizzare le chiavi

KMS ogni volta che è possibile. Per decidere quale chiave di avvolgimento utilizzare, vedi Selezione delle chiavi di avvolgimento.

Quando si utilizza la crittografia a busta, è necessario proteggere le chiavi di wrapping da accessi non autorizzati. È possibile eseguire questa operazione in uno dei seguenti modi:

- Utilizzate un servizio progettato per questo scopo, ad esempio AWS Key Management Service (AWS KMS).
- Utilizza un modulo di sicurezza hardware (HSM) come quelli offerti da AWS CloudHSM.
- Utilizzate altri strumenti e servizi di gestione delle chiavi.

Se non disponi di un sistema di gestione delle chiavi, ti consigliamo AWS KMS. Il AWS Database Encryption SDK si integra con AWS KMS per aiutarti a proteggere e utilizzare le tue chiavi di wrapping.

Portachiavi

Per specificare le chiavi di avvolgimento utilizzate per la crittografia e la decrittografia, si utilizza un portachiavi. È possibile utilizzare i portachiavi forniti da AWS Database Encryption SDK o progettare implementazioni personalizzate.

Un keyring genera, crittografa e decritta le chiavi di dati. Genera inoltre le chiavi MAC utilizzate per calcolare gli HMAC (Hash Based Message Authentication Codes) contenuti nella firma. Quando definisci un portachiavi, puoi specificare le chiavi di avvolgimento che crittografano le tue chiavi dati. La maggior parte dei portachiavi specifica almeno una chiave di avvolgimento o un servizio che fornisce e protegge le chiavi di avvolgimento. Durante la crittografia, AWS Database Encryption SDK utilizza tutte le chiavi di wrapping specificate nel portachiavi per crittografare la chiave dati. Per informazioni sulla scelta e l'utilizzo dei portachiavi definiti da AWS Database Encryption SDK, consulta Utilizzo dei portachiavi.

Azioni crittografiche

Le azioni crittografiche indicano al crittografo quali azioni eseguire su ogni campo di un record.

I valori delle azioni crittografiche possono essere uno dei seguenti:

- Crittografa e firma: crittografa il campo. Includi il campo crittografato nella firma.
- Solo firma: includi il campo nella firma.
- Non fare nulla: non crittografare o includere il campo nella firma.

Per qualsiasi campo in cui è possibile archiviare dati sensibili, utilizza Encrypt and sign. Per i valori della chiave primaria (ad esempio, una chiave di partizione e una chiave di ordinamento in una tabella DynamoDB), usa solo Sign. Non è necessario specificare azioni crittografiche per la descrizione del materiale. Il AWS Database Encryption SDK firma automaticamente il campo in cui è memorizzata la descrizione del materiale.

Scegliete con attenzione le vostre azioni crittografiche. In caso di dubbio, usa Encrypt and sign (Crittografa e firma). Dopo aver utilizzato AWS Database Encryption SDK per proteggere i record, non è possibile modificare un `SIGN_ONLY` campo `ENCRYPT_AND_SIGN` o esistente in o modificare l'azione crittografica assegnata a un campo esistente. `DO_NOTHING` `DO_NOTHING` Tuttavia, puoi comunque apportare altre modifiche al tuo modello di dati. Ad esempio, puoi aggiungere o rimuovere campi crittografati, in un'unica distribuzione.

Descrizione dei materiali

La descrizione del materiale funge da intestazione per un record crittografato. Quando crittografate e firmate i campi con il AWS Database Encryption SDK, il criptatore registra la descrizione del materiale mentre assembla i materiali crittografici e archivia la descrizione del materiale in un nuovo campo (`aws_dbe_head`) che il crittografo aggiunge al record.

La descrizione del materiale è una struttura di dati formattata portatile che contiene copie crittografate delle chiavi di dati e altre informazioni, come algoritmi di crittografia, contesto di crittografia e istruzioni di crittografia e firma. Il criptatore registra la descrizione del materiale mentre assembla i materiali crittografici per la crittografia e la firma. Successivamente, quando deve assemblare materiale crittografico per verificare e decrittografare un campo, utilizza la descrizione del materiale come guida.

La memorizzazione delle chiavi dati crittografate insieme al campo crittografato semplifica l'operazione di decrittografia ed evita la necessità di archiviare e gestire le chiavi dati crittografate indipendentemente dai dati crittografati.

Per informazioni tecniche sulla descrizione del materiale, vedere. Formato di descrizione del materiale

Contesto di crittografia

Per migliorare la sicurezza delle operazioni crittografiche, il AWS Database Encryption SDK include un contesto di crittografia in tutte le richieste di crittografia e firma di un record.

Un contesto di crittografia è un set di coppie nome-valore che contiene dati autenticati aggiuntivi arbitrari e non segreti. Il AWS Database Encryption SDK include il nome logico del database e i valori della chiave primaria (ad esempio, una chiave di partizione e una chiave di ordinamento in una tabella DynamoDB) nel contesto di crittografia. Quando si crittografa e si firma un campo, il contesto di crittografia viene associato crittograficamente al record crittografato in modo che lo stesso contesto di crittografia sia necessario per decrittografare il campo.

Se utilizzi un AWS KMS portachiavi, AWS Database Encryption SDK utilizza anche il contesto di crittografia per fornire dati autenticati aggiuntivi (AAD) nelle chiamate a cui effettua il portachiavi. AWS KMS

Ogni volta che si utilizza la suite di algoritmi predefinita, il gestore dei materiali crittografici (CMM) aggiunge una coppia nome-valore al contesto di crittografia che consiste in un nome riservato e un valore che rappresenta la chiave di `aws-crypto-public-key` verifica pubblica. La chiave di verifica pubblica è memorizzata nella descrizione del materiale.

Responsabile di materiali crittografici

Il gestore dei materiali crittografici (CMM) assembla i materiali crittografici utilizzati per crittografare, decrittografare e firmare i dati. Ogni volta che si utilizza la suite di algoritmi predefinita, i materiali crittografici includono chiavi di dati in chiaro e crittografate, chiavi di firma simmetriche e una chiave di firma asimmetrica. Non interagisci mai direttamente con la CMM. I metodi di crittografia e decrittazione lo gestiscono per te.

Poiché la CMM funge da collegamento tra il AWS Database Encryption SDK e un portachiavi, è il punto ideale per la personalizzazione e l'estensione, ad esempio il supporto per l'applicazione delle politiche. È possibile specificare esplicitamente una CMM, ma non è obbligatorio. Quando specificate un portachiavi, AWS Database Encryption SDK crea automaticamente una CMM predefinita. La CMM predefinita ottiene i materiali di crittografia o decrittografia dal portachiavi specificato. Ciò potrebbe comportare una chiamata a un servizio crittografico, come AWS Key Management Service (AWS KMS).

Crittografia simmetrica e asimmetrica

La crittografia simmetrica utilizza la stessa chiave per crittografare e decrittografare i dati.

La crittografia asimmetrica utilizza una coppia di chiavi di dati matematicamente correlata. Una chiave della coppia crittografa i dati; solo l'altra chiave della coppia può decrittografare i dati. Per i dettagli, consulta Algoritmi crittografici nella Guida ai servizi e agli strumenti di AWS crittografia.

Il AWS Database Encryption SDK utilizza la crittografia a busta. Crittografa i dati con una chiave dati simmetrica. Crittografa la chiave dati simmetrica con una o più chiavi di avvolgimento simmetriche o asimmetriche. Aggiunge una descrizione del materiale al record che include almeno una copia crittografata della chiave dati.

Crittografia dei dati (crittografia simmetrica)

Per crittografare i dati, AWS Database Encryption SDK utilizza una chiave dati simmetrica e una suite di algoritmi che include un algoritmo di crittografia simmetrica. Per decrittografare i dati, AWS Database Encryption SDK utilizza la stessa chiave di dati e la stessa suite di algoritmi.

Crittografia della chiave dati (crittografia simmetrica o asimmetrica)

Il portachiavi fornito per un'operazione di crittografia e decrittografia determina il modo in cui la chiave dati simmetrica viene crittografata e decrittografata. Puoi scegliere un portachiavi che utilizza la crittografia simmetrica, ad esempio un portachiavi con una chiave KMS di crittografia simmetrica, o uno che utilizza la crittografia asimmetrica, come un AWS KMS portachiavi con una chiave RSA KMS asimmetrica. AWS KMS

Impegno chiave

Il AWS Database Encryption SDK supporta Key Commitment (talvolta nota come robustezza), una proprietà di sicurezza che garantisce che ogni testo cifrato possa essere decrittografato solo in un singolo testo non crittografato. A tale scopo, key commit garantisce che solo la chiave dati che ha crittografato il record venga utilizzata per decrittografarlo. Il AWS Database Encryption SDK include un impegno fondamentale per tutte le operazioni di crittografia e decrittografia.

La maggior parte dei cifrari simmetrici moderni (incluso AES) crittografa il testo in chiaro con un'unica chiave segreta, come la chiave dati univoca utilizzata da AWS Database Encryption SDK per crittografare ogni campo di testo in chiaro contrassegnato in un record. `ENCRYPT_AND_SIGN` La decrittografia di questo record con la stessa chiave di dati restituisce un testo in chiaro identico all'originale. La decrittografia con una chiave diversa di solito non riesce. Sebbene difficile, è tecnicamente possibile decrittografare un testo cifrato con due chiavi diverse. In rari casi, è possibile trovare una chiave in grado di decrittografare parzialmente il testo cifrato in un testo semplice diverso, ma comunque comprensibile.

Il AWS Database Encryption SDK cripta sempre ogni attributo con un'unica chiave dati. Potrebbe crittografare quella chiave dati con più chiavi di wrapping, ma le chiavi di wrapping crittografano sempre la stessa chiave dati. Tuttavia, un record crittografato sofisticato creato manualmente

potrebbe effettivamente contenere diverse chiavi di dati, ognuna crittografata da una chiave di wrapping diversa. Ad esempio, se un utente decrittografa il record crittografato, restituisce 0x0 (falso) mentre un altro utente che decrittografa lo stesso record crittografato ottiene 0x1 (vero).

Per evitare questo scenario, AWS Database Encryption SDK include un impegno chiave durante la crittografia e la decrittografia. Il metodo di crittografia associa crittograficamente la chiave di dati univoca che ha prodotto il testo cifrato all'impegno chiave, un codice di autenticazione dei messaggi basato su hash (HMAC) calcolato sulla descrizione del materiale utilizzando una derivazione della chiave dati. Quindi memorizza l'impegno chiave nella descrizione del materiale. Quando decripta un record con l'impegno della chiave, AWS Database Encryption SDK verifica che la chiave dati sia l'unica chiave per quel record crittografato. Se la verifica della chiave dati non riesce, l'operazione di decrittografia ha esito negativo.

Firme digitali

Per garantire l'autenticità dei dati durante lo spostamento tra i sistemi, è possibile applicare una firma digitale al record. Le firme digitali sono sempre asimmetriche. La chiave privata viene utilizzata per creare la firma e aggiungerla al record originale. Il destinatario utilizza una chiave pubblica per verificare che il record non sia stato modificato dopo la firma. È consigliabile utilizzare le firme digitali se gli utenti che crittografano i dati e gli utenti che decifrano i dati non sono altrettanto affidabili.

AWS Database Encryption SDK crittografa i dati utilizzando un algoritmo di crittografia autenticato, AES-GCM, ma poiché AES-GCM utilizza chiavi simmetriche, chiunque sia in grado di decrittografare la chiave dati utilizzata per decrittografare il testo cifrato può anche creare manualmente un nuovo testo cifrato crittografato, causando potenziali problemi di sicurezza.

Per evitare questo problema, la suite di algoritmi predefinita aggiunge una firma Elliptic Curve Digital Signature Algorithm (ECDSA) ai record crittografati. La suite di algoritmi predefinita crittografa i campi del record contrassegnati `ENCRYPT_AND_SIGN` utilizzando un algoritmo di crittografia autenticato, AES-GCM. Quindi, calcola sia i codici di autenticazione dei messaggi basati su hash (HMAC) che le firme ECDSA asimmetriche sui campi del record contrassegnati con e. `ENCRYPT_AND_SIGN` `SIGN_ONLY` Il processo di decrittografia utilizza le firme per verificare che un utente autorizzato abbia crittografato il record.

Quando viene utilizzata la suite di algoritmi predefinita, AWS Database Encryption SDK genera una chiave privata temporanea e una coppia di chiavi pubbliche per ogni record crittografato. AWS Database Encryption SDK memorizza la chiave pubblica nella descrizione del materiale e scarta la chiave privata, e nessuno può creare un'altra firma che verifichi con la chiave pubblica. Poiché

l'algoritmo associa la chiave pubblica alla chiave dati crittografata come dati autenticati aggiuntivi nella descrizione del materiale, un utente che può solo decrittografare i record non può alterare la chiave pubblica.

Il AWS Database Encryption SDK include sempre la verifica HMAC. Le firme digitali ECDSA sono abilitate per impostazione predefinita, ma non sono obbligatorie. Se gli utenti che crittografano i dati e gli utenti che decifrano i dati sono altrettanto affidabili, potresti prendere in considerazione l'utilizzo di una suite di algoritmi che non includa firme digitali per migliorare le tue prestazioni. Per ulteriori informazioni sulla selezione di suite di algoritmi alternative, consulta Scelta di una suite di algoritmi.

Come funziona il AWS Database Encryption SDK

> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Il AWS Database Encryption SDK fornisce librerie di crittografia lato client progettate specificamente per proteggere i dati archiviati nei database. Le librerie includono implementazioni sicure che puoi estendere o utilizzare senza modificarle. Per ulteriori informazioni sulla definizione e l'uso dei componenti personalizzati, consulta il GitHub repository per l'implementazione del database.

I flussi di lavoro in questa sezione spiegano come AWS Database Encryption SDK crittografa, firma, decrittografa e verifica i dati nel tuo database. Questi flussi di lavoro descrivono il processo di base utilizzando elementi astratti e funzionalità predefinite. Per informazioni dettagliate su come funziona AWS Database Encryption SDK con l'implementazione del database, consulta l'argomento Cos'è crittografato per il tuo database.

Il AWS Database Encryption SDK utilizza la crittografia a busta per proteggere i tuoi dati. Ogni record è crittografato con una chiave dati univoca. La chiave dati viene utilizzata per derivare una chiave di crittografia dei dati univoca per ogni campo contrassegnato `ENCRYPT_AND_SIGN` nelle azioni crittografiche. Quindi, una copia della chiave dati viene crittografata dalle chiavi di wrapping specificate. Per decrittografare il record crittografato, AWS Database Encryption SDK utilizza le chiavi di wrapping specificate per decrittografare almeno una chiave di dati crittografata. Quindi può decrittografare il testo cifrato e restituire una voce di testo non crittografato.

Per ulteriori informazioni sui termini utilizzati nel AWS Database Encryption SDK, vedereAWS Concetti dell'SDK per la crittografia dei database.

Crittografa e firma

Fondamentalmente, AWS Database Encryption SDK è un crittografo di record che crittografa, firma, verifica e decrittografa i record nel database. Raccoglie informazioni sui tuoi record e le istruzioni su quali campi crittografare e firmare. Riceve i materiali di crittografia e le istruzioni su come utilizzarli da un gestore di materiali crittografici configurato dalla chiave di wrapping specificata.

La procedura dettagliata seguente descrive come AWS Database Encryption SDK crittografa e firma i dati immessi.

1. Il gestore dei materiali crittografici fornisce al AWS Database Encryption SDK chiavi di crittografia dei dati univoche: una chiave dati in testo normale, una copia della chiave dati crittografata dalla chiave di wrapping specificata e una chiave MAC.

   > **Note**
   >
   > È possibile crittografare la chiave dati con più chiavi di avvolgimento. Ciascuna delle chiavi di wrapping crittografa una copia separata della chiave dati. Il AWS Database Encryption SDK memorizza tutte le chiavi di dati crittografate nella descrizione del materiale. Il AWS Database Encryption SDK aggiunge un nuovo campo (`aws_dbe_head`) al record che memorizza la descrizione del materiale.
   > Viene derivata una chiave MAC per ogni copia crittografata della chiave dati. I tasti MAC non vengono memorizzati nella descrizione del materiale. Invece, il metodo di decrittografia utilizza le chiavi wrapping per derivare nuovamente le chiavi MAC.

2. Il metodo di crittografia crittografa ogni campo contrassegnato come `ENCRYPT_AND_SIGN` nelle azioni crittografiche specificate.
3. Il metodo di crittografia ricava a `commitKey` dalla chiave dati e la utilizza per generare un valore di impegno della chiave, quindi elimina la chiave dati.
4. Il metodo di crittografia aggiunge una descrizione del materiale al record. La descrizione del materiale contiene le chiavi dati crittografate e le altre informazioni sul record crittografato. Per un elenco completo delle informazioni incluse nella descrizione del materiale, vedere Formato di descrizione del materiale.
5. Il metodo di crittografia utilizza le chiavi MAC restituite nel passaggio 1 per calcolare i valori HMAC (Hash-Based Message Authentication Code) durante la canonicalizzazione della descrizione del materiale, del contesto di crittografia e di ogni campo contrassegnato `ENCRYPT_AND_SIGN` o `SIGN_ONLY` nelle azioni crittografiche. I valori HMAC vengono

memorizzati in un nuovo campo (`aws_dbe_foot`) che il metodo di crittografia aggiunge al record.

6. Il metodo di crittografia calcola una firma ECDSA mediante la canonicalizzazione della descrizione del materiale, del contesto di crittografia e di ogni campo contrassegnato `ENCRYPT_AND_SIGN` o `SIGN_ONLY` e memorizza le firme ECDSA nel campo. `aws_dbe_foot`

   > **Note**
   >
   > Le firme ECDSA sono abilitate per impostazione predefinita, ma non sono obbligatorie.

7. Il metodo di crittografia memorizza il record crittografato e firmato nel tuo database

Decrittografa e verifica

1. Il gestore dei materiali crittografici (CMM) fornisce il metodo di decrittografia con i materiali di decrittografia memorizzati nella descrizione del materiale, inclusa la chiave dati in chiaro e la chiave MAC associata.

   - La CMM decrittografa la chiave dati crittografata con le chiavi di avvolgimento nel portachiavi specificato e restituisce la chiave dati in testo non crittografato.

2. Il metodo di decrittografia confronta e verifica il valore di impegno chiave nella descrizione del materiale.
3. Il metodo di decrittografia verifica le firme nel campo della firma.

   Identifica quali campi sono contrassegnati `ENCRYPT_AND_SIGN` e `SIGN_ONLY` dall'elenco dei campi consentiti non autenticati che hai definito. Il metodo di decrittografia utilizza la chiave MAC restituita nel passaggio 1 per ricalcolare e confrontare i valori HMAC per i campi contrassegnati con o. `ENCRYPT_AND_SIGN` `SIGN_ONLY` Quindi, verifica le firme ECDSA utilizzando la chiave pubblica memorizzata nel contesto di crittografia.

4. Il metodo di decrittografia utilizza la chiave dati in testo normale per decrittografare ogni valore contrassegnato. `ENCRYPT_AND_SIGN` L'SDK AWS Database Encryption quindi elimina la chiave dati in testo normale.
5. Il metodo di decrittografia restituisce il record in testo normale.

Suite di algoritmi supportate nel AWS Database Encryption SDK

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce ancora informazioni sul client di crittografia DynamoDB.

Una suite di algoritmi è una raccolta di algoritmi di crittografia e dei relativi valori. I sistemi crittografici utilizzano l'implementazione di algoritmi per generare il messaggio di testo cifrato.

Il AWS Database Encryption SDK utilizza una suite di algoritmi per crittografare e firmare i campi del database. Il AWS Database Encryption SDK supporta due suite di algoritmi. Tutte le suite supportate utilizzano Advanced Encryption Standard (AES) come algoritmo primario e lo combinano con altri algoritmi e valori.

Suite di algoritmi predefinita

La suite di algoritmi AWS Database Encryption SDK utilizza l'algoritmo Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM), noto come AES-GCM, per crittografare i dati grezzi. Il AWS Database Encryption SDK supporta chiavi di crittografia a 256 bit. La lunghezza del tag di autenticazione è sempre 16 byte.

Per impostazione predefinita, AWS Database Encryption SDK utilizza una suite di algoritmi con AES-GCM con una funzione di derivazione delle extract-and-expand chiavi basata su HMAC (HKDF), impegno delle chiavi, firma simmetrica e asimmetrica e una chiave di crittografia a 256 bit.

Il AWS Database Encryption SDK utilizza una suite di algoritmi che ricava una chiave dati AES-GCM fornendo una chiave di crittografia dei dati a 256 bit alla funzione di derivazione delle chiavi basata su HMAC (HKDF). extract-and-expand Ricava anche una chiave MAC per la chiave dati. Il AWS Database Encryption SDK utilizza questa chiave dati per derivare una chiave di crittografia dei dati univoca per crittografare ogni campo. Quindi, AWS Database Encryption SDK utilizza la chiave MAC per calcolare un codice HMAC (Hash-Based Message Authentication Code) per ogni copia crittografata della chiave dati e aggiunge una firma ECDSA (Elliptic Curve Digital Signature Algorithm) al record. Questa suite di algoritmi comporta anche un impegno fondamentale: un HMAC che associa la chiave dei dati al record. Il valore di impegno chiave è un HMAC calcolato dalla descrizione del materiale e dalla chiave di impegno, derivata tramite HKDF utilizzando una procedura simile alla derivazione della chiave di crittografia dei dati. Il valore chiave dell'impegno viene quindi memorizzato nella descrizione del materiale.

Algoritmo di crittografia	Lunghezza della chiave di crittografia dei dati (in bit)	Algoritmo di firma simmetrica	Algoritmo di firma asimmetrica	Impegno chiave
AES-GCM	256	HMAC-SHA-384	ECDSA su P384	HKDF con SHA-512

Questa suite di algoritmi serializza la descrizione del materiale e tutti i campi contrassegnati `ENCRYPT_AND_SIGN` e `SIGN_ONLY` nelle azioni crittografiche, quindi utilizza HMAC con un algoritmo di funzione hash crittografica (SHA-512) per firmare la canonicalizzazione. Quindi calcola una firma digitale ECDSA. Le firme HMAC ed ECDSA sono archiviate in un nuovo campo (`aws_dbe_foot`) che AWS Database Encryption SDK aggiunge al record. Le firme digitali sono particolarmente utili quando la politica di autorizzazione consente a un gruppo di utenti di crittografare i dati e a un gruppo diverso di utenti di decrittografare i dati.

L'impegno chiave garantisce che ogni testo cifrato venga decrittografato in un solo testo non crittografato. Lo fanno convalidando la chiave dati utilizzata come input per l'algoritmo di crittografia. Durante la crittografia, queste suite di algoritmi derivano un impegno chiave HMAC. Prima della decrittografia, convalidano che la chiave dati produce lo stesso impegno di chiave HMAC. In caso contrario, la chiamata di decrittografia fallisce.

AES-GCM senza firme digitali

Sebbene la suite di algoritmi predefinita sia probabilmente adatta alla maggior parte delle applicazioni, puoi scegliere una suite di algoritmi alternativa. Ad esempio, alcuni modelli di fiducia sarebbero soddisfatti da una suite di algoritmi senza firme digitali. Usa questa suite solo quando gli utenti che crittografano i dati e quelli che decrittografano i dati sono altrettanto affidabili.

Tutte le suite di algoritmi AWS Database Encryption SDK supportano la firma simmetrica HMAC-SHA-384. L'unica differenza è che la suite di algoritmi AES-GCM senza firme digitali è priva della firma ECDSA che fornisce un ulteriore livello di autenticità e non ripudio.

Ad esempio, se nel portachiavi sono presenti più chiavi di avvolgimento,, e `wrappingKeyAwrappingKeyB`, e decifrate un record utilizzando `wrappingKeyCwrappingKeyA`, la firma simmetrica HMAC-SHA-384 verifica che il record sia stato crittografato da un utente con accesso a. `wrappingKeyA` Se hai utilizzato gli algoritmi predefiniti, gli HMAC forniscono la stessa

verifica e utilizzano inoltre la firma ECDSA per garantire che il record sia stato crittografato da un utente con autorizzazioni di crittografia per. `wrappingKeyA` `wrappingKeyA`

Per selezionare la suite di algoritmi AES-GCM senza firme digitali, specificala nella configurazione di crittografia.

Utilizzo del AWS Database Encryption SDK con AWS KMS

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Per utilizzare AWS Database Encryption SDK, è necessario configurare un portachiavi e specificare una o più chiavi di wrapping. Se non un'infrastruttura di chiavi non è disponibile, consigliamo di utilizzare AWS Key Management Service (AWS KMS).

Il AWS Database Encryption SDK supporta due tipi di AWS KMS portachiavi. Il tradizionale AWS KMSportachiavi viene utilizzato AWS KMS keysper generare, crittografare e decrittografare le chiavi dati. È possibile utilizzare la crittografia simmetrica (`SYMMETRIC_DEFAULT`) o le chiavi RSA KMS asimmetriche. Poiché AWS Database Encryption SDK crittografa e firma ogni record con una chiave dati univoca, il AWS KMS portachiavi deve AWS KMS richiedere ogni operazione di crittografia e decrittografia. Per le applicazioni che devono ridurre al minimo il numero di chiamate versoAWS KMS, AWS Database Encryption SDK supporta anche il portachiavi AWS KMSgerarchico. Il portachiavi gerarchico è una soluzione di memorizzazione nella cache dei materiali crittografici che riduce il numero di AWS KMS chiamate utilizzando chiavi di derivazione AWS KMS protette conservate in una tabella Amazon DynamoDB e quindi memorizzando nella cache locale i materiali delle chiavi di diramazione utilizzati nelle operazioni di crittografia e decrittografia. Si consiglia di utilizzare i AWS KMS portachiavi ogni volta che è possibile.

Per interagire conAWS KMS, AWS Database Encryption SDK richiede il AWS KMS modulo di. AWS SDK for Java

Per prepararsi a utilizzare il AWS Database Encryption SDK con AWS KMS

1. Creazione di un Account AWS Per ulteriori informazioni, vedi Come posso creare e attivare un nuovo account Amazon Web Services? nel AWS Knowledge Center.
2. Crea una crittografia simmetrica. AWS KMS key Per assistenza, consulta Creazione di chiavi nella Guida per gli AWS Key Management Service sviluppatori.

> **Tip**
>
> Per utilizzare a livello di AWS KMS key programmazione, è necessario l'Amazon Resource Name (ARN) di. AWS KMS key Per assistenza nell'individuazione dell'ARN di un codiceAWS KMS key, consulta Individuazione dell'ID chiave e dell'ARN nella Guida per gli AWS Key Management Servicesviluppatori.

3. Genera un ID chiave di accesso e una chiave di accesso di sicurezza. Puoi utilizzare l'ID della chiave di accesso e la chiave di accesso segreta per un utente IAM oppure puoi utilizzarli per AWS Security Token Service creare una nuova sessione con credenziali di sicurezza temporanee che includono un ID chiave di accesso, una chiave di accesso segreta e un token di sessione. Come best practice di sicurezza, ti consigliamo di utilizzare credenziali temporanee anziché le credenziali a lungo termine associate ai tuoi account utente IAM o utente AWS (root).

   Per creare un utente IAM con una chiave di accesso, consulta Creating IAM Users nella IAM User Guide.

   Per generare credenziali di sicurezza temporanee, consulta Richiesta di credenziali di sicurezza temporanee nella Guida per l'utente IAM.
4. Imposta AWS le tue credenziali utilizzando le istruzioni contenute nell'AWS SDK for JavaID della chiave di accesso e nella chiave di accesso segreta che hai generato nel passaggio 3. Se hai generato credenziali temporanee, dovrai anche specificare il token di sessione.

   Questa procedura consente al SDK AWS di firmare le richieste ad AWS per te. Gli esempi di codice nel AWS Database Encryption SDK con cui interagisci AWS KMS presuppongono che tu abbia completato questo passaggio.

Configurazione del Database Encryption SDK AWS

La nostra libreria di crittografia lato client è stata rinominata Database Encryption SDK. AWS Questa guida per sviluppatori fornisce ancora informazioni sul DynamoDB Encryption Client.

Il AWS Database Encryption SDK è progettato per essere facile da usare. Sebbene AWS Database Encryption SDK abbia diverse opzioni di configurazione, i valori predefiniti vengono scelti con cura per essere pratici e sicuri per la maggior parte delle applicazioni. Tuttavia, potrebbe essere necessario modificare la configurazione per migliorare le prestazioni o includere una funzionalità personalizzata nella progettazione.

Argomenti

Selezione delle chiavi di avvolgimento

Il AWS Database Encryption SDK genera una chiave dati simmetrica unica per crittografare ogni campo. Non è necessario configurare, gestire o utilizzare le chiavi dati. AWS Database Encryption SDK lo fa per te.

Tuttavia, è necessario selezionare una o più chiavi di wrapping per crittografare ogni chiave di dati. AWS Database Encryption SDK supporta AWS Key Management Service(AWS KMS) chiavi KMS di crittografia simmetrica e chiavi KMS RSA asimmetriche. Supporta anche chiavi simmetriche AES e chiavi asimmetriche RSA fornite in diverse dimensioni. Sei responsabile della sicurezza e della durata delle tue chiavi di wrapping, quindi ti consigliamo di utilizzare una chiave di crittografia in un modulo di sicurezza hardware o in un servizio di infrastruttura chiave, ad esempio. AWS KMS

Per specificare le chiavi di avvolgimento per la crittografia e la decrittografia, si utilizza un portachiavi. A seconda del tipo di portachiavi utilizzato, è possibile specificare una chiave di avvolgimento o più chiavi di avvolgimento dello stesso tipo o di tipi diversi. Se utilizzi più chiavi di wrapping per racchiudere una chiave dati, ogni chiave di wrapping crittograferà una copia della stessa chiave dati.

Le chiavi dati crittografate (una per chiave di avvolgimento) vengono memorizzate nella descrizione del materiale memorizzata accanto al campo crittografato. Per decrittografare i dati, il AWS Database Encryption SDK deve prima utilizzare una delle chiavi di wrapping per decrittografare una chiave dati crittografata.

Ti consigliamo di utilizzare uno dei portachiavi quando possibile. AWS KMS Il AWS Database Encryption SDK fornisce il AWS KMS portachiavi e il portachiaviAWS KMS gerarchico, che riducono il numero di chiamate effettuate a. AWS KMSPer specificare un elemento AWS KMS key in un portachiavi, utilizza un identificatore di chiave supportato. AWS KMS Se si utilizza il portachiavi AWS KMS gerarchico, è necessario specificare l'ARN della chiave. Per i dettagli sugli identificatori chiave per una chiave, consulta Identificatori AWS KMS chiave nella Guida per gli sviluppatori.AWS Key Management Service

- Quando si esegue la crittografia con un AWS KMS portachiavi, è possibile specificare qualsiasi identificatore di chiave valido (ARN della chiave, nome alias, alias ARN o ID chiave) per una chiave KMS di crittografia simmetrica. Se si utilizza una chiave RSA KMS asimmetrica, è necessario specificare la chiave ARN.

  Se si specifica un nome alias o un alias ARN per una chiave KMS durante la crittografia, AWS Database Encryption SDK salva la chiave ARN attualmente associata a quell'alias; non salva l'alias. Le modifiche all'alias non influiscono sulla chiave KMS utilizzata per decrittografare le chiavi dati.
- Per impostazione predefinita, il AWS KMS portachiavi decripta i record in modalità rigorosa (dove si specificano particolari chiavi KMS). È necessario utilizzare una chiave ARN per l'identificazione AWS KMS keys per la decrittografia.

  Quando si esegue la crittografia con un AWS KMS portachiavi, AWS Database Encryption SDK memorizza l'ARN della chiave AWS KMS key nella descrizione del materiale con la chiave dati crittografata. Durante la decrittografia in modalità rigorosa, AWS Database Encryption SDK verifica che la stessa chiave ARN sia presente nel portachiavi prima di tentare di utilizzare la chiave di wrapping per decrittografare la chiave dati crittografata. Se si utilizza un identificatore di chiave diverso, AWS Database Encryption SDK non lo riconoscerà né lo utilizzerà, anche se gli identificatori si riferiscono alla AWS KMS keystessa chiave.
- Durante la decrittografia in modalità Discovery, non viene specificata alcuna chiave di wrapping. Innanzitutto, il AWS Database Encryption SDK tenta di decrittografare il record con la chiave ARN memorizzata nella descrizione del materiale. Se ciò non funziona, AWS Database Encryption SDK chiede AWS KMS di decrittografare il record utilizzando la chiave KMS che lo ha crittografato, indipendentemente da chi possiede o ha accesso a quella chiave KMS.

Per specificare una chiave AES non elaborata o una coppia di chiavi RSA non elaborata come chiave di wrapping in un portachiavi, è necessario specificare uno spazio dei nomi e un nome. Durante la decrittografia, è necessario utilizzare lo stesso identico spazio dei nomi e lo stesso nome per ogni chiave di wrapping non elaborata utilizzata durante la crittografia. Se utilizzi un namespace o un nome diverso, AWS Database Encryption SDK non riconoscerà né utilizzerà la chiave di wrapping, anche se il materiale della chiave è lo stesso.

Creazione di un filtro di rilevamento

Quando si decifrano dati crittografati con chiavi KMS, è consigliabile decrittografarli in modalità rigorosa, ovvero limitare le chiavi di wrapping utilizzate solo a quelle specificate dall'utente. Tuttavia, se necessario, puoi anche decrittografare in modalità di scoperta, in cui non specifichi alcuna chiave di wrapping. In questa modalità, AWS KMS puoi decrittografare la chiave dati crittografata utilizzando la chiave KMS che l'ha crittografata, indipendentemente da chi possiede o ha accesso a quella chiave KMS.

Se è necessario decrittografare in modalità di rilevamento, si consiglia di utilizzare sempre un filtro di rilevamento, che limita le chiavi KMS che possono essere utilizzate a quelle presenti in una partizione e specificata. Account AWS Il filtro di rilevamento è facoltativo, ma è una procedura consigliata.

Utilizza la tabella seguente per determinare il valore della partizione per il filtro di rilevamento.

Regione	Partizione
Regioni AWS	`aws`
Regioni della Cina	`aws-cn`
AWS GovCloud (US) Regions	`aws-us-gov`

Il seguente esempio di Java mostra come creare un filtro di rilevamento. Prima di utilizzare il codice, sostituite i valori di esempio con valori validi per la partizione Account AWS and.

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

Lavorare con database multitenant

Con AWS Database Encryption SDK, puoi configurare la crittografia lato client per i database con uno schema condiviso isolando ogni tenant con materiali di crittografia distinti. Quando prendi in considerazione un database multitenant, dedica del tempo a esaminare i requisiti di sicurezza e il modo in cui la multitenancy potrebbe influire su di essi. Ad esempio, l'utilizzo di un database multitenant potrebbe influire sulla capacità di combinare Database Encryption SDK con un'altra soluzione di crittografia lato server AWS .

Se più utenti eseguono operazioni di crittografia all'interno del database, puoi utilizzare uno dei AWS KMS portachiavi per fornire a ciascun utente una chiave distinta da utilizzare nelle proprie operazioni crittografiche. La gestione delle chiavi dati per una soluzione di crittografia lato client multitenant può essere complicata. Ti consigliamo di organizzare i dati per tenant quando possibile. Se il tenant è identificato dai valori della chiave primaria (ad esempio, la chiave di partizione in una tabella Amazon DynamoDB), la gestione delle chiavi è più semplice.

Puoi usare il AWS KMS portachiavi per isolare ogni tenant con un portachiavi distinto e. AWS KMS AWS KMS keysIn base al volume di AWS KMS chiamate effettuate per inquilino, potresti voler utilizzare il portachiavi AWS KMS gerarchico per ridurre al minimo le chiamate a. AWS KMSIl portachiaviAWS KMS Hierarchical è una soluzione di memorizzazione nella cache dei materiali crittografici che riduce il numero di AWS KMS chiamate utilizzando chiavi branch AWS KMS protette persistenti in una tabella Amazon DynamoDB e quindi memorizzando nella cache locale i materiali chiave delle branch utilizzati nelle operazioni di crittografia e decrittografia. È necessario utilizzare il portachiavi Hierarchical per implementare la crittografia ricercabile nel database. AWS KMS

Creazione di beacon firmati

AWS Database Encryption SDK utilizza beacon standard e beacon compositi per fornire soluzioni di crittografia ricercabili che consentono di cercare record crittografati senza decrittografare l'intero database interrogato. Tuttavia, AWS Database Encryption SDK supporta anche beacon firmati che possono essere configurati interamente da campi di testo in chiaro. `SIGN_ONLY` I beacon firmati sono un tipo di beacon composto che indicizza ed esegue query complesse sui campi. `SIGN_ONLY`

Ad esempio, se disponete di un database multitenant, potreste voler creare un beacon firmato che consenta di interrogare il database alla ricerca di record crittografati dalla chiave di un tenant specifico. Per ulteriori informazioni, consulta Interrogazione dei beacon in un database multitenant.

È necessario utilizzare il portachiavi AWS KMS gerarchico per creare beacon firmati.

Per configurare un beacon firmato, fornite i seguenti valori.

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

Nome del beacon

Il nome che usi quando interroghi il faro.

Il nome di un beacon firmato non può avere lo stesso nome di un campo non crittografato. Due beacon non possono avere lo stesso nome.

Carattere diviso

Il carattere usato per separare le parti che compongono il faro firmato.

Il carattere diviso non può apparire nei valori in chiaro di nessuno dei campi da cui è costruito il beacon firmato.

Elenco delle parti firmate

Identifica i `SIGN_ONLY` campi inclusi nel beacon firmato.

Ogni parte deve includere un nome, una fonte e un prefisso. L'origine è il `SIGN_ONLY` campo identificato dalla parte. L'origine deve essere un nome di campo o un indice che si riferisce al valore di un campo annidato. Se il nome della parte identifica la fonte, puoi omettere la fonte e AWS Database Encryption SDK utilizzerà automaticamente il nome come fonte. Ti consigliamo di specificare l'origine come nome della parte ogni volta che è possibile. Il prefisso può essere qualsiasi stringa, ma deve essere univoco. Due parti firmate in un beacon firmato non possono avere lo stesso prefisso. Si consiglia di utilizzare un valore breve che distingua la parte dalle altre parti servite dal beacon firmato. Per semplificare le interrogazioni sui beacon, si consiglia inoltre di identificare una parte con lo stesso prefisso in ogni beacon in cui è inclusa ed evitare di utilizzare lo stesso prefisso per identificare parti diverse.

```
List<SignedPart> signedPartList = new ArrayList<>);
```

```
    SignedPart signedPartExample = SignedPart.builder()
        .name("signedFieldName")
        .prefix("S-")
        .build();
    signedPartList.add(signedPartExample);
```

Elenco dei costruttori (opzionale)

Identifica i costruttori che definiscono i diversi modi in cui le parti firmate possono essere assemblate dal faro firmato.

Se non specificate un elenco di costruttori, AWS Database Encryption SDK assembla il beacon firmato con il seguente costruttore predefinito.

- Tutte le parti firmate nell'ordine in cui sono state aggiunte all'elenco delle parti firmate
- Tutte le parti sono obbligatorie

Costruttori

Ogni costruttore è un elenco ordinato di parti del costruttore che definisce un modo in cui il faro firmato può essere assemblato. Le parti del costruttore vengono unite nell'ordine in cui vengono aggiunte all'elenco, con ogni parte separata dal carattere di divisione specificato.

Ogni parte del costruttore nomina una parte firmata e definisce se tale parte è obbligatoria o facoltativa all'interno del costruttore. Ad esempio, se si desidera interrogare un faro firmato su`Field1`, and `Field1.Field2``Field1.Field2.Field3`, contrassegnare e `Field3` come facoltativo `Field2` e creare un costruttore.

Ogni costruttore deve avere almeno una parte obbligatoria. Si consiglia di rendere obbligatoria la prima parte di ogni costruttore in modo da poter utilizzare l'`BEGINS_WITH`operatore nelle query.

Un costruttore ha successo se tutte le parti necessarie sono presenti nel record. Quando si scrive un nuovo record, il beacon firmato utilizza l'elenco dei costruttori per determinare se il beacon può essere assemblato in base ai valori forniti. Tenta di assemblare il beacon nell'ordine in cui i costruttori sono stati aggiunti all'elenco dei costruttori e utilizza il primo costruttore che riesce. Se nessun costruttore ha successo, il beacon non viene scritto nel record.

Tutti i lettori e gli scrittori devono specificare lo stesso ordine di costruttori per garantire che i risultati delle query siano corretti.

Utilizzate le seguenti procedure per specificare il vostro elenco di costruttori.

1. Create una parte costruttore per ogni parte firmata per definire se quella parte è necessaria o meno.

   Il nome della parte del costruttore deve essere il nome del campo firmato.

   L'esempio seguente dimostra come creare una parte costruttore per un campo firmato.

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

2. Create un costruttore per ogni modo possibile in cui il faro firmato può essere assemblato utilizzando le parti del costruttore create nel passaggio 1.

   Ad esempio, se si desidera eseguire un'interrogazione su `Field1.Field2.Field3` and`Field4.Field2.Field3`, è necessario creare due costruttori. `Field1`e `Field4` possono essere entrambi obbligatori perché sono definiti in due costruttori separati.

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

3. Create un elenco di costruttori che includa tutti i costruttori creati nel passaggio 2.

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

```
constructorList.add(field421Constructor)
```

4. Specificate `constructorList` quando create il beacon firmato.

Utilizzo dei keyring

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Il AWS Database Encryption SDK utilizza i portachiavi per eseguire la crittografia delle buste. I keyring generano, crittografano e decrittano le chiavi di dati. I portachiavi determinano l'origine delle chiavi dati univoche che proteggono ogni record crittografato e le chiavi di avvolgimento che crittografano quella chiave dati. Puoi specificare un keyring durante la crittografia e lo stesso keyring o uno diverso durante la decrittazione.

I keyring possono essere utilizzati singolarmente o combinati in keyring multipli. Anche se la maggior parte dei keyring è in grado di generare, crittografare e decrittare le chiavi di dati, ne puoi creare uno che esegua solo una determinata operazione, ad esempio la generazione delle chiavi di dati, e utilizzarlo in combinazione con altri.

Ti consigliamo di utilizzare un portachiavi che protegga le chiavi di avvolgimento ed esegua operazioni crittografiche entro un limite sicuro, come il AWS KMS portachiavi, che utilizza AWS KMS keys that never leave () non crittografato. AWS Key Management ServiceAWS KMS Puoi anche scrivere un portachiavi che utilizzi chiavi di avvolgimento memorizzate nei moduli di sicurezza hardware (HSM) o protette da altri servizi di chiavi master.

Questo argomento spiega come utilizzare la funzione portachiavi di AWS Database Encryption SDK e come scegliere un portachiavi.

Argomenti

Come funzionano i keyring

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Quando crittografi e firmi un campo nel tuo database, Database Encryption SDK richiede al AWS portachiavi i materiali di crittografia. Il portachiavi restituisce una chiave dati in testo normale, una copia della chiave dati crittografata da ciascuna delle chiavi di avvolgimento del portachiavi e una chiave MAC associata alla chiave dati. Il AWS Database Encryption SDK utilizza la chiave di testo normale per crittografare i dati e quindi rimuove la chiave di dati in testo normale dalla memoria il prima possibile. Quindi, AWS Database Encryption SDK aggiunge una descrizione del materiale che include le chiavi di dati crittografate e altre informazioni, come le istruzioni di crittografia e firma. Il AWS Database Encryption SDK utilizza la chiave MAC per calcolare i codici di autenticazione dei messaggi basati su hash (HMAC) sulla canonicalizzazione della descrizione del materiale e di tutti i campi contrassegnati con o. `ENCRYPT_AND_SIGN` `SIGN_ONLY`

Quando decrittografi i dati, puoi usare lo stesso portachiavi che hai usato per crittografare i dati o uno diverso. Per decrittografare i dati, un portachiavi di decrittografia deve avere accesso ad almeno una chiave di avvolgimento nel portachiavi di crittografia.

Il AWS Database Encryption SDK passa le chiavi di dati crittografate dalla descrizione del materiale al portachiavi e chiede al portachiavi di decrittografare ognuna di esse. Il keyring utilizza le chiavi di wrapping per decrittare una delle chiavi di dati crittografate e restituisce una chiave di dati di testo normale. Il AWS Database Encryption SDK utilizza la chiave dati in testo normale per decrittografare i dati. Se nessuna delle chiavi di wrapping nel keyring è in grado di decrittare una qualsiasi delle chiavi di dati crittografate, l'operazione di decrittazione non riesce.

Puoi utilizzare un singolo keyring o combinarne più di uno dello stesso tipo o di tipi diversi in un keyring multiplo. Quando si crittografano i dati, il portachiavi multiplo restituisce una copia della chiave dati crittografata da tutte le chiavi di avvolgimento di tutti i portachiavi che compongono il portachiavi multiplo e una chiave MAC associata alla chiave dati. Puoi decrittografare i dati utilizzando un portachiavi con una qualsiasi delle chiavi di avvolgimento nel portachiavi multiplo.

Scegliere un portachiavi

> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Il tuo portachiavi determina le chiavi di avvolgimento che proteggono le tue chiavi dati e, in ultima analisi, i tuoi dati. Usa le chiavi di imballaggio più sicure e pratiche per il tuo compito. Quando

possibile, usa chiavi di wrapping protette da un modulo di sicurezza hardware (HSM) o da un'infrastruttura di gestione delle chiavi, come le chiavi KMS in AWS Key Management Service(AWS KMS) o le chiavi di crittografia in ingresso. AWS CloudHSM

Il AWS Database Encryption SDK fornisce diverse configurazioni di portachiavi e portachiavi ed è possibile creare portachiavi personalizzati. Puoi anche creare un portachiavi multiplo che includa uno o più portachiavi dello stesso tipo o di un tipo diverso.

Argomenti

- Prtachiavi AWS KMS
- AWS KMS Portachiavi gerarchici
- Keyring non elaborati AES
- Keyring non elaborato RSA
- Keyring multipli

Prtachiavi AWS KMS

> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Un AWS KMS portachiavi utilizza la crittografia simmetrica o RSA asimmetrica AWS KMS keysper generare, crittografare e decrittografare le chiavi dati. AWS Key Management Service(AWS KMS) protegge le chiavi KMS ed esegue operazioni crittografiche entro i limiti FIPS. Ti consigliamo di utilizzare un AWS KMS portachiavi o un portachiavi con proprietà di sicurezza simili, quando possibile.

Puoi anche utilizzare una chiave KMS multiregionale simmetrica in un portachiavi. AWS KMS Per ulteriori dettagli ed esempi di utilizzo di Multi-regionAWS KMS keys, vedereUtilizzo di più aree geografiche AWS KMS keys. Per informazioni sulle chiavi multiregionali, consulta Utilizzo di chiavi multiregionali nella Guida per gli AWS Key Management Servicesviluppatori.

AWS KMSi portachiavi possono includere due tipi di chiavi da imballaggio:

- Chiave generatrice: genera una chiave dati in testo normale e la crittografa. Un portachiavi che crittografa i dati deve avere una chiave del generatore.

- Chiavi aggiuntive: crittografa la chiave di dati in testo normale generata dalla chiave del generatore. AWS KMSi portachiavi possono avere zero o più chiavi aggiuntive.

È necessario disporre di una chiave generatrice per crittografare i record. Quando un AWS KMS portachiavi ha una sola AWS KMS chiave, quella chiave viene utilizzata per generare e crittografare la chiave dati.

Come tutti i portachiavi, i AWS KMS portachiavi possono essere usati indipendentemente o in un portachiavi multiplo con altri portachiavi dello stesso tipo o di tipo diverso.

Argomenti

- Autorizzazioni richieste per i portachiavi AWS KMS
- Identificazione di AWS KMS keys in un keyring AWS KMS
- Creare un AWS KMS portachiavi
- Utilizzo di più aree geografiche AWS KMS keys
- Utilizzo di un keyring di individuazione AWS KMS
- Utilizzo di un keyring di individuazione regionale AWS KMS

Autorizzazioni richieste per i portachiavi AWS KMS

Il AWS Database Encryption SDK non richiede Account AWS e non dipende da nessunoServizio AWS. Tuttavia, per utilizzare un AWS KMS portachiavi, è necessario disporre delle seguenti autorizzazioni minime sul AWS KMS keys portachiavi. Account AWS

- Per crittografare con un AWS KMS portachiavi, è necessario il GenerateDataKey permesso kms: sulla chiave del generatore. È necessaria l'autorizzazione KMS:Encrypt su tutte le chiavi aggiuntive nel portachiavi. AWS KMS
- Per decifrare con un AWS KMS portachiavi, è necessaria l'autorizzazione KMS:Decrypt su almeno una chiave nel portachiavi. AWS KMS
- Per crittografare con un portachiavi composto da portachiavi, è necessario il GenerateDataKey permesso kms: sulla AWS KMS chiave del generatore nel portachiavi del generatore. È necessaria l'autorizzazione KMS:Encrypt su tutte le altre chiavi in tutti gli altri portachiavi. AWS KMS

Per informazioni dettagliate sulle autorizzazioni perAWS KMS keys, consulta Autenticazione e controllo degli accessi nella Guida per gli AWS Key Management Service sviluppatori.

Identificazione di AWS KMS keys in un keyring AWS KMS

Un AWS KMS portachiavi può includerne uno o piùAWS KMS keys. Per specificare un elemento AWS KMS key in un AWS KMS portachiavi, usa un identificatore di AWS KMS chiave supportato. Gli identificatori chiave che è possibile utilizzare per identificare un elemento AWS KMS key in un portachiavi variano a seconda dell'operazione e dell'implementazione del linguaggio. Per i dettagli sugli identificatori chiave di unAWS KMS key, consulta Identificatori chiave nella Guida per gli sviluppatori. AWS Key Management Service

Come procedura consigliata, utilizza l'identificatore chiave più specifico e pratico per la tua attività.

- Per crittografare con un AWS KMS portachiavi, puoi utilizzare un ID chiave, un ARN chiave, un nome alias o un alias ARN per crittografare i dati.

  > **Note**
  >
  > Se si specifica un nome alias o un alias ARN per una chiave KMS in un portachiavi di crittografia, l'operazione di crittografia salva la chiave ARN attualmente associata all'alias nei metadati della chiave dati crittografata. Non salva l'alias. Le modifiche all'alias non influiscono sulla chiave KMS utilizzata per decrittografare le chiavi dati crittografate.

- Per decifrare con un AWS KMS portachiavi, è necessario utilizzare una chiave ARN per identificare. AWS KMS keys Per informazioni dettagliate, consultare Selezione delle chiavi di avvolgimento.
- In un keyring utilizzato per la crittografia e la decrittazione devi utilizzare un ARN di chiave per identificare le AWS KMS keys.

Durante la decrittografia, AWS Database Encryption SDK cerca nel AWS KMS portachiavi un elemento in AWS KMS key grado di decrittografare una delle chiavi di dati crittografate. In particolare, AWS Database Encryption SDK utilizza il seguente schema per ogni chiave di dati crittografata nella descrizione del materiale.

- Il AWS Database Encryption SDK ottiene la chiave ARN di chi AWS KMS key ha crittografato la chiave dati dai metadati della descrizione del materiale.
- Il AWS Database Encryption SDK cerca nel portachiavi di decrittografia un ARN AWS KMS key con una chiave corrispondente.

- Se trova un ARN AWS KMS key con una chiave corrispondente nel portachiavi, AWS Database Encryption SDK chiede di utilizzare la chiave KMS AWS KMS per decrittografare la chiave dati crittografata.
- In caso contrario, passa alla chiave di dati crittografata successiva, se presente.

Creare un AWS KMS portachiavi

È possibile configurare ogni AWS KMS portachiavi con uno AWS KMS key o più AWS KMS keys chiavi uguali o diversiAccount AWS. Regioni AWS AWS KMS keyDeve essere una chiave di crittografia simmetrica (`SYMMETRIC_DEFAULT`) o una chiave RSA KMS asimmetrica. È inoltre possibile utilizzare una chiave KMS multiregionale con crittografia simmetrica. È possibile utilizzare uno o più AWS KMS portachiavi in un portachiavi multiplo.

Puoi creare un AWS KMS portachiavi che crittografa e decrittografa i dati oppure puoi creare AWS KMS portachiavi specifici per la crittografia o la decrittografia. Quando si crea un AWS KMS portachiavi per crittografare i dati, è necessario specificare una chiave generatrice, AWS KMS key che viene utilizzata per generare una chiave di dati in testo non crittografato e crittografarla. La chiave dati non è matematicamente correlata alla chiave KMS. Facoltativamente, puoi anche specificare ulteriori AWS KMS keys per crittografare la stessa chiave di dati di testo normale. Per decrittografare un campo crittografato protetto da questo portachiavi, il portachiavi di decrittografia utilizzato deve includere almeno uno dei valori AWS KMS keys definiti nel portachiavi, oppure no. AWS KMS keys (Un AWS KMS portachiavi senza nome AWS KMS keys è noto come portachiavi AWS KMS Discovery.)

Tutte le chiavi racchiuse in un portachiavi crittografico o in un portachiavi multiplo devono essere in grado di crittografare la chiave dati. Se una chiave di wrapping non riesce a crittografare, il metodo di crittografia fallisce. Di conseguenza, il chiamante deve disporre delle autorizzazioni necessarie per tutte le chiavi del portachiavi. Se si utilizza un portachiavi di rilevamento per crittografare i dati, da solo o in un portachiavi multiplo, l'operazione di crittografia ha esito negativo.

Il seguente esempio Java utilizza un `CreateAwsKmsMrkMultiKeyring` metodo per creare un AWS KMS portachiavi con una chiave KMS di crittografia simmetrica. Il `CreateAwsKmsMrkMultiKeyring` metodo garantisce che il portachiavi gestisca correttamente sia le chiavi monoregionali che quelle multiregionali. L'esempio utilizza una chiave ARN per identificare la chiave KMS.

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

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

Utilizzo di più aree geografiche AWS KMS keys

Puoi utilizzare Multi-region AWS KMS keys come chiavi di wrapping nel AWS Database Encryption SDK. Se si esegue la crittografia con una chiave multiregionale in unaRegione AWS, è possibile decrittografare utilizzando una chiave multiregionale correlata in un'altra. Regione AWS

Le chiavi KMS multiregionali sono un insieme di AWS KMS keys chiavi diverse Regioni AWS che hanno lo stesso materiale chiave e lo stesso ID chiave. Puoi usare queste chiavi correlate come se fossero la stessa chiave in regioni diverse. Le chiavi multiregionali supportano scenari di disaster recovery e backup comuni che richiedono la crittografia in una regione e la decrittografia in un'altra regione senza effettuare chiamate interregionali. AWS KMS Per informazioni sulle chiavi multiregionali, consulta Utilizzo di chiavi multiregionali nella Guida per gli AWS Key Management Servicesviluppatori.

Per supportare le chiavi multiregionali, il AWS Database Encryption SDK include portachiavi compatibili con AWS KMS più regioni. Il `CreateAwsKmsMrkMultiKeyring` metodo supporta chiavi sia a regione singola che a più regioni.

- Per le chiavi a regione singola, il simbolo compatibile con più regioni si comporta esattamente come il portachiavi a regione singola. AWS KMS Tenta di decrittografare il testo cifrato solo con la chiave a regione singola che ha crittografato i dati. Per semplificare l'esperienza AWS KMS con il portachiavi, consigliamo di utilizzare questo `CreateAwsKmsMrkMultiKeyring` metodo ogni volta che si utilizza una chiave KMS con crittografia simmetrica.
- Per le chiavi multiregionali, il simbolo multiregionale tenta di decrittografare il testo cifrato con la stessa chiave multiregionale che ha crittografato i dati o con la relativa chiave multiregionale nella regione specificata.

Nei portachiavi compatibili con più aree geografiche che richiedono più di una chiave KMS, puoi specificare più chiavi monoregionali e multiregionali. Tuttavia, è possibile specificare solo una chiave per ogni set di chiavi multiregionali correlate. Se specificate più di un identificatore chiave con lo stesso ID chiave, la chiamata al costruttore ha esito negativo.

L'esempio Java seguente crea un AWS KMS portachiavi con una chiave KMS multiregionale. L'esempio specifica una chiave multiregionale come chiave del generatore e una chiave a regione singola come chiave figlio

```
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);
```

Quando si utilizzano AWS KMS portachiavi multiregionali, è possibile decrittografare il testo cifrato in modalità rigorosa o in modalità di scoperta. Per decrittografare il testo cifrato in modalità rigorosa, crea un'istanza del simbolo compatibile con più regioni con la chiave ARN della relativa chiave multiregionale nella regione in cui stai decrittografando il testo cifrato. Se specifichi la chiave ARN di una chiave multiregionale correlata in un'altra regione (ad esempio, la regione in cui il record è stato crittografato), il simbolo che riconosce più regioni effettuerà una chiamata interregionale a tale scopo. AWS KMS key

Quando si esegue la decrittografia in modalità rigorosa, il simbolo compatibile con più regioni richiede una chiave ARN. Accetta solo una chiave ARN da ogni set di chiavi multiregionali correlate.

Puoi anche decrittografare in modalità scoperta con chiavi AWS KMS multiregionali. Durante la decrittografia in modalità scoperta, non ne viene specificata alcuna. AWS KMS keys (Per informazioni sui portachiavi Single-region AWS KMS Discovery, consultaUtilizzo di un keyring di individuazione AWS KMS.)

Se hai crittografato con una chiave multiregionale, il simbolo multiregionale in modalità di rilevamento tenterà di decrittografare utilizzando una chiave multiregionale correlata nella regione locale. Se non esiste, la chiamata fallisce. In modalità di rilevamento, AWS Database Encryption SDK non tenterà una chiamata interregionale per la chiave multiregionale utilizzata per la crittografia.

Utilizzo di un keyring di individuazione AWS KMS

Durante la decrittografia, è consigliabile specificare le chiavi di wrapping che il AWS Database Encryption SDK può utilizzare. Per seguire questa procedura consigliata, utilizzate un portachiavi

AWS KMS di decrittografia che limiti le chiavi di AWS KMS avvolgimento a quelle specificate. Tuttavia, puoi anche creare un portachiavi AWS KMS Discovery, ovvero un AWS KMS portachiavi che non specifichi alcuna chiave di avvolgimento.

Il AWS Database Encryption SDK fornisce un portachiavi di AWS KMS rilevamento standard e un portachiavi di rilevamento per AWS KMS chiavi multiregionali. Per informazioni sull'utilizzo delle chiavi multiregionali con AWS Database Encryption SDK, vedere. Utilizzo di più aree geografiche AWS KMS keys

Poiché non specifica alcuna chiave di avvolgimento, un portachiavi Discovery non può crittografare i dati. Se si utilizza un portachiavi di rilevamento per crittografare i dati, da solo o in un portachiavi multiplo, l'operazione di crittografia ha esito negativo.

Durante la decrittografia, un discovery keyring consente al AWS Database Encryption SDK di chiedere AWS KMS di decrittografare qualsiasi chiave di dati crittografata utilizzando AWS KMS key quella crittografata, indipendentemente da chi la possiede o ne ha accesso. AWS KMS key La chiamata ha esito positivo solo quando il chiamante dispone dell'`kms:Decrypt`autorizzazione per il. AWS KMS key

> **Important**
>
> Se includi un portachiavi AWS KMS Discovery in un portachiavi con crittografia multipla, il portachiavi Discovery sostituisce tutte le restrizioni relative alla chiave KMS specificate dagli altri portachiavi del portachiavi multiplo. Il portachiavi multiplo si comporta come il portachiavi meno restrittivo. Se si utilizza un portachiavi di rilevamento per crittografare i dati, da solo o in un portachiavi multiplo, l'operazione di crittografia fallisce

Il AWS Database Encryption SDK fornisce un AWS KMS comodo portachiavi di rilevamento. ma, se possibile, consigliamo di utilizzare un keyring di portata più limitata per i motivi seguenti.

- Autenticità: un portachiavi di AWS KMS scoperta può utilizzare qualsiasi chiave utilizzata per crittografare una chiave di dati nella descrizione del materiale, a condizione AWS KMS key che il chiamante sia autorizzato a utilizzarla per la decrittografia. AWS KMS key anche se potrebbe non essere la AWS KMS key che intende impiegare. Ad esempio, una delle chiavi di dati potrebbe essere stata crittografata con una AWS KMS key meno sicura rispetto allo standard richiesto.
- Latenza e prestazioni: un portachiavi AWS KMS Discovery potrebbe essere sensibilmente più lento di altri portachiavi perché AWS Database Encryption SDK tenta di decrittografare tutte le chiavi

di dati crittografate, comprese quelle crittografate da AWS KMS keys in other Account AWS e Regions, e AWS KMS keys che il chiamante non è autorizzato a utilizzare per la decrittografia.

Se utilizzi un portachiavi di rilevamento, ti consigliamo di utilizzare un filtro di rilevamento per limitare le chiavi KMS che possono essere utilizzate a quelle nelle partizioni Account AWS e nelle partizioni specificate. Per assistenza nell'individuazione dell'ID e della partizione dell'account, consulta I tuoi Account AWS identificatori e il formato ARN nel. Riferimenti generali di AWS

Il codice Java seguente istanzia un portachiavi di AWS KMS rilevamento con un filtro di rilevamento che limita le chiavi KMS che AWS Database Encryption SDK può utilizzare a quelle nella partizione e nell'awsaccount di esempio. `111122223333`

Prima di utilizzare questo codice, sostituisci i valori di esempio Account AWS e della partizione con valori validi per la tua partizione Account AWS and. Se le tue chiavi KMS si trovano nelle regioni della Cina, usa il valore della `aws-cn` partizione. Se le tue chiavi KMS sono inseriteAWS GovCloud (US) Regions, usa il valore della `aws-us-gov` partizione. Per tutti gli altriRegioni AWS, usa il valore `aws` della partizione.

```
// 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);
```

Utilizzo di un keyring di individuazione regionale AWS KMS

Un portachiavi di rilevamento AWS KMS regionale è un portachiavi che non specifica gli ARN delle chiavi KMS. Consente invece al AWS Database Encryption SDK di decrittografare utilizzando solo le chiavi KMS in particolare. Regioni AWS

Durante la decrittografia con un portachiavi di rilevamento AWS KMS regionale, AWS Database Encryption SDK decrittografa qualsiasi chiave di dati crittografata crittografata in base a un punto specificato. AWS KMS key Regione AWS Per avere successo, il chiamante deve disporre

dell'`kms:Decrypt`autorizzazione su almeno una delle chiavi AWS KMS keys di dati specificate Regione AWS che hanno crittografato.

Come altri portachiavi Discovery, il portachiavi di rilevamento regionale non ha alcun effetto sulla crittografia. Funziona solo quando si decifrano campi crittografati. Se si utilizza un portachiavi di rilevamento regionale in un portachiavi multiplo utilizzato per la crittografia e la decrittografia, è efficace solo durante la decrittografia. Se si utilizza un portachiavi di rilevamento multiregionale per crittografare i dati, da solo o in un portachiavi multiplo, l'operazione di crittografia non riesce.

> **Important**
>
> Se includi un portachiavi di rilevamento AWS KMS regionale in un portachiavi multichiave per la decrittografia, il portachiavi di rilevamento regionale sostituisce tutte le restrizioni relative alle chiavi KMS specificate dagli altri portachiavi del portachiavi multiplo. Il portachiavi multiplo si comporta come il portachiavi meno restrittivo. Un portachiavi AWS KMS Discovery non ha alcun effetto sulla crittografia se utilizzato da solo o in un portachiavi multiplo.

Il portachiavi di rilevamento regionale in AWS Database Encryption SDK tenta di decrittografare solo con le chiavi KMS nella regione specificata. Quando si utilizza un portachiavi Discovery, si configura la regione sul AWS KMS client. Queste implementazioni di AWS Database Encryption SDK non filtrano le chiavi KMS per regione, ma AWS KMS falliranno una richiesta di decrittografia per le chiavi KMS al di fuori della regione specificata.

Se utilizzi un portachiavi di rilevamento, ti consigliamo di utilizzare un filtro di rilevamento per limitare le chiavi KMS utilizzate nella decrittografia a quelle nelle partizioni e nelle partizioni specificate. Account AWS

Ad esempio, il codice seguente crea un portachiavi di rilevamento AWS KMS regionale con un filtro di scoperta. Questo portachiavi limita il AWS Database Encryption SDK alle chiavi KMS nell'account 111122223333 nella regione Stati Uniti occidentali (Oregon) (us-west-2).

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

```
        .discoveryFilter(discoveryFilter)
        .regions("us-west-2")
        .build();
IKeyring decryptKeyring =
 matProv.CreateAwsKmsMrkDiscoveryMultiKeyring(createAwsKmsMrkDiscoveryMultiKeyringInput);
```

AWS KMS Portachiavi gerarchici

La nostra libreria di crittografia lato client è stata rinominata Database Encryption SDK. AWS Questa guida per sviluppatori fornisce ancora informazioni sul DynamoDB Encryption Client.

> **Note**
>
> A partire dal 24 luglio 2023, le chiavi di filiale create durante l'anteprima per sviluppatori non sono supportate. Crea nuove chiavi di filiale per continuare a utilizzare l'archivio di chiavi di filiale creato durante l'anteprima per sviluppatori.

Con il portachiavi AWS KMS Hierarchical, puoi proteggere i tuoi materiali crittografici con una chiave KMS a crittografia simmetrica senza chiamare AWS KMS ogni volta che crittografi o decrittografi un record. È una buona scelta per le applicazioni che devono ridurre al minimo le chiamate e le applicazioni che possono riutilizzare alcuni materiali crittografici AWS KMSsenza violare i requisiti di sicurezza.

Il portachiavi Hierarchical è una soluzione di memorizzazione nella cache dei materiali crittografici che riduce il numero di AWS KMS chiamate utilizzando chiavi branch AWS KMS protette persistenti in una tabella Amazon DynamoDB e quindi memorizzando nella cache locale i materiali chiave delle branch utilizzati nelle operazioni di crittografia e decrittografia. La tabella DynamoDB funge da archivio delle chiavi di filiale che gestisce e protegge le chiavi delle filiali. Memorizza la chiave di ramo attiva e tutte le versioni precedenti della chiave di ramo. La chiave di ramo attiva è la versione più recente della chiave di filiale. Il portachiavi Hierarchical utilizza una chiave dati unica per crittografare ogni campo e crittografa ogni chiave di dati con una chiave di wrapping unica derivata dalla chiave branch attiva. Il portachiavi Hierarchical dipende dalla gerarchia stabilita tra le chiavi branch attive e le relative chiavi di wrapping derivate.

Il portachiavi Hierarchical utilizza in genere ogni versione della chiave branch per soddisfare più richieste. Tuttavia, puoi controllare la misura in cui le chiavi di ramo attive vengono riutilizzate e

determinare la frequenza con cui la chiave di ramo attiva viene ruotata. La versione attiva della chiave di ramo rimane attiva finché non viene ruotata. Le versioni precedenti della chiave di ramo attiva non verranno utilizzate per eseguire operazioni di crittografia, ma potranno comunque essere interrogate e utilizzate nelle operazioni di decrittografia.

Quando si crea un'istanza del portachiavi Hierarchical, viene creata una cache locale. Si specifica un limite di cache che definisce la quantità massima di tempo in cui i materiali chiave del branch vengono archiviati nella cache locale prima che scadano e vengano rimossi dalla cache. Il portachiavi Hierarchical effettua una AWS KMS chiamata per decrittografare la chiave del ramo e assemblare i materiali delle chiavi del ramo la prima volta che a viene specificato in un'operazione. `branch-key-id` I materiali delle chiavi di filiale vengono quindi archiviati nella cache locale e riutilizzati per tutte le operazioni di crittografia e decrittografia che lo specificano fino alla scadenza del limite di cache. `branch-key-id` La memorizzazione dei materiali chiave della filiale nella cache locale riduce le chiamate. AWS KMS Ad esempio, si consideri un limite di cache di 15 minuti. Se si eseguono 10.000 operazioni di crittografia entro tale limite di cache, il AWS KMS portachiavi tradizionale dovrebbe effettuare 10.000 AWS KMS chiamate per soddisfare 10.000 operazioni di crittografia. Se ne hai uno attivo`branch-key-id`, il portachiavi Hierarchical deve effettuare solo una AWS KMS chiamata per soddisfare 10.000 operazioni di crittografia.

La cache locale è composta da due partizioni, una per le operazioni di crittografia e una seconda per le operazioni di decrittografia. La partizione di crittografia memorizza i materiali delle chiavi di branch assemblati dalla chiave branch attiva e li riutilizza per tutte le operazioni di crittografia fino alla scadenza del limite della cache. La partizione di decrittografia memorizza i materiali delle chiavi di filiale assemblati per altre versioni di chiavi di filiale identificate nelle operazioni di decrittografia. La partizione di decrittografia può memorizzare più versioni di materiali chiave di filiale attivi contemporaneamente. Quando è configurata per utilizzare un fornitore di ID di chiavi di filiale per un database multitenant, la partizione encrypt può anche archiviare più versioni di materiali relativi alle chiavi di filiale contemporaneamente. Per ulteriori informazioni, consulta Utilizzo del portachiavi Hierarchical con database multitenant.

> **Note**
>
> Tutte le menzioni del portachiavi gerarchico nel Database Encryption SDK si riferiscono al AWS portachiavi gerarchico. AWS KMS

Argomenti

- Come funziona

- Prerequisiti
- Crea un portachiavi gerarchico
- Ruota la chiave branch attiva
- Utilizzo del portachiavi Hierarchical con database multitenant
- Utilizzo del portachiavi Hierarchical per una crittografia ricercabile

Come funziona

Le seguenti procedure dettagliate descrivono come il portachiavi Hierarchical assembla i materiali di crittografia e decrittografia e le diverse chiamate che il portachiavi effettua per le operazioni di crittografia e decrittografia. Per i dettagli tecnici sulla derivazione delle chiavi di wrapping e sui processi di crittografia delle chiavi di dati in chiaro, consulta Dettagli tecnici del portachiavi gerarchico.AWS KMS

Crittografa e firma

La procedura dettagliata seguente descrive come il portachiavi Hierarchical assembla i materiali di crittografia e ricava una chiave di avvolgimento univoca.

1. Il metodo di crittografia richiede al portachiavi Hierarchical i materiali di crittografia. Il portachiavi genera una chiave di dati in testo semplice, quindi verifica se nella cache locale sono presenti materiali branch validi per generare la chiave di wrapping. Se sono presenti materiali validi per le chiavi di filiale, il portachiavi passa alla Fase 5.
2. Se non ci sono materiali validi per le chiavi di filiale, il portachiavi Hierarchical interroga l'archivio delle chiavi di filiale per trovare la chiave di filiale attiva.
    a. L'archivio delle chiavi di filiale chiama AWS KMS per decrittografare la chiave di ramo attiva e restituisce la chiave di ramo attiva in testo semplice. I dati che identificano la chiave di ramo attiva vengono serializzati per fornire dati autenticati aggiuntivi (AAD) nella chiamata di decrittografia a. AWS KMS
    b. L'archivio delle chiavi di filiale restituisce la chiave di ramo in testo semplice e i dati che la identificano, ad esempio la versione della chiave di filiale.
3. Il portachiavi Hierarchical assembla i materiali chiave del ramo (la chiave di ramo in testo semplice e la versione della chiave di ramo) e ne archivia una copia nella cache locale.

4. Il portachiavi Hierarchical ricava una chiave di avvolgimento unica dalla chiave branch in testo semplice e un sale casuale a 16 byte. Utilizza la chiave di wrapping derivata per crittografare una copia della chiave dati in chiaro.

Il metodo di crittografia utilizza i materiali di crittografia per crittografare e firmare il record. Per ulteriori informazioni su come i record vengono crittografati e firmati nel AWS Database Encryption SDK, consulta Encrypt and sign.

Decrittografa e verifica

La procedura dettagliata seguente descrive come il portachiavi gerarchico assembla i materiali di decrittografia e decrittografa la chiave di dati crittografata.

1. Il metodo di decrittografia identifica la chiave di dati crittografata dal campo di descrizione del materiale del record crittografato e la passa al portachiavi gerarchico.
2. Il portachiavi Hierarchical deserializza i dati che identificano la chiave dati crittografata, inclusa la versione della chiave branch, il sale da 16 byte e altre informazioni che descrivono come è stata crittografata la chiave dati.

   Per ulteriori informazioni, consulta AWS KMSDettagli tecnici del portachiavi gerarchico.
3. Il portachiavi Hierarchical verifica se nella cache locale sono presenti materiali chiave di filiale validi che corrispondono alla versione della chiave di filiale identificata nel passaggio 2. Se sono presenti materiali validi per le chiavi di filiale, il portachiavi passa alla Fase 6.
4. Se non ci sono materiali validi per le chiavi di filiale, il portachiavi Hierarchical interroga l'archivio delle chiavi di filiale per trovare la chiave di filiale che corrisponde alla versione della chiave di filiale identificata nello Step 2.

   a. L'archivio delle chiavi di filiale chiama AWS KMS per decrittografare la chiave di ramo e restituisce la chiave di ramo attiva in testo semplice. I dati che identificano la chiave di ramo attiva vengono serializzati per fornire dati autenticati aggiuntivi (AAD) nella chiamata di decrittografia a. AWS KMS

   b. L'archivio delle chiavi di filiale restituisce la chiave di ramo in testo semplice e i dati che la identificano, ad esempio la versione della chiave di filiale.
5. Il portachiavi Hierarchical assembla i materiali chiave del ramo (la chiave di ramo in testo semplice e la versione della chiave di ramo) e ne archivia una copia nella cache locale.

6. Il portachiavi Hierarchical utilizza i materiali delle chiavi branch assemblate e il sale da 16 byte identificato nella fase 2 per riprodurre la chiave di avvolgimento univoca che crittografava la chiave dati.
7. Il portachiavi Hierarchical utilizza la chiave di wrapping riprodotta per decrittografare la chiave dati e restituisce la chiave dati in testo semplice.

Il metodo di decrittografia utilizza i materiali di decrittografia e la chiave di dati in testo semplice per decrittografare e verificare il record. Per ulteriori informazioni su come i record vengono decrittografati e verificati nel Database Encryption SDK, consulta Decriptare e verificare. AWS

Prerequisiti

Il AWS Database Encryption SDK non richiede Account AWS e non dipende da nessuno. Servizio AWSTuttavia, il portachiavi gerarchico dipende da Amazon AWS KMS DynamoDB.

Per utilizzare un portachiavi gerarchico, è necessaria una crittografia simmetrica con autorizzazioni KMS:Decrypt. AWS KMS key È inoltre possibile utilizzare una chiave multiregionale con crittografia simmetrica. Per informazioni dettagliate sulle autorizzazioni per AWS KMS keys, consulta Autenticazione e controllo degli accessi nella Guida per gli sviluppatori.AWS Key Management Service

Prima di poter creare e utilizzare un portachiavi gerarchico, è necessario creare l'archivio delle chiavi della filiale e compilarlo con la prima chiave di filiale attiva.

Passaggio 1: configurare un nuovo servizio di archiviazione delle chiavi

Il servizio di archiviazione delle chiavi offre diverse operazioni, ad esempio `CreateKeyStore` e`CreateKey`, per aiutarti a assemblare i prerequisiti gerarchici del portachiavi e a gestire l'archivio delle chiavi della filiale.

Il seguente esempio di Java crea un servizio di archiviazione delle chiavi. È necessario specificare un nome di tabella DynamoDB che funga da nome dell'archivio chiavi della filiale, un nome logico per l'archivio chiavi della filiale e l'ARN della chiave KMS che identifica la chiave KMS che proteggerà le chiavi della filiale.

Il nome dell'archivio di chiavi logiche è associato crittograficamente a tutti i dati memorizzati nella tabella per semplificare le operazioni di ripristino di DynamoDB. Il nome dell'archivio di chiavi logiche può essere lo stesso del nome della tabella DynamoDB, ma non è necessario. Consigliamo vivamente di specificare il nome della tabella DynamoDB come nome della tabella

logica quando si configura per la prima volta il servizio di archiviazione delle chiavi. È necessario specificare sempre lo stesso nome di tabella logica. Nel caso in cui il nome dell'archivio chiavi della filiale cambi dopo il ripristino della tabella DynamoDB da un backup, il nome dell'archivio di chiavi logico viene mappato al nome della tabella DynamoDB specificato per garantire che il portachiavi Hierarchical possa ancora accedere all'archivio delle chiavi della filiale.

```
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();
```

Fase 2: Chiama per creare un archivio di chiavi della filiale **`CreateKeyStore`**

La seguente operazione Java crea l'archivio delle chiavi di filiale che persisterà e proteggerà le chiavi della filiale.

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

L'`CreateKeyStore`operazione crea una tabella DynamoDB con il nome della tabella specificato nel passaggio 1 e i seguenti valori obbligatori.

	Chiave di partizione	Chiave di ordinamento
Tabella di base	`branch-key-id`	`version`

Fase 3: Chiama **`CreateKey`** per creare una nuova chiave branch attiva

La seguente operazione Java crea una nuova chiave di ramo attiva utilizzando la chiave KMS specificata nel passaggio 1 e aggiunge la chiave di ramo attiva alla tabella DynamoDB creata nel passaggio 2.

Quando si chiama`CreateKey`, è possibile scegliere di specificare i seguenti valori opzionali.

- `branchKeyIdentifier`: definisce una personalizzazione`branch-key-id`.

Per creare una personalizzazione`branch-key-id`, è necessario includere anche un contesto di crittografia aggiuntivo con il `encryptionContext` parametro.

- `encryptionContext`: definisce un set opzionale di coppie chiave-valore non segrete che fornisce dati autenticati aggiuntivi (AAD) nel contesto di crittografia incluso nella chiamata kms:. GenerateDataKeyWithoutPlaintext

  Questo contesto di crittografia aggiuntivo viene visualizzato con il prefisso. `aws-crypto-ec:`

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

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

Innanzitutto, l'`CreateKey`operazione genera i seguenti valori.

- Un identificatore univoco universale (UUID) versione 4 per (a meno che non sia stato specificato un identificatore personalizzato). `branch-key-id` `branch-key-id`
- Un UUID versione 4 per la versione branch key
- A `timestamp` nel formato di data e ora ISO 8601 in formato UTC (Coordinated Universal Time).

Quindi, l'`CreateKey`operazione chiama kms: GenerateDataKeyWithoutPlaintext utilizzando la seguente richiesta.

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

```
    "NumberOfBytes": "32"
}
```

> **Note**
>
> L'`CreateKey`operazione crea una chiave branch attiva e una chiave beacon, anche se il database non è stato configurato per la crittografia ricercabile. Entrambe le chiavi sono archiviate nell'archivio delle chiavi della filiale. Per ulteriori informazioni, consulta Utilizzo del portachiavi gerarchico per la crittografia ricercabile.

Successivamente, l'`CreateKey`operazione chiama kms: ReEncrypt per creare un record attivo per la chiave branch aggiornando il contesto di crittografia.

Infine, l'**`CreateKey`**operazione chiama ddb: TransactWriteItems per scrivere un nuovo elemento che mantenga la chiave di ramo nella tabella creata nel passaggio 2. L'elemento ha i seguenti attributi.

```
{
    "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"
}
```

Crea un portachiavi gerarchico

Per inizializzare il portachiavi gerarchico, è necessario fornire i seguenti valori:

- Il nome di un archivio di chiavi della filiale

  Il nome della tabella DynamoDB che hai creato per fungere da archivio delle chiavi della filiale.

-

  Un limite di durata della cache (TTL)

La quantità di tempo, in secondi, durante la quale una chiave di filiale deve essere inserita nella cache locale può essere utilizzata prima della scadenza. Questo valore deve essere maggiore di zero. Quando il limite TTL della cache scade, la voce viene rimossa dalla cache locale.

- Un identificatore di chiave di ramo

  Il `branch-key-id` che identifica la chiave di filiale attiva nell'archivio delle chiavi della filiale.

  > **Note**
  >
  > Per inizializzare il portachiavi Hierarchical per l'uso multitenant, è necessario specificare un fornitore di ID di chiavi di filiale anziché un. `branch-key-id` Per ulteriori informazioni, consulta Utilizzo del portachiavi Hierarchical con database multitenant.

- (Facoltativo) Un elenco di token di concessione

  Se controlli l'accesso alla chiave KMS nel tuo portachiavi gerarchico con le concessioni, devi fornire tutti i token di concessione necessari quando inizializzi il portachiavi.

Il seguente esempio Java dimostra come inizializzare un portachiavi gerarchico con il client Database AWS Encryption SDK per DynamoDB. L'esempio seguente specifica un limite di cache TTL di 600 secondi.

```
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);
```

Ruota la chiave branch attiva

Può esserci una sola versione attiva per ogni chiave di ramo alla volta. Il portachiavi Hierarchical utilizza in genere ogni versione di chiave branch attiva per soddisfare più richieste. Ma sei tu a

controllare la misura in cui le chiavi branch attive vengono riutilizzate e a determinare la frequenza con cui la chiave branch attiva viene ruotata.

Le chiavi branch non vengono utilizzate per crittografare le chiavi di dati in testo semplice. Vengono utilizzate per derivare le chiavi di wrapping univoche che crittografano le chiavi di dati in testo non crittografato. Il processo di derivazione della chiave di wrapping produce una chiave di wrapping unica da 32 byte con 28 byte di casualità. Ciò significa che una chiave branch può derivare più di 79 ottilioni, o 2 96, chiavi di wrapping uniche prima che si verifichi l'usura crittografica. Nonostante questo rischio di esaurimento molto basso, potrebbe essere necessario ruotare le chiavi di filiale attive a causa di norme aziendali o contrattuali o normative governative.

La versione attiva della chiave di filiale rimane attiva finché non viene ruotata. Le versioni precedenti della chiave branch attiva non verranno utilizzate per eseguire operazioni di crittografia e non possono essere utilizzate per derivare nuove chiavi di wrapping. Tuttavia, possono comunque essere interrogate e fornire chiavi di wrapping per decrittografare le chiavi di dati che hanno crittografato mentre erano attive.

Utilizza il servizio di archiviazione delle chiavi per ruotare `VersionKey` la chiave branch attiva. Quando si ruota la chiave di ramo attiva, viene creata una nuova chiave di ramo per sostituire la versione precedente. Non `branch-key-id` cambia quando si ruota la chiave di ramo attiva. È necessario specificare la chiave `branch-key-id` che identifica la chiave di ramo attiva corrente quando si chiama. `VersionKey`

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

Utilizzo del portachiavi Hierarchical con database multitenant

È possibile utilizzare la gerarchia di chiavi stabilita tra le chiavi branch attive e le relative chiavi di wrapping derivate per supportare i database multitenant creando una chiave branch per ogni tenant del database. Il portachiavi Hierarchical crittografa e firma quindi tutti i dati di un determinato tenant con la relativa chiave branch distinta. Ciò consente di archiviare i dati di più tenant in un unico database e di isolare i dati del tenant per chiave di filiale.

Ogni tenant ha la propria chiave di filiale definita da un'unica. `branch-key-id` Può esserci solo una versione attiva di ciascuno `branch-key-id` alla volta.

Prima di poter inizializzare il portachiavi Hierarchical per l'uso multitenant, è necessario creare una chiave di filiale per ogni tenant e creare un fornitore di ID di chiavi di filiale. Utilizza il fornitore di ID della chiave di filiale per creare un nome descrittivo per facilitare il riconoscimento di `branch-key-ids` quello corretto per un tenant. `branch-key-id` Ad esempio, il nome descrittivo consente di fare riferimento a una chiave di filiale come `tenant1` invece di`b3f61619-4d35-48ad-a275-050f87e15122`.

Per le operazioni di decrittografia, è possibile configurare staticamente un singolo portachiavi gerarchico per limitare la decrittografia a un singolo tenant, oppure è possibile utilizzare il fornitore di ID della chiave di filiale per identificare quale tenant è responsabile della decrittografia di un record.

Per prima cosa, segui i passaggi 1 e 2 delle procedure relative ai prerequisiti. Quindi, utilizzate le seguenti procedure per creare una chiave di filiale per ogni tenant, creare un fornitore di ID di chiavi di filiale e inizializzare il portachiavi Hierarchical per l'uso multitenant.

Passaggio 1: crea una chiave di filiale per ogni tenant nel database

Chiama `CreateKey` ogni inquilino del tuo database.

La seguente operazione Java crea due chiavi di ramo utilizzando la chiave KMS specificata durante la creazione del servizio di archivio chiavi e aggiunge le chiavi di ramo alla tabella DynamoDB creata per fungere da archivio chiavi di filiale. La stessa chiave KMS deve proteggere tutte le chiavi di filiale.

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

Fase 2: Creare un fornitore di chiavi di filiale (ID)

Il seguente esempio Java crea nomi descrittivi per le due chiavi di branch create nel passaggio 1 e chiama `CreateDynamoDbEncryptionBranchKeyIdSupplier` per creare un fornitore di ID di chiavi di filiale con il client AWS Database Encryption SDK per DynamoDB.

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

```
    private static String branchKeyIdForTenant2;

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

Fase 3: inizializza il portachiavi Hierarchical con il branch key ID (fornitore)

Per inizializzare il portachiavi gerarchico è necessario fornire i seguenti valori:

- Il nome di un archivio di chiavi della filiale
- Un limite di durata della cache (TTL)
- Un fornitore di codici identificativi per filiali
- (Facoltativo) Una cache

  Se desideri personalizzare il tipo di cache o il numero di voci relative ai materiali chiave della filiale che possono essere archiviate nella cache locale, specifica il tipo di cache e la capacità di accesso quando inizializzi il portachiavi.

  Il tipo di cache definisce il modello di threading. Il portachiavi Hierarchical fornisce tre tipi di cache che supportano database multitenant: Predefinito,,. MultiThreaded StormTracking

  Se non si specifica una cache, il portachiavi Hierarchical utilizza automaticamente il tipo di cache predefinito e imposta la capacità di ingresso su 1000.

  Default (Recommended)

  Per la maggior parte degli utenti, la cache predefinita soddisfa i requisiti di threading. La cache predefinita è progettata per supportare ambienti con molti multithread. Quando una voce relativa ai materiali delle chiavi di branch scade, la cache predefinita impedisce la chiamata di più thread AWS KMS notificando a un thread che la voce relativa ai materiali

della chiave di branch sta per scadere con 10 secondi di anticipo. Ciò garantisce che solo un thread invii una richiesta di aggiornamento della cache AWS KMS .

Per inizializzare il tuo portachiavi gerarchico con una cache predefinita, specifica il seguente valore:

- Capacità di ingresso: limita il numero di voci relative ai materiali chiave della filiale che possono essere archiviate nella cache locale.

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

L'impostazione predefinita e le StormTracking cache supportano lo stesso modello di threading, ma è sufficiente specificare la capacità di ingresso per inizializzare il portachiavi gerarchico con la cache predefinita. Per personalizzazioni più granulari della cache, usa la cache. StormTracking

MultiThreaded

La MultiThreaded cache è sicura da usare in ambienti multithread, ma non fornisce alcuna funzionalità per ridurre al minimo AWS KMS le chiamate Amazon DynamoDB. Di conseguenza, quando scade l'immissione di materiali chiave in una filiale, tutti i thread verranno avvisati contemporaneamente. Ciò può comportare più AWS KMS chiamate per aggiornare la cache.

Per inizializzare il tuo portachiavi Hierarchical con una MultiThreaded cache, specifica i seguenti valori:

- Capacità di ingresso: limita il numero di voci relative ai materiali chiave della filiale che possono essere archiviate nella cache locale.
- Entry Poting Tail Size: definisce il numero di elementi da potare se viene raggiunta la capacità di ingresso.

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

StormTracking

La StormTracking cache è progettata per supportare ambienti fortemente multithread. Quando una voce relativa ai materiali della chiave di filiale scade, la StormTracking cache impedisce la chiamata di più thread AWS KMS notificando in anticipo a un thread che la voce relativa ai materiali chiave della branch sta per scadere. Ciò garantisce che solo un thread invii una richiesta di aggiornamento della cache AWS KMS .

Per inizializzare il tuo portachiavi Hierarchical con una StormTracking cache, specifica i seguenti valori:

- Capacità di ingresso: limita il numero di voci relative ai materiali chiave della filiale che possono essere archiviate nella cache locale.
- Dimensione della coda di potatura d'ingresso: definisce il numero di materiali chiave del ramo da potare alla volta.

  Valore predefinito: 1 voce
- Periodo di tolleranza: definisce il numero di secondi prima della scadenza in cui viene effettuato un tentativo di aggiornare i materiali chiave della filiale.

  Valore predefinito: 10 secondi
- Intervallo di grazia: definisce il numero di secondi tra i tentativi di aggiornamento dei materiali chiave del ramo.

  Valore predefinito: 1 secondo
- Fan out: definisce il numero di tentativi simultanei che è possibile effettuare per aggiornare i materiali chiave della filiale.

  Valore predefinito: 20 tentativi
- In flight time to live (TTL): definisce il numero di secondi che mancano al timeout di un tentativo di aggiornamento dei materiali chiave della filiale. Ogni volta che la cache ritorna `NoSuchEntry` in risposta a una`GetCacheEntry`, quella chiave di ramo viene considerata in esecuzione finché la stessa chiave non viene scritta con una `PutCache` voce.

  Valore predefinito: 20 secondi
- Sospensione: definisce il numero di secondi in cui un thread deve essere sospeso se `fanOut` viene superato il limite.

Valore predefinito: 20 millisecondi

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

- (Facoltativo) Un elenco di token di concessione

  Se controlli l'accesso alla chiave KMS nel tuo portachiavi gerarchico con le concessioni, devi fornire tutti i token di concessione necessari quando inizializzi il portachiavi.

Il seguente esempio di Java inizializza un portachiavi Hierarchical con il branch key ID supplier creato nel passaggio 2, un TLL limite di cache di 600 secondi e una dimensione massima della cache di 1000. Questo esempio inizializza un portachiavi Hierarchical con il client Database AWS Encryption SDK per DynamoDB.

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

Utilizzo del portachiavi Hierarchical per una crittografia ricercabile

La crittografia ricercabile consente di cercare record crittografati senza decrittografare l'intero database. Ciò si ottiene indicizzando il valore di testo in chiaro di un campo crittografato con un beacon. Per implementare la crittografia ricercabile, è necessario utilizzare un portachiavi gerarchico.

L'*`CreateKey`*operazione di archiviazione delle chiavi genera sia una chiave branch che una chiave beacon. La chiave branch viene utilizzata nelle operazioni di crittografia e decrittografia dei record. La chiave beacon viene utilizzata per generare beacon.

La chiave branch e la chiave beacon sono protette dallo stesso AWS KMS key che hai specificato durante la creazione del servizio di archiviazione delle chiavi. Dopo che l'`CreateKey`operazione chiama AWS KMS per generare la chiave branch, chiama kms: GenerateDataKeyWithoutPlaintext una seconda volta per generare la chiave beacon utilizzando la seguente richiesta.

```
{
   "EncryptionContext": {
      "branch-key-id" : "branch-key-id",
      "type" : type,
      "create-time" : "timestamp",
      "logical-key-store-name" : "the logical table name for your branch key store",
      "kms-arn" : the KMS key ARN,
      "hierarchy-version" : 1
   },
   "KeyId": "the KMS key ARN",
   "NumberOfBytes": "32"
}
```

Dopo aver generato entrambe le chiavi, l'`CreateKey`operazione chiama ddb: TransactWriteItems per scrivere due nuovi elementi che mantengano la chiave branch e la chiave beacon nell'archivio delle chiavi della filiale.

Quando configuri un beacon standard, il AWS Database Encryption SDK interroga l'archivio delle chiavi del branch per cercare la chiave beacon. Quindi, utilizza una funzione di derivazione delle extract-and-expand chiavi basata su HMAC (HKDF) per combinare la chiave beacon con il nome del beacon standard per creare la chiave HMAC per un determinato beacon.

A differenza delle chiavi branch, esiste una sola versione di chiave beacon per ogni filiale. `branch-key-id` La chiave beacon non viene mai ruotata.

Definizione della fonte della chiave del beacon

Quando si definisce la versione beacon per i beacon standard e compositi, è necessario identificare la chiave beacon e definire un TTL (cache limit time to live) per i materiali chiave del beacon. I materiali delle chiavi beacon vengono archiviati in una cache locale separata dalle chiavi branch. Il seguente frammento mostra come definire un database single-tenant. `keySource` Identifica la tua chiave beacon in base alla quale è associata. `branch-key-id`

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

Definizione della fonte del beacon in un database multitenant

Se si dispone di un database multitenant, è necessario specificare i seguenti valori durante la configurazione di. `keySource`

- keyFieldName

  Definisce il nome del campo che memorizza la chiave beacon `branch-key-id` associata alla chiave beacon utilizzata per generare i beacon per un determinato tenant. `keyFieldName`Può essere una stringa qualsiasi, ma deve essere univoca per tutti gli altri campi del database. Quando si scrivono nuovi record nel database, la chiave `branch-key-id` che identifica la chiave beacon utilizzata per generare i beacon per quel record viene memorizzata in questo campo. È necessario includere questo campo nelle query del beacon e identificare i materiali chiave del beacon appropriati necessari per ricalcolare il beacon. Per ulteriori informazioni, consulta Interrogazione dei beacon in un database multitenant.
- CacheTTL

  La quantità di tempo, in secondi, necessaria per l'immissione di una chiave beacon nella cache locale del beacon prima della scadenza. Questo valore deve essere maggiore di zero. Quando il limite di cache TTL scade, la voce viene rimossa dalla cache locale.
- (Facoltativo) Una cache

Se desideri personalizzare il tipo di cache o il numero di voci relative ai materiali chiave della filiale che possono essere archiviate nella cache locale, specifica il tipo di cache e la capacità di accesso quando inizializzi il portachiavi.

Il tipo di cache definisce il modello di threading. Il portachiavi Hierarchical fornisce tre tipi di cache che supportano database multitenant: Predefinito,,. MultiThreaded StormTracking

Se non si specifica una cache, il portachiavi Hierarchical utilizza automaticamente il tipo di cache predefinito e imposta la capacità di ingresso su 1000.

Default (Recommended)

Per la maggior parte degli utenti, la cache predefinita soddisfa i requisiti di threading. La cache predefinita è progettata per supportare ambienti con molti multithread. Quando una voce relativa ai materiali delle chiavi di branch scade, la cache predefinita impedisce la chiamata di più thread AWS KMS notificando a un thread che la voce relativa ai materiali della chiave di branch sta per scadere con 10 secondi di anticipo. Ciò garantisce che solo un thread invii una richiesta di aggiornamento della cache AWS KMS .

Per inizializzare il tuo portachiavi gerarchico con una cache predefinita, specifica il seguente valore:

- Capacità di ingresso: limita il numero di voci relative ai materiali chiave della filiale che possono essere archiviate nella cache locale.

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

L'impostazione predefinita e le StormTracking cache supportano lo stesso modello di threading, ma è sufficiente specificare la capacità di ingresso per inizializzare il portachiavi gerarchico con la cache predefinita. Per personalizzazioni più granulari della cache, usa la cache. StormTracking

MultiThreaded

La MultiThreaded cache è sicura da usare in ambienti multithread, ma non fornisce alcuna funzionalità per ridurre al minimo AWS KMS le chiamate Amazon DynamoDB. Di conseguenza, quando scade l'immissione di materiali chiave in una filiale, tutti i thread

verranno avvisati contemporaneamente. Ciò può comportare più AWS KMS chiamate per aggiornare la cache.

Per inizializzare il tuo portachiavi Hierarchical con una MultiThreaded cache, specifica i seguenti valori:

- Capacità di ingresso: limita il numero di voci relative ai materiali chiave della filiale che possono essere archiviate nella cache locale.
- Entry Poting Tail Size: definisce il numero di elementi da potare se viene raggiunta la capacità di ingresso.

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

StormTracking

La StormTracking cache è progettata per supportare ambienti fortemente multithread. Quando una voce relativa ai materiali della chiave di filiale scade, la StormTracking cache impedisce la chiamata di più thread AWS KMS notificando in anticipo a un thread che la voce relativa ai materiali chiave della branch sta per scadere. Ciò garantisce che solo un thread invii una richiesta di aggiornamento della cache AWS KMS .

Per inizializzare il tuo portachiavi Hierarchical con una StormTracking cache, specifica i seguenti valori:

- Capacità di ingresso: limita il numero di voci relative ai materiali chiave della filiale che possono essere archiviate nella cache locale.
- Dimensione della coda di potatura d'ingresso: definisce il numero di materiali chiave del ramo da potare alla volta.

  Valore predefinito: 1 voce
- Periodo di tolleranza: definisce il numero di secondi prima della scadenza in cui viene effettuato un tentativo di aggiornare i materiali chiave della filiale.

  Valore predefinito: 10 secondi
- Intervallo di grazia: definisce il numero di secondi tra i tentativi di aggiornamento dei materiali chiave del ramo.

Valore predefinito: 1 secondo

- Fan out: definisce il numero di tentativi simultanei che è possibile effettuare per aggiornare i materiali chiave della filiale.

  Valore predefinito: 20 tentativi

- In flight time to live (TTL): definisce il numero di secondi che mancano al timeout di un tentativo di aggiornamento dei materiali chiave della filiale. Ogni volta che la cache ritorna `NoSuchEntry` in risposta a una`GetCacheEntry`, quella chiave di ramo viene considerata in esecuzione finché la stessa chiave non viene scritta con una `PutCache` voce.

  Valore predefinito: 20 secondi

- Sospensione: definisce il numero di secondi in cui un thread deve essere sospeso se `fanOut` viene superato il limite.

  Valore predefinito: 20 millisecondi

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

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

Keyring non elaborati AES

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Il AWS Database Encryption SDK consente di utilizzare una chiave simmetrica AES fornita come chiave di wrapping che protegge la chiave dei dati. È necessario generare, archiviare e proteggere il materiale chiave, preferibilmente in un modulo di sicurezza hardware (HSM) o in un sistema di gestione delle chiavi. Usa un portachiavi AES Raw quando devi fornire la chiave di avvolgimento e crittografare le chiavi dati localmente o offline.

Il portachiavi Raw AES crittografa i dati utilizzando l'algoritmo AES-GCM e una chiave di wrapping specificata come array di byte. Puoi specificare solo una chiave di avvolgimento in ogni portachiavi Raw AES, ma puoi includere più portachiavi Raw AES, da soli o con altri portachiavi, in un portachiavi multiplo.

Namespace e nomi chiave

Per identificare la chiave AES in un portachiavi, il portachiavi Raw AES utilizza uno spazio dei nomi chiave e un nome chiave forniti dall'utente. Questi valori non sono segreti. Appaiono in testo normale nella descrizione del materiale che AWS Database Encryption SDK aggiunge al record. Ti consigliamo di utilizzare uno spazio dei nomi delle chiavi, il tuo sistema HSM o di gestione delle chiavi e un nome chiave che identifichi la chiave AES in quel sistema.

> **Note**
>
> Lo spazio dei nomi e il nome chiave sono equivalenti ai campi Provider ID (o Provider) e Key ID in. `JceMasterKey`

Se costruisci portachiavi diversi per crittografare e decrittografare un determinato campo, i valori dello spazio dei nomi e dei nomi sono fondamentali. Se lo spazio dei nomi della chiave e il nome della chiave nel portachiavi di decrittografia non corrispondono esattamente, con distinzione tra maiuscole e minuscole, allo spazio dei nomi della chiave e al nome della chiave nel portachiavi di crittografia, il portachiavi di decrittografia non viene utilizzato, anche se i byte del materiale chiave sono identici.

Ad esempio, potresti definire un portachiavi AES Raw con lo spazio dei nomi `HSM_01` e il nome della chiave. `AES_256_012` Quindi, usi quel portachiavi per crittografare alcuni dati. Per decrittografare quei dati, crea un portachiavi AES Raw con lo stesso spazio dei nomi chiave, nome chiave e materiale chiave.

Il seguente esempio Java mostra come creare un portachiavi AES Raw. La `AESWrappingKey` variabile rappresenta il materiale chiave fornito.

```
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);
```

Keyring non elaborato RSA

> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Il portachiavi RSA Raw esegue la crittografia e la decrittografia asimmetriche delle chiavi dati nella memoria locale con chiavi RSA pubbliche e private fornite dall'utente. È necessario generare, archiviare e proteggere la chiave privata, preferibilmente in un modulo di sicurezza hardware (HSM) o in un sistema di gestione delle chiavi. La funzione di crittografia consente di crittografare la chiave di dati nella chiave pubblica RSA. La funzione di decrittazione consente di decrittare la chiave di dati con la chiave privata. Puoi scegliere tra i diverse modalità di padding RSA.

Un keyring non elaborato RSA che esegue crittografia e decrittazione deve includere una coppia di chiavi pubblica e privata asimmetriche. Tuttavia, puoi crittografare i dati con un portachiavi RSA Raw con solo una chiave pubblica e puoi decrittografare i dati con un portachiavi RSA Raw con solo una chiave privata. Puoi includere qualsiasi portachiavi Raw RSA in un portachiavi multiplo. Se configuri un portachiavi RSA Raw con una chiave pubblica e privata, assicurati che facciano parte della stessa coppia di chiavi.

Il portachiavi RSA Raw è equivalente e interagisce con esso SDK di crittografia AWS per Java quando viene utilizzato con chiavi di crittografia asimmetriche RSA. JceMasterKey

> **Note**
>
> Il portachiavi Raw RSA non supporta chiavi KMS asimmetriche. Per utilizzare chiavi RSA KMS asimmetriche, costruisci un portachiavi. AWS KMS

Namespace e nomi

Per identificare il materiale chiave RSA in un portachiavi, il portachiavi RSA Raw utilizza uno spazio dei nomi chiave e un nome chiave forniti dall'utente. Questi valori non sono segreti. Appaiono in testo normale nella descrizione del materiale che AWS Database Encryption SDK aggiunge al record. Si consiglia di utilizzare lo spazio dei nomi e il nome chiave che identificano la coppia di chiavi RSA (o la relativa chiave privata) nel sistema HSM o di gestione delle chiavi.

> **Note**
>
> Lo spazio dei nomi e il nome chiave sono equivalenti ai campi Provider ID (o Provider) e Key ID in. `JceMasterKey`

Se costruisci portachiavi diversi per crittografare e decrittografare un determinato record, lo spazio dei nomi e i valori dei nomi sono fondamentali. Se lo spazio dei nomi della chiave e il nome della chiave nel portachiavi di decrittografia non corrispondono esattamente, con distinzione tra maiuscole e minuscole, allo spazio dei nomi della chiave e al nome della chiave nel portachiavi di crittografia, il portachiavi di decrittografia non viene utilizzato, anche se le chiavi provengono dalla stessa coppia di chiavi.

Lo spazio dei nomi e il nome chiave del materiale chiave nei portachiavi di crittografia e decrittografia devono essere gli stessi indipendentemente dal fatto che il portachiavi contenga la chiave pubblica RSA, la chiave privata RSA o entrambe le chiavi nella coppia di chiavi. Ad esempio, supponiamo di crittografare i dati con un portachiavi RSA non elaborato per una chiave pubblica RSA con namespace e nome chiave. `HSM_01` `RSA_2048_06` Per decrittografare i dati, crea un portachiavi RSA Raw con la chiave privata (o coppia di chiavi) e lo stesso spazio dei nomi e lo stesso nome della chiave.

modalità Padding

È necessario specificare una modalità di riempimento per i portachiavi RSA Raw utilizzati per la crittografia e la decrittografia oppure utilizzare le funzionalità dell'implementazione del linguaggio che la specificano per te.

AWS Encryption SDKSupporta le seguenti modalità di riempimento, soggette ai vincoli di ogni lingua. Consigliamo una modalità di riempimento OAEP, in particolare OAEP con SHA-256 e MGF1 con imbottitura SHA-256. La modalità di riempimento PKCS1 è supportata solo per la compatibilità con le versioni precedenti.

- OAEP con SHA-1 e MGF1 con imbottitura SHA-1
- OAEP con SHA-256 e MGF1 con imbottitura SHA-256
- OAEP con SHA-384 e MGF1 con imbottitura SHA-384
- OAEP con SHA-512 e MGF1 con imbottitura SHA-512
- Imbottitura PKCS1 v1.5

L'esempio Java seguente mostra come creare un portachiavi RSA Raw con la chiave pubblica e privata di una coppia di chiavi RSA e l'OAEP con SHA-256 e MGF1 con la modalità padding SHA-256. Le `RSAPrivateKey` variabili `RSAPublicKey` e rappresentano il materiale chiave che fornisci.

```
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);
```

Keyring multipli

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

È possibile combinare più keyring in un keyring multiplo. Un keyring multiplo è composto da uno o più keyring dello stesso tipo o di tipi diversi. Il risultato è analogo a quello ottenuto utilizzando diversi keyring in serie. Quando utilizzi un keyring multiplo per crittografare i dati, questi possono essere decrittati con le chiavi di wrapping contenute in qualsiasi keyring.

Quando crei un keyring multiplo per crittografare i dati, uno dei keyring viene designato come keyring generatore, tutti gli altri keyring sono i keyring figlio. che si occupa di generare e crittografare la chiave di dati di testo normale. Quindi, tutte le chiavi di wrapping in tutti i keyring figlio crittografano la stessa chiave di dati di testo normale. Il keyring multiplo restituisce la chiave di dati di testo normale e una chiave di dati crittografata per ciascuna chiave di wrapping nel keyring multiplo. Se il portachiavi del generatore è un portachiavi KMS, la chiave del generatore nel AWS KMS portachiavi genera e crittografa la chiave in testo normale. Quindi, tutte le chiavi aggiuntive AWS KMS keys nel AWS KMS portachiavi e tutte le chiavi di avvolgimento in tutti i portachiavi per bambini nel portachiavi multiplo, crittografano la stessa chiave in chiaro.

Durante la decrittografia, AWS Database Encryption SDK utilizza i portachiavi per provare a decrittografare una delle chiavi di dati crittografate. I keyring sono chiamati nell'ordine in cui sono specificati nel keyring multiplo. L'elaborazione si interrompe non appena una chiave in qualsiasi keyring può decrittare una chiave di dati crittografata.

Per creare un keyring multiplo, crea prima un'istanza dei keyring figlio. In questo esempio di Java, utilizziamo un AWS KMS portachiavi e un portachiavi Raw AES, ma puoi combinare tutti i portachiavi supportati in un portachiavi multiplo.

```
// 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);
```

Crea quindi il keyring multiplo e specifica il keyring generatore, se presente. In questo esempio, creiamo un portachiavi multiplo in cui il portachiavi è il AWS KMS portachiavi del generatore e il portachiavi AES è il portachiavi del bambino.

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

Ora puoi utilizzare il keyring multiplo per crittografare e decrittare i dati.

Crittografia ricercabile

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce ancora informazioni sul client di crittografia DynamoDB.

La crittografia ricercabile consente di eseguire ricerche nei record crittografati senza decrittografare l'intero database. Ciò viene ottenuto utilizzando i beacon, che creano una mappa tra il valore di testo in chiaro scritto in un campo e il valore crittografato effettivamente archiviato nel database. Il AWS Database Encryption SDK archivia il beacon in un nuovo campo che aggiunge al record. A seconda del tipo di beacon che utilizzi, puoi eseguire ricerche con corrispondenza esatta o interrogazioni complesse più personalizzate sui tuoi dati crittografati.

> **Note**
>
> La crittografia ricercabile in AWS Database Encryption SDK differisce dalla crittografia simmetrica ricercabile definita nella ricerca accademica, come la crittografia simmetrica ricercabile.

Un beacon è un tag HMAC (Hash-Based Message Authentication Code) troncato che crea una mappa tra il testo in chiaro e i valori crittografati di un campo. Quando scrivi un nuovo valore in un campo crittografato configurato per la crittografia ricercabile, AWS Database Encryption SDK calcola un HMAC rispetto al valore di testo normale. Questo output HMAC corrisponde uno a uno (1:1) per il valore in testo normale di quel campo. L'output HMAC viene troncato in modo che più valori di testo in chiaro distinti vengano mappati allo stesso tag HMAC troncato. Questi falsi positivi limitano la capacità di un utente non autorizzato di identificare informazioni distintive sul valore del testo in chiaro. Quando esegui una query su un beacon, AWS Database Encryption SDK filtra automaticamente questi falsi positivi e restituisce il risultato in testo non crittografato della tua richiesta.

Il numero medio di falsi positivi generati per ogni beacon è determinato dalla lunghezza residua del beacon dopo il troncamento. Per aiutarti a determinare la lunghezza del beacon appropriata per la tua implementazione, vedi Determinazione della lunghezza del beacon.

> **Note**
>
> La crittografia ricercabile è progettata per essere implementata in nuovi database non popolati. Qualsiasi beacon configurato in un database esistente mapperà solo i nuovi record caricati nel database, non è possibile per un beacon di mappare i dati esistenti.

Argomenti

I beacon sono adatti al mio set di dati?

L'uso dei beacon per eseguire query su dati crittografati riduce i costi di prestazioni associati ai database crittografati lato client. Quando si utilizzano i beacon, esiste un compromesso intrinseco tra l'efficienza delle query e la quantità di informazioni rivelate sulla distribuzione dei dati. Il beacon non altera lo stato crittografato del campo. Quando si crittografa e si firma un campo con AWS Database Encryption SDK, il valore in testo normale del campo non viene mai esposto al database. Il database memorizza il valore randomizzato e crittografato del campo.

I beacon vengono archiviati insieme ai campi crittografati da cui vengono calcolati. Ciò significa che anche se un utente non autorizzato non è in grado di visualizzare i valori di testo non crittografato di un campo crittografato, potrebbe essere in grado di eseguire analisi statistiche sui beacon per saperne di più sulla distribuzione del set di dati e, in casi estremi, identificare i valori di testo non crittografato a cui un beacon è mappato. Il modo in cui configuri i tuoi beacon può mitigare questi rischi. In particolare, la scelta della giusta lunghezza del beacon può aiutarti a preservare la riservatezza del tuo set di dati.

Sicurezza e prestazioni

- Più breve è la lunghezza del faro, maggiore è la sicurezza.
- Maggiore è la lunghezza del faro, maggiori sono le prestazioni.

La crittografia ricercabile potrebbe non essere in grado di fornire i livelli desiderati di prestazioni e sicurezza per tutti i set di dati. Esamina il tuo modello di minaccia, i requisiti di sicurezza e le esigenze prestazionali prima di configurare qualsiasi beacon.

Considera i seguenti requisiti di unicità del set di dati per determinare se la crittografia ricercabile è adatta al tuo set di dati.

Distribution (Distribuzione)

La quantità di sicurezza preservata da un beacon dipende dalla distribuzione del set di dati. Quando configuri un campo crittografato per la crittografia ricercabile, AWS Database Encryption SDK calcola un HMAC sui valori in chiaro scritti in quel campo. Tutti i beacon calcolati per un determinato campo vengono calcolati utilizzando la stessa chiave, ad eccezione dei database multitenant che utilizzano una chiave distinta per ogni tenant. Ciò significa che se lo stesso valore di testo non crittografato viene scritto più volte nel campo, viene creato lo stesso tag HMAC per ogni istanza di quel valore di testo non crittografato.

Dovresti evitare di costruire beacon da campi che contengono valori molto comuni. Ad esempio, si consideri un database che memorizza l'indirizzo di ogni residente dello stato dell'Illinois. Se costruisci un beacon dal `City` campo crittografato, il beacon calcolato su «Chicago» sarà sovrarappresentato a causa della grande percentuale della popolazione dell'Illinois che vive a Chicago. Anche se un utente non autorizzato può leggere solo i valori crittografati e i valori dei beacon, potrebbe essere in grado di identificare quali record contengono dati per i residenti di Chicago se il beacon mantiene questa distribuzione. Per ridurre al minimo la quantità di informazioni distintive rivelate sulla tua distribuzione, devi troncare sufficientemente il tuo beacon. La lunghezza del beacon necessaria per nascondere questa distribuzione irregolare comporta costi di prestazioni significativi che potrebbero non soddisfare le esigenze dell'applicazione.

È necessario analizzare attentamente la distribuzione del set di dati per determinare la quantità di beacon da troncare. La lunghezza residua del beacon dopo il troncamento è direttamente correlata alla quantità di informazioni statistiche che è possibile identificare sulla distribuzione. Potrebbe essere necessario scegliere lunghezze di beacon più corte per ridurre al minimo la quantità di informazioni distintive rivelate sul set di dati.

In casi estremi, non è possibile calcolare la lunghezza di un beacon per un set di dati distribuito in modo non uniforme che bilancia efficacemente prestazioni e sicurezza. Ad esempio, non dovresti costruire un faro da un campo che contiene i risultati di un test medico per una malattia rara. Poiché si prevede che `NEGATIVE` i risultati siano significativamente più diffusi all'interno del set di dati, i `POSITIVE` risultati possono essere facilmente identificati in base alla loro rarità. È molto difficile nascondere la distribuzione quando il campo ha solo due valori possibili. Se utilizzate una lunghezza del beacon sufficientemente corta da nascondere la distribuzione, tutti i valori di testo in chiaro sono mappati allo stesso tag HMAC. Se si utilizza una lunghezza del beacon maggiore, è ovvio quali beacon vengono mappati ai valori di testo in chiaro. `POSITIVE`

Correlazione

Ti consigliamo vivamente di evitare di costruire beacon distinti da campi con valori correlati. I beacon costruiti a partire da campi correlati richiedono lunghezze di beacon più brevi per ridurre al minimo la quantità di informazioni rivelate sulla distribuzione di ciascun set di dati a un utente non autorizzato. È necessario analizzare attentamente il set di dati, inclusa la sua entropia e la distribuzione congiunta dei valori correlati, per determinare la quantità di beacon da troncare. Se la lunghezza del beacon risultante non soddisfa le tue esigenze di prestazioni, i beacon potrebbero non essere adatti al tuo set di dati.

Ad esempio, non dovresti creare due beacon separati da `City` e `ZIPCode` campi perché il codice postale sarà probabilmente associato a una sola città. In genere, i falsi positivi generati da un beacon limitano la capacità di un utente non autorizzato di identificare informazioni distintive sul set di dati. Ma la correlazione tra i `ZIPCode` campi `City` e significa che un utente non autorizzato può facilmente identificare quali risultati sono falsi positivi e distinguere i diversi codici postali.

Dovresti anche evitare di costruire beacon a partire da campi che contengono gli stessi valori di testo in chiaro. Ad esempio, non dovresti creare un beacon da `mobilePhone` e `preferredPhone` campi perché probabilmente contengono gli stessi valori. Se costruisci beacon distinti da entrambi i campi, AWS Database Encryption SDK crea i beacon per ogni campo con chiavi diverse. Ciò si traduce in due diversi tag HMAC per lo stesso valore di testo non crittografato. È improbabile che i due distinti beacon abbiano gli stessi falsi positivi e un utente non autorizzato potrebbe essere in grado di distinguere numeri di telefono diversi.

Anche se il tuo set di dati contiene campi correlati o ha una distribuzione non uniforme, potresti essere in grado di costruire beacon che preservino la riservatezza del tuo set di dati utilizzando lunghezze di beacon più brevi. Tuttavia, la lunghezza del beacon non garantisce che ogni valore univoco nel set di dati produca una serie di falsi positivi che riducano efficacemente al minimo la quantità di informazioni distintive rivelate sul set di dati. La lunghezza del beacon stima solo il numero medio di falsi positivi prodotti. Più il set di dati è distribuito in modo non uniforme, minore è la lunghezza effettiva del beacon nel determinare il numero medio di falsi positivi prodotti.

Valuta attentamente la distribuzione dei campi da cui costruisci i beacon e considera quanto ti servirà per troncare la lunghezza del faro per soddisfare i tuoi requisiti di sicurezza. I seguenti argomenti di questo capitolo presuppongono che i beacon siano distribuiti uniformemente e non contengano dati correlati.

Scenario di crittografia ricercabile

L'esempio seguente mostra una soluzione di crittografia semplice e ricercabile. Nell'applicazione, i campi di esempio utilizzati in questo esempio potrebbero non soddisfare le raccomandazioni sull'unicità della distribuzione e della correlazione per i beacon. Puoi usare questo esempio come riferimento mentre leggi i concetti di crittografia ricercabili in questo capitolo.

Prendi in considerazione un database denominato `Employees` che tiene traccia dei dati dei dipendenti di un'azienda. Ogni record del database contiene campi denominati EmployeeID, LastNameFirstName, e Address. Ogni campo del `Employees` database è identificato dalla chiave primaria`EmployeeID`.

Di seguito è riportato un esempio di record in testo normale nel database.

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

Se hai contrassegnato i `FirstName` campi `LastName` and come `ENCRYPT_AND_SIGN` nelle tue azioni crittografiche, i valori in questi campi vengono crittografati localmente prima di essere caricati nel database. I dati crittografati caricati sono completamente randomizzati, il database non riconosce questi dati come protetti. Rileva solo le tipiche immissioni di dati. Ciò significa che il record effettivamente archiviato nel database potrebbe essere simile al seguente.

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

```
    }
}
```

Se devi interrogare il database per verificare le corrispondenze esatte nel `LastName` campo, configura un beacon standard denominato LastNameper mappare i valori di testo in chiaro scritti nel `LastName` campo ai valori crittografati memorizzati nel database.

Questo beacon calcola gli HMAC dai valori di testo in chiaro presenti nel campo. `LastName` Ogni uscita HMAC viene troncata in modo che non corrisponda più esattamente al valore di testo in chiaro. Ad esempio, l'hash completo e l'hash troncato per `Jones` potrebbero essere simili ai seguenti.

Hash completo

`2aa4e9b404c68182562b6ec761fcca5306de527826a69468885e59dc36d0c3f824bdd44cab45526f`

Hash troncato

`b35099d408c833`

Dopo aver configurato il beacon standard, puoi eseguire ricerche di uguaglianza sul campo. `LastName` Ad esempio, se desideri cercare`Jones`, usa il LastNamebeacon per eseguire la seguente query.

```
LastName = Jones
```

Il AWS Database Encryption SDK filtra automaticamente i falsi positivi e restituisce il risultato in testo normale della query.

fari

> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce ancora informazioni sul client di crittografia DynamoDB.

Un beacon è un tag HMAC (Hash-Based Message Authentication Code) troncato che crea una mappa tra il valore di testo in chiaro scritto in un campo e il valore crittografato effettivamente archiviato nel database. Il beacon non altera lo stato crittografato del campo. Il beacon calcola un HMAC sul valore di testo non crittografato del campo e lo memorizza insieme al valore crittografato. Questo output HMAC corrisponde uno a uno (1:1) per il valore in testo normale di quel campo.

L'output HMAC viene troncato in modo che più valori di testo in chiaro distinti vengano mappati allo stesso tag HMAC troncato. Questi falsi positivi limitano la capacità di un utente non autorizzato di identificare informazioni distintive sul valore del testo in chiaro.

I beacon possono essere costruiti solo a partire da campi contrassegnati `ENCRYPT_AND_SIGN` o contenuti `SIGN_ONLY` nelle tue azioni crittografiche. Il beacon stesso non è firmato o crittografato. Non è possibile costruire un faro con campi contrassegnati. `DO_NOTHING`

Il tipo di beacon configurato determina il tipo di query che è possibile eseguire. Esistono due tipi di beacon che supportano la crittografia ricercabile. I beacon standard eseguono ricerche sull'uguaglianza. I beacon composti combinano stringhe letterali in chiaro e beacon standard per eseguire operazioni complesse sul database. Dopo aver configurato i beacon, devi configurare un indice secondario per ogni beacon prima di poter cercare nei campi crittografati. Per ulteriori informazioni, consulta Configurazione degli indici secondari con i beacon.

Argomenti

- Fari standard
- Fari composti

Fari standard

I beacon standard sono il modo più semplice per implementare la crittografia ricercabile nel tuo database. Possono eseguire ricerche di uguaglianza solo per un singolo campo crittografato o virtuale. Per informazioni su come configurare i beacon standard, vedere Configurazione dei beacon standard.

Il campo da cui è costruito un beacon standard è chiamato sorgente del beacon. Identifica la posizione dei dati che il beacon deve mappare. La sorgente del beacon può essere un campo crittografato o un campo virtuale. La sorgente del beacon in ogni beacon standard deve essere unica. Non è possibile configurare due beacon con la stessa sorgente di beacon.

È possibile creare un beacon standard che esegua ricerche di uguaglianza per un singolo campo crittografato oppure è possibile creare un beacon standard che esegua ricerche di uguaglianza sulla concatenazione di più campi `ENCRYPT_AND_SIGN` e `SIGN_ONLY` campi creando un campo virtuale.

Campi virtuali

Un campo virtuale è un campo concettuale costruito a partire da uno o più campi di origine. La creazione di un campo virtuale non comporta la scrittura di un nuovo campo nel record. Il campo virtuale non è archiviato in modo esplicito nel database. Viene utilizzato nella configurazione standard del beacon per fornire al beacon istruzioni su come identificare un segmento specifico di un campo o concatenare più campi all'interno di un record per eseguire una query specifica. Un campo virtuale richiede almeno un campo crittografato.

> **Note**
>
> L'esempio seguente mostra i tipi di trasformazioni e interrogazioni che è possibile eseguire con un campo virtuale. Nell'applicazione, i campi di esempio utilizzati in questo esempio potrebbero non soddisfare le raccomandazioni sull'unicità della distribuzione e della correlazione per i beacon.

Ad esempio, se desideri eseguire ricerche di uguaglianza sulla concatenazione di `LastName` campi `FirstName` e, puoi creare uno dei seguenti campi virtuali.

- Un `NameTag` campo virtuale, composto dalla prima lettera del `FirstName` campo, seguita dal `LastName` campo, tutto in minuscolo. Questo campo virtuale consente di eseguire interrogazioni`NameTag=mjones`.
- Un `LastFirst` campo virtuale, che è costruito dal `LastName` campo, seguito dal `FirstName` campo. Questo campo virtuale consente di eseguire interrogazioni`LastFirst=JonesMary`.

Oppure, se desideri eseguire ricerche di uguaglianza su un segmento specifico di un campo crittografato, crea un campo virtuale che identifichi il segmento da interrogare.

Ad esempio, se desideri interrogare un `IPAddress` campo crittografato utilizzando i primi tre segmenti dell'indirizzo IP, crea il seguente campo virtuale.

- Un `IPSegment` campo virtuale, costruito da`Segments('.', 0, 3)`. Questo campo virtuale consente di eseguire interrogazioni`IPSegment=192.0.2`. La query restituisce tutti i record con un `IPAddress` valore che inizia con «192.0.2".

I campi virtuali devono essere univoci. Non è possibile creare due campi virtuali a partire dagli stessi campi di origine.

Per assistenza nella configurazione dei campi virtuali e dei beacon che li utilizzano, consulta Creazione di un campo virtuale.

Fari composti

I beacon composti creano indici che migliorano le prestazioni delle query e consentono di eseguire operazioni di database più complesse. È possibile utilizzare i beacon composti per combinare stringhe letterali in chiaro e beacon standard per eseguire query complesse su record crittografati, ad esempio interrogare due diversi tipi di record da un singolo indice o interrogare una combinazione di campi con una chiave di ordinamento. Per altri esempi di soluzioni di beacon composte, consulta Scegliere un tipo di beacon.

I beacon composti possono essere costruiti a partire da beacon standard o da una combinazione di beacon e campi standard. `SIGN_ONLY` Sono costituiti da un elenco di parti. Tutti i beacon composti devono includere un elenco di parti crittografate che identifichi `ENCRYPT_AND_SIGN` i campi inclusi nel beacon. Ogni `ENCRYPT_AND_SIGN` campo deve essere identificato da un faro standard. I beacon composti più complessi possono includere anche un elenco di parti firmate che identifica i `SIGN_ONLY` campi di testo in chiaro inclusi nel beacon e un elenco di parti del costruttore che identificano tutti i possibili modi in cui il beacon composto può assemblare i campi.

> **Note**
>
> Il AWS Database Encryption SDK supporta anche beacon firmati che possono essere configurati interamente da campi di testo normale. `SIGN_ONLY` I beacon firmati sono un tipo di beacon composto che indicizza ed esegue query complesse sui campi. `SIGN_ONLY` Per ulteriori informazioni, consulta Creazione di beacon firmati.

Per assistenza nella configurazione dei beacon composti, vedere Configurazione dei beacon composti.

Il modo in cui configuri il tuo beacon composto determina i tipi di query che può eseguire. Ad esempio, puoi rendere opzionali alcune parti crittografate e firmate per consentire una maggiore flessibilità nelle tue domande. Per ulteriori informazioni sui tipi di query che i beacon composti possono eseguire, vedere. Interrogazione dei beacon

Pianificazione dei beacon

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce ancora informazioni sul client di crittografia DynamoDB.

I beacon sono progettati per essere implementati in nuovi database non popolati. Qualsiasi beacon configurato in un database esistente mapperà solo i nuovi record scritti nel database. I beacon vengono calcolati in base al valore di testo in chiaro di un campo, una volta che il campo è crittografato, non è possibile per il beacon di mappare i dati esistenti. Dopo aver scritto nuovi record con un beacon, non è possibile aggiornare la configurazione del beacon. Tuttavia, puoi aggiungere nuovi beacon per i nuovi campi che aggiungi al tuo record.

Per implementare la crittografia ricercabile, è necessario utilizzare il portachiavi AWS KMS gerarchico per generare, crittografare e decrittografare le chiavi dati utilizzate per proteggere i record. Per ulteriori informazioni, consulta Utilizzo del portachiavi Hierarchical per una crittografia ricercabile.

Prima di poter configurare i beacon per la crittografia ricercabile, è necessario esaminare i requisiti di crittografia, i modelli di accesso al database e il modello di minaccia per determinare la soluzione migliore per il database.

Il tipo di beacon configurato determina il tipo di query che è possibile eseguire. La lunghezza del beacon specificata nella configurazione standard del beacon determina il numero previsto di falsi positivi prodotti per un determinato beacon. Ti consigliamo vivamente di identificare e pianificare i tipi di query da eseguire prima di configurare i beacon. Una volta utilizzato un beacon, la configurazione non può essere aggiornata.

Ti consigliamo vivamente di esaminare e completare le seguenti attività prima di configurare qualsiasi beacon.

- Determina se i beacon sono adatti al tuo set di dati
- Scegli un tipo di faro
- Scegli la lunghezza del faro
- Scegli un nome per il beacon

Ricorda i seguenti requisiti di unicità dei beacon quando pianifichi la soluzione di crittografia ricercabile per il tuo database.

- Ogni beacon standard deve avere una sorgente di beacon unica

  Non è possibile creare più beacon standard a partire dallo stesso campo crittografato o virtuale.

  Tuttavia, è possibile utilizzare un singolo faro standard per costruire più beacon composti.
- Evita di creare un campo virtuale con campi sorgente che si sovrappongono ai beacon standard esistenti

La costruzione di un beacon standard da un campo virtuale che contiene un campo sorgente utilizzato per creare un altro beacon standard può ridurre la sicurezza di entrambi i beacon.

Per ulteriori informazioni, consulta Considerazioni sulla sicurezza per i campi virtuali.

Considerazioni per i database multitenant

Per interrogare i beacon configurati in un database multitenant, è necessario includere nella query il campo che memorizza i dati `branch-key-id` associati al tenant che ha crittografato il record. Questo campo viene definito quando si definisce la sorgente della chiave del beacon. Affinché la query abbia esito positivo, il valore in questo campo deve identificare i materiali chiave del beacon appropriati necessari per ricalcolare il beacon.

Prima di configurare i beacon, devi decidere come intendi includerli `branch-key-id` nelle tue query. Per ulteriori informazioni sui diversi modi in cui puoi includerli `branch-key-id` nelle tue domande, vediInterrogazione dei beacon in un database multitenant.

Scelta del tipo di faro

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce ancora informazioni sul client di crittografia DynamoDB.

Con la crittografia ricercabile, puoi cercare nei record crittografati mappando i valori di testo in chiaro in un campo crittografato con un beacon. Il tipo di beacon configurato determina il tipo di query che è possibile eseguire.

Ti consigliamo vivamente di identificare e pianificare i tipi di query da eseguire prima di configurare i beacon. Dopo aver configurato i beacon, è necessario configurare un indice secondario per ogni beacon prima di poter eseguire la ricerca nei campi crittografati. Per ulteriori informazioni, consulta Configurazione degli indici secondari con i beacon.

I beacon creano una mappa tra il valore di testo in chiaro scritto in un campo e il valore crittografato effettivamente archiviato nel database. Non è possibile confrontare i valori di due beacon standard, anche se contengono lo stesso testo in chiaro sottostante. I due beacon standard produrranno due diversi tag HMAC per gli stessi valori di testo in chiaro. Di conseguenza, i beacon standard non possono eseguire le seguenti domande.

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

È possibile eseguire le query precedenti solo se si confrontano le parti firmate dei beacon composti, ad eccezione dell'CONTAINSoperatore, che è possibile utilizzare con i beacon composti per identificare l'intero valore di un campo crittografato o firmato contenuto nel beacon assemblato. Quando si confrontano parti firmate, è possibile includere facoltativamente il prefisso di una parte crittografata, ma non il valore crittografato di un campo. Per ulteriori informazioni sui tipi di query che possono eseguire i beacon standard e compositi, vedere Interrogare i beacon.

Considerate le seguenti soluzioni di crittografia ricercabili mentre esaminate i modelli di accesso al database. Gli esempi seguenti definiscono quale beacon configurare per soddisfare diversi requisiti di crittografia e interrogazione.

Fari standard

I beacon standard possono eseguire solo ricerche di uguaglianza. È possibile utilizzare i beacon standard per eseguire le seguenti domande.

Interroga un singolo campo crittografato

Se desideri identificare i record che contengono un valore specifico per un campo crittografato, crea un beacon standard.

Esempi

Per l'esempio seguente, si consideri un database denominato `UnitInspection` che tiene traccia dei dati di ispezione per un impianto di produzione. Ogni record nel database contiene campi denominati `work_id`,`inspection_date`,`inspector_id_last4`, e`unit`. L'ID completo dell'ispettore è un numero compreso tra 0 e 99.999.999. Tuttavia, per garantire che il set di dati sia distribuito uniformemente, memorizza `inspector_id_last4` solo le ultime quattro cifre dell'ID dell'ispettore. Ogni campo del database è identificato dalla chiave primaria`work_id`. I `unit` campi `inspector_id_last4` e sono contrassegnati `ENCRYPT_AND_SIGN` nelle azioni crittografiche.

Di seguito è riportato un esempio di voce di testo non crittografato nel `UnitInspection` database.

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

Interroga un singolo campo crittografato in un record

Se il `inspector_id_last4` campo deve essere crittografato, ma hai comunque bisogno di interrogarlo per trovare le corrispondenze esatte, costruisci un beacon standard dal campo. `inspector_id_last4` Quindi, usa il beacon standard per creare un indice secondario. È possibile utilizzare questo indice secondario per eseguire una ricerca sul `inspector_id_last4` campo crittografato.

Per assistenza nella configurazione dei beacon standard, vedere Configurazione dei beacon standard.

Interroga un campo virtuale

Un campo virtuale è un campo concettuale costruito a partire da uno o più campi di origine. Se desideri eseguire ricerche di uguaglianza per un segmento specifico di un campo crittografato o eseguire ricerche di uguaglianza sulla concatenazione di più campi, costruisci un beacon standard da un campo virtuale. Tutti i campi virtuali devono includere almeno un campo sorgente crittografato.

Esempi

Gli esempi seguenti creano campi virtuali per il `Employees` database. Di seguito è riportato un esempio di record in testo normale nel `Employees` database.

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

```
    }
}
```

Interroga un segmento di un campo crittografato

In questo esempio, il `SSN` campo è crittografato.

Se desideri interrogare il `SSN` campo utilizzando le ultime quattro cifre di un numero di previdenza sociale, crea un campo virtuale che identifichi il segmento che intendi interrogare.

Un `Last4SSN` campo virtuale, creato da, `Suffix(4)` consente di eseguire interrogazioni`Last4SSN=0000`. Usa questo campo virtuale per costruire un faro standard. Quindi, usa il beacon standard per creare un indice secondario. È possibile utilizzare questo indice secondario per eseguire una ricerca sul campo virtuale. Questa query restituisce tutti i record con un `SSN` valore che termina con le ultime quattro cifre specificate.

Interroga la concatenazione di più campi

> **Note**
>
> L'esempio seguente mostra i tipi di trasformazioni e interrogazioni che è possibile eseguire con un campo virtuale. Nell'applicazione, i campi di esempio utilizzati in questo esempio potrebbero non soddisfare le raccomandazioni sull'unicità della distribuzione e della correlazione per i beacon.

Se desideri eseguire ricerche di uguaglianza su una concatenazione di `LastName` campi `FirstName` e, puoi creare un `NameTag` campo virtuale, costruito dalla prima lettera del `FirstName` campo, seguita dal `LastName` campo, tutto in minuscolo. Usa questo campo virtuale per costruire un faro standard. Quindi, usa il beacon standard per creare un indice secondario. È possibile utilizzare questo indice secondario per eseguire una ricerca `NameTag=mjones` sul campo virtuale.

Almeno uno dei campi di origine deve essere crittografato. `FirstName`O `LastName` potrebbero essere crittografati, oppure entrambi potrebbero essere crittografati. Tutti i campi di origine in testo normale devono essere contrassegnati come `SIGN_ONLY` nelle azioni crittografiche.

Per assistenza nella configurazione dei campi virtuali e dei beacon che li utilizzano, consulta Creazione di un campo virtuale.

Fari composti

I beacon composti creano un indice da stringhe letterali in chiaro e beacon standard per eseguire operazioni complesse sul database. È possibile utilizzare i beacon composti per eseguire le seguenti domande.

Interroga una combinazione di campi crittografati su un singolo indice

Se devi interrogare una combinazione di campi crittografati su un singolo indice, crea un beacon composto che combini i singoli beacon standard costruiti per ogni campo crittografato per formare un singolo indice.

Dopo aver configurato il beacon composto, è possibile creare un indice secondario che specifichi il beacon composto come chiave di partizione per eseguire query con corrispondenza esatta o con una chiave di ordinamento per eseguire query più complesse. Gli indici secondari che specificano il beacon composto come chiave di ordinamento possono eseguire query con corrispondenza esatta e query complesse più personalizzate.

Esempi

Per gli esempi seguenti, si consideri un database denominato `UnitInspection` che tiene traccia dei dati di ispezione per un impianto di produzione. Ogni record nel database contiene campi denominati `work_id`,`inspection_date`,`inspector_id_last4`, e`unit`. L'ID completo dell'ispettore è un numero compreso tra 0 e 99.999.999. Tuttavia, per garantire che il set di dati sia distribuito uniformemente, memorizza `inspector_id_last4` solo le ultime quattro cifre dell'ID dell'ispettore. Ogni campo del database è identificato dalla chiave primaria`work_id`. I `unit` campi `inspector_id_last4` e sono contrassegnati `ENCRYPT_AND_SIGN` nelle azioni crittografiche.

Di seguito è riportato un esempio di voce di testo non crittografato nel `UnitInspection` database.

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

Esegui ricerche di uguaglianza su una combinazione di campi crittografati

Se desideri interrogare il `UnitInspection` database per individuare le corrispondenze esatte`inspector_id_last4.unit`, crea innanzitutto dei beacon standard distinti per i campi `inspector_id_last4` and`unit`. Quindi, crea un faro composto dai due beacon standard.

Dopo aver configurato il beacon composto, create un indice secondario che specifichi il beacon composto come chiave di partizione. Usa questo indice secondario per cercare le corrispondenze esatte su`inspector_id_last4.unit`. Ad esempio, è possibile interrogare questo faro per trovare un elenco di ispezioni eseguite da un ispettore per una determinata unità.

Esegui interrogazioni complesse su una combinazione di campi crittografati

Se desideri interrogare il `UnitInspection` database su `inspector_id_last4` e`inspector_id_last4.unit`, crea innanzitutto beacon standard distinti per i campi `inspector_id_last4` and`unit`. Quindi, crea un faro composto dai due beacon standard.

Dopo aver configurato il beacon composto, create un indice secondario che specifichi il beacon composto come chiave di ordinamento. Utilizzate questo indice secondario per ricercare `UnitInspection` nel database le voci che iniziano con un determinato ispettore o interrogare il database per un elenco di tutte le unità all'interno di uno specifico intervallo di ID di unità che sono state ispezionate da un determinato ispettore. Puoi anche eseguire ricerche sulle corrispondenze esatte su`inspector_id_last4.unit`.

Per assistenza nella configurazione dei beacon composti, vedere Configurazione dei beacon composti.

Interroga una combinazione di campi crittografati e di testo normale su un singolo indice

Se devi interrogare una combinazione di campi crittografati e di testo in chiaro su un singolo indice, crea un beacon composto che combini singoli beacon standard e campi di testo in chiaro per formare un unico indice. I campi di testo in chiaro utilizzati per costruire il beacon composto devono essere contrassegnati `SIGN_ONLY` nelle azioni crittografiche.

Dopo aver configurato il beacon composto, è possibile creare un indice secondario che specifichi il beacon composto come chiave di partizione per eseguire query con corrispondenza esatta o con una chiave di ordinamento per eseguire query più complesse. Gli indici secondari che specificano il beacon composto come chiave di ordinamento possono eseguire query con corrispondenza esatta e query complesse più personalizzate.

Esempi

Per gli esempi seguenti, si consideri un database denominato `UnitInspection` che tiene traccia dei dati di ispezione per un impianto di produzione. Ogni record nel database contiene campi denominati `work_id`inspection_date,`inspector_id_last4`, e`unit`. L'ID completo dell'ispettore è un numero compreso tra 0 e 99.999.999. Tuttavia, per garantire che il set di dati sia distribuito uniformemente, memorizza `inspector_id_last4` solo le ultime quattro cifre dell'ID dell'ispettore. Ogni campo del database è identificato dalla chiave primaria`work_id`. I `unit` campi `inspector_id_last4` e sono contrassegnati `ENCRYPT_AND_SIGN` nelle azioni crittografiche.

Di seguito è riportato un esempio di voce di testo non crittografato nel `UnitInspection` database.

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

Esegui ricerche di uguaglianza su una combinazione di campi

Se desideri interrogare il `UnitInspection` database per le ispezioni condotte da un ispettore specifico in una data specifica, crea innanzitutto un faro standard per il campo. `inspector_id_last4` Il `inspector_id_last4` campo è contrassegnato `ENCRYPT_AND_SIGN` nelle azioni crittografiche. Tutte le parti crittografate richiedono il proprio beacon standard. Il `inspection_date` campo è contrassegnato `SIGN_ONLY` e non richiede un faro standard. Quindi, crea un faro composto dal `inspection_date` campo e dal faro `inspector_id_last4` standard.

Dopo aver configurato il beacon composto, create un indice secondario che specifichi il beacon composto come chiave di partizione. Utilizza questo indice secondario per interrogare i database alla ricerca di record con corrispondenze esatte con un determinato ispettore e data di ispezione. Ad esempio, è possibile interrogare il database per un elenco di tutte le ispezioni che l'ispettore il cui ID termina `8744` ha effettuato in una data specifica.

Esegui interrogazioni complesse su una combinazione di campi

Se desideri interrogare il database per le ispezioni condotte entro un `inspection_date` intervallo o interrogare il database per le ispezioni condotte su un particolare `inspection_date` vincolato da `inspector_id_last4` or`inspector_id_last4.unit`, crea innanzitutto beacon

standard distinti per i campi and. `inspector_id_last4` `unit` Quindi, crea un beacon composto dal `inspection_date` campo di testo in chiaro e dai due beacon standard.

Dopo aver configurato il beacon composto, create un indice secondario che specifichi il beacon composto come chiave di ordinamento. Utilizza questo indice secondario per eseguire domande relative alle ispezioni condotte in date specifiche da un ispettore specifico. Ad esempio, è possibile interrogare il database per un elenco di tutte le unità ispezionate nella stessa data. In alternativa, è possibile interrogare il database per un elenco di tutte le ispezioni eseguite su un'unità specifica in un determinato intervallo di date di ispezione.

Per assistenza nella configurazione dei beacon composti, vedere Configurazione dei beacon composti.

Scelta della lunghezza del faro

> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Quando scrivi un nuovo valore in un campo crittografato configurato per la crittografia ricercabile, AWS Database Encryption SDK calcola un HMAC rispetto al valore di testo normale. Questo output HMAC corrisponde uno a uno (1:1) per il valore in testo normale di quel campo. L'output HMAC viene troncato in modo che più valori di testo in chiaro distinti vengano mappati allo stesso tag HMAC troncato. Queste collisioni, o falsi positivi, limitano la capacità di un utente non autorizzato di identificare informazioni distintive sul valore in testo non crittografato.

Il numero medio di falsi positivi generati per ogni beacon è determinato dalla lunghezza residua del beacon dopo il troncamento. È necessario definire la lunghezza del beacon solo quando si configurano i beacon standard. I beacon composti utilizzano le lunghezze dei beacon standard da cui sono costruiti.

Il beacon non altera lo stato crittografato del campo. Tuttavia, quando si utilizzano i beacon, esiste un compromesso intrinseco tra l'efficienza delle query e la quantità di informazioni rivelate sulla distribuzione dei dati.

L'obiettivo della crittografia ricercabile è ridurre i costi di prestazioni associati ai database crittografati lato client utilizzando i beacon per eseguire query su dati crittografati. I beacon vengono archiviati

insieme ai campi crittografati da cui vengono calcolati. Ciò significa che possono rivelare informazioni distintive sulla distribuzione del set di dati. In casi estremi, un utente non autorizzato potrebbe essere in grado di analizzare le informazioni rivelate sulla tua distribuzione e utilizzarle per identificare il valore di testo in chiaro di un campo. La scelta della giusta lunghezza del beacon può contribuire a mitigare questi rischi e preservare la riservatezza della distribuzione.

Esamina il tuo modello di minaccia per determinare il livello di sicurezza di cui hai bisogno. Ad esempio, più persone hanno accesso al tuo database, ma non dovrebbero avere accesso ai dati in chiaro, più potresti voler proteggere la riservatezza della distribuzione del tuo set di dati. Per aumentare la riservatezza, un beacon deve generare più falsi positivi. Una maggiore riservatezza si traduce in una riduzione delle prestazioni delle query.

Sicurezza e prestazioni

- Una lunghezza del beacon troppo lunga produce un numero insufficiente di falsi positivi e potrebbe rivelare informazioni distintive sulla distribuzione del set di dati.
- Una lunghezza del beacon troppo corta produce troppi falsi positivi e aumenta il costo delle prestazioni delle query perché richiede una scansione più ampia del database.

Nel determinare la lunghezza del beacon appropriata per la tua soluzione, devi trovare una lunghezza che preservi adeguatamente la sicurezza dei tuoi dati senza influire sulle prestazioni delle tue query più del necessario. La quantità di sicurezza preservata da un beacon dipende dalla distribuzione del set di dati e dalla correlazione dei campi da cui sono costruiti i beacon. Gli argomenti seguenti presuppongono che i beacon siano distribuiti uniformemente e non contengano dati correlati.

Argomenti

Calcolo della lunghezza del faro

La lunghezza del beacon è definita in bit e si riferisce al numero di bit del tag HMAC che vengono conservati dopo il troncamento. La lunghezza del beacon consigliata varia in base alla distribuzione del set di dati, alla presenza di valori correlati e ai requisiti specifici di sicurezza e prestazioni. Se il tuo set di dati è distribuito in modo uniforme, puoi utilizzare le seguenti equazioni e procedure per identificare la lunghezza del beacon migliore per la tua implementazione. Queste equazioni stimano

solo il numero medio di falsi positivi che il beacon produrrà, non garantiscono che ogni valore univoco nel set di dati produca un numero specifico di falsi positivi.

> **Note**
>
> L'efficacia di queste equazioni dipende dalla distribuzione del set di dati. Se il tuo set di dati non è distribuito uniformemente, consulta. I beacon sono adatti al mio set di dati?
> In generale, più il set di dati si allontana da una distribuzione uniforme, più è necessario ridurre la lunghezza del beacon.

1.

Stimare la popolazione

La popolazione è il numero previsto di valori univoci nel campo da cui è costruito il beacon standard, non il numero totale previsto di valori memorizzati nel campo. Ad esempio, si consideri un `Room` campo crittografato che identifica il luogo delle riunioni dei dipendenti. Il `Room` campo dovrebbe contenere 100.000 valori totali, ma ci sono solo 50 sale diverse che i dipendenti possono prenotare per le riunioni. Ciò significa che la popolazione è 50 perché nel `Room` campo possono essere memorizzati solo 50 valori univoci possibili.

> **Note**
>
> Se il beacon standard è costruito a partire da un campo virtuale, la popolazione utilizzata per calcolare la lunghezza del faro è il numero di combinazioni univoche create dal campo virtuale.

Quando stimate la popolazione, assicuratevi di considerare la crescita prevista del set di dati. Dopo aver scritto nuovi record con il beacon, non è possibile aggiornare la lunghezza del beacon. Esamina il tuo modello di minaccia e tutte le soluzioni di database esistenti per creare una stima del numero di valori univoci che prevedi che questo campo memorizzerà nei prossimi cinque anni.

Non è necessario che la tua popolazione sia precisa. Innanzitutto, identifica il numero di valori univoci nel tuo database corrente o stima il numero di valori univoci che prevedi di memorizzare

nel primo anno. Successivamente, utilizza le seguenti domande per determinare la crescita prevista di valori unici nei prossimi cinque anni.

- Ti aspetti che i valori univoci si moltiplichino per 10?
- Ti aspetti che i valori univoci si moltiplichino per 100?
- Ti aspetti che i valori univoci si moltiplichino per 1000?

La differenza tra 50.000 e 60.000 valori univoci non è significativa e entrambi produrranno la stessa lunghezza del beacon consigliata. Tuttavia, la differenza tra 50.000 e 500.000 valori univoci influirà in modo significativo sulla lunghezza consigliata del beacon.

Valuta la possibilità di esaminare i dati pubblici sulla frequenza dei tipi di dati più comuni, come i codici postali o i cognomi. Ad esempio, ci sono 41,707 codici postali negli Stati Uniti. La popolazione utilizzata deve essere proporzionale al proprio database. Se il `ZIPCode` campo del database include dati provenienti da tutti gli Stati Uniti, è possibile definire la popolazione come 41.707, anche se al momento il `ZIPCode` campo non ha 41.707 valori univoci. Se il `ZIPCode` campo del database include solo i dati di un singolo stato e includerà sempre e solo i dati di un singolo stato, è possibile definire la popolazione come il numero totale di codici postali in quello stato anziché 41.704.

2. Calcola l'intervallo consigliato per il numero previsto di collisioni

   Per determinare la lunghezza del beacon appropriata per un determinato campo, è necessario innanzitutto identificare un intervallo appropriato per il numero previsto di collisioni. Il numero previsto di collisioni rappresenta il numero medio previsto di valori di testo in chiaro univoci mappati a un particolare tag HMAC. Il numero previsto di falsi positivi per un valore di testo non crittografato univoco è inferiore di uno al numero previsto di collisioni.

   Ti consigliamo che il numero previsto di collisioni sia maggiore o uguale a due e inferiore alla radice quadrata della tua popolazione. Le equazioni seguenti funzionano solo se la tua popolazione ha 16 o più valori univoci.

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

   Se il numero di collisioni è inferiore a due, il beacon produrrà un numero insufficiente di falsi positivi. Ne consigliamo due come numero minimo di collisioni previste perché significa che, in media, ogni valore univoco nel campo genererà almeno un falso positivo mappando un altro valore univoco.

3. Calcola l'intervallo consigliato per le lunghezze dei beacon

   Dopo aver identificato il numero minimo e massimo di collisioni previste, utilizza la seguente equazione per identificare un intervallo di lunghezze di segnalazione appropriate.

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

   Innanzitutto, calcola la lunghezza del faro in cui il numero di collisioni previste è uguale a due (il numero minimo consigliato di collisioni previste).

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

   Quindi, calcola la lunghezza del faro in cui il numero previsto di collisioni è uguale alla radice quadrata della tua popolazione (il numero massimo consigliato di collisioni previste).

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

   Si consiglia di arrotondare l'uscita prodotta da questa equazione alla lunghezza più corta del faro. Ad esempio, se l'equazione produce una lunghezza del beacon di 15,6, consigliamo di arrotondare quel valore a 15 bit invece di arrotondare a 16 bit.

4. Scegli la lunghezza del faro

   Queste equazioni identificano solo la gamma consigliata di lunghezze dei beacon per il tuo campo. Ti consigliamo di utilizzare una lunghezza del beacon inferiore per preservare la sicurezza del tuo set di dati quando possibile. Tuttavia, la lunghezza del beacon che utilizzi effettivamente è determinata dal tuo modello di minaccia. Considerate i vostri requisiti prestazionali mentre esaminate il vostro modello di minaccia per determinare la lunghezza del beacon migliore per il vostro settore.

   L'utilizzo di una lunghezza del beacon inferiore riduce le prestazioni delle query, mentre l'utilizzo di una lunghezza del beacon maggiore riduce la sicurezza. In generale, se il set di dati è distribuito in modo non uniforme o se si costruiscono beacon distinti da campi correlati, è necessario utilizzare lunghezze di beacon più brevi per ridurre al minimo la quantità di informazioni rivelate sulla distribuzione dei set di dati.

   Se esamini il tuo modello di minaccia e decidi che qualsiasi informazione distintiva rivelata sulla distribuzione di un campo non rappresenta una minaccia per la tua sicurezza generale, potresti scegliere di utilizzare una lunghezza del beacon maggiore dell'intervallo consigliato che hai

calcolato. Ad esempio, se si calcola che l'intervallo consigliato di lunghezze dei beacon per un campo sia di 9-16 bit, è possibile scegliere di utilizzare una lunghezza del beacon di 24 bit per evitare perdite di prestazioni.

Scegli con cura la lunghezza del tuo faro. Dopo aver scritto nuovi record con il beacon, non è possibile aggiornare la lunghezza del beacon.

Esempio

Si consideri un database che contrassegna il `unit` campo come `ENCRYPT_AND_SIGN` nelle azioni crittografiche. Per configurare un beacon standard per il `unit` campo, è necessario determinare il numero previsto di falsi positivi e la lunghezza del beacon per il campo. `unit`

1. Stimare la popolazione

   Dopo aver esaminato il nostro modello di minaccia e l'attuale soluzione di database, prevediamo che alla fine il `unit` campo avrà 100.000 valori univoci.

   Ciò significa che Popolazione = 100.000.

2. Calcola l'intervallo consigliato per il numero previsto di collisioni.

   Per questo esempio, il numero previsto di collisioni dovrebbe essere compreso tra 2 e 316.

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

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

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

3. Calcola l'intervallo consigliato per la lunghezza del faro.

   Per questo esempio, la lunghezza del beacon deve essere compresa tra 9 e 16 bit.

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

   a. Calcola la lunghezza del faro in cui il numero previsto di collisioni è uguale al minimo identificato nella fase 2.

```
2 = 100,000 * 2^-(beacon length)
```

Lunghezza del beacon = 15,6 o 15 bit

b. Calcola la lunghezza del faro in cui il numero previsto di collisioni è uguale al massimo identificato nella fase 2.

```
316 = 100,000 * 2^-(beacon length)
```

Lunghezza del beacon = 8,3 o 8 bit

4. Determina la lunghezza del beacon appropriata per i tuoi requisiti di sicurezza e prestazioni.

   Per ogni bit inferiore a 15, il costo delle prestazioni e la sicurezza raddoppiano.

   - 16 bit
     - In media, ogni valore univoco corrisponderà ad altre 1,5 unità.
     - Sicurezza: due record con lo stesso tag HMAC troncato hanno il 66% di probabilità di avere lo stesso valore di testo non crittografato.
     - Prestazioni: una query recupererà 15 record ogni 10 record effettivamente richiesti.
   - 14 bit
     - In media, ogni valore univoco corrisponderà ad altre 6,1 unità.
     - Sicurezza: due record con lo stesso tag HMAC troncato hanno il 33% di probabilità di avere lo stesso valore di testo in chiaro.
     - Prestazioni: una query recupererà 30 record ogni 10 record effettivamente richiesti.

Scelta del nome del beacon

> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Ogni beacon è identificato da un nome univoco. Una volta configurato un beacon, il nome del beacon è il nome che usi quando richiedi un campo crittografato. Il nome di un beacon può avere lo stesso

nome di un campo crittografato o virtuale, ma non può avere lo stesso nome di un campo non crittografato. Due beacon diversi non possono avere lo stesso nome.

Per esempi che dimostrano come denominare e configurare i beacon, vedere Configurazione dei beacon.

Denominazione del beacon standard

Quando assegnate un nome ai beacon standard, vi consigliamo vivamente di far sì che il nome del beacon venga ricondotto alla sorgente del beacon ogni volta che è possibile. Ciò significa che il nome del beacon e il nome del campo crittografato o virtuale da cui è costruito il beacon standard sono gli stessi. Ad esempio, se stai creando un beacon standard per un campo crittografato denominato`LastName`, dovrebbe esserlo anche il nome del beacon. `LastName`

Quando il nome del beacon è uguale alla sorgente del beacon, puoi omettere la sorgente del beacon dalla tua configurazione e AWS Database Encryption SDK utilizzerà automaticamente il nome del beacon come sorgente del beacon.

Configurazione dei beacon

> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Esistono due tipi di beacon che supportano la crittografia ricercabile. I beacon standard eseguono ricerche sull'uguaglianza. Sono il modo più semplice per implementare la crittografia ricercabile nel tuo database. I beacon composti combinano stringhe letterali in chiaro e beacon standard per eseguire query più complesse.

I beacon sono progettati per essere implementati in nuovi database non popolati. Qualsiasi beacon configurato in un database esistente mapperà solo i nuovi record scritti nel database. I beacon vengono calcolati in base al valore di testo in chiaro di un campo, una volta che il campo è crittografato, non è possibile per il beacon di mappare i dati esistenti. Dopo aver scritto nuovi record con un beacon, non è possibile aggiornare la configurazione del beacon. Tuttavia, puoi aggiungere nuovi beacon per i nuovi campi che aggiungi al tuo record.

Dopo aver determinato i modelli di accesso, la configurazione dei beacon dovrebbe essere il secondo passaggio dell'implementazione del database. Quindi, dopo aver configurato tutti i beacon, è necessario creare un portachiavi AWS KMS gerarchico, definire la versione del beacon, configurare un indice secondario per ogni beacon, definire le azioni crittografiche e configurare il database e il client Database Encryption SDK. AWS Per ulteriori informazioni, vedere Utilizzo dei beacon.

Per semplificare la definizione della versione del beacon, consigliamo di creare elenchi per beacon standard e composti. Aggiungi ogni beacon che crei al rispettivo elenco di beacon standard o composto mentre li configuri.

Argomenti

Configurazione dei beacon standard

I beacon standard sono il modo più semplice per implementare la crittografia ricercabile nel tuo database. Possono eseguire ricerche di uguaglianza solo per un singolo campo crittografato o virtuale.

Esempio di sintassi di configurazione

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

Per configurare un beacon standard, fornisci i seguenti valori.

Nome del faro

Il nome che usi per interrogare un campo crittografato.

Il nome di un beacon può avere lo stesso nome di un campo crittografato o virtuale, ma non può avere lo stesso nome di un campo non crittografato. Si consiglia vivamente di utilizzare il nome

del campo crittografato o del campo virtuale da cui è costruito il beacon standard ogni volta che è possibile. Due beacon diversi non possono avere lo stesso nome. Per aiutarti a determinare il nome del beacon migliore per la tua implementazione, vedi Scelta del nome del beacon.

Lunghezza del faro

Il numero di bit del valore hash del beacon che vengono conservati dopo il troncamento.

La lunghezza del beacon determina il numero medio di falsi positivi prodotti da un determinato beacon. Per ulteriori informazioni e per determinare la lunghezza del beacon appropriata per l'implementazione, vedere Determinazione della lunghezza del beacon.

Fonte beacon (opzionale)

Il campo da cui è costruito un faro standard.

La fonte del beacon deve essere un nome di campo o un indice che si riferisce al valore di un campo annidato. Quando il nome del beacon è uguale alla sorgente del beacon, puoi omettere la sorgente del beacon dalla tua configurazione e AWS Database Encryption SDK utilizzerà automaticamente il nome del beacon come sorgente del beacon.

Creare un campo virtuale

Per creare un campo virtuale, è necessario fornire un nome per il campo virtuale e un elenco dei campi di origine. L'ordine in cui si aggiungono i campi di origine all'elenco delle parti virtuali determina l'ordine in cui vengono concatenati per creare il campo virtuale. L'esempio seguente concatena due campi di origine nella loro interezza per creare un campo virtuale.

```
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);
```

Per creare un campo virtuale con un segmento specifico di un campo di origine, è necessario definire tale trasformazione prima di aggiungere il campo di origine all'elenco delle parti virtuali.

Considerazioni sulla sicurezza per i campi virtuali

I beacon non alterano lo stato crittografato del campo. Tuttavia, quando si utilizzano i beacon, esiste un compromesso intrinseco tra l'efficienza delle query e la quantità di informazioni rivelate sulla distribuzione dei dati. Il modo in cui configuri il tuo beacon determina il livello di sicurezza che viene mantenuto da quel beacon.

Evita di creare un campo virtuale con campi sorgente che si sovrappongono ai beacon standard esistenti. La creazione di campi virtuali che includono un campo sorgente già utilizzato per creare un beacon standard può ridurre il livello di sicurezza per entrambi i beacon. La misura in cui la sicurezza viene ridotta dipende dal livello di entropia aggiunto dai campi di origine aggiuntivi. Il livello di entropia è determinato dalla distribuzione di valori univoci nel campo sorgente aggiuntivo e dal numero di bit che il campo sorgente aggiuntivo contribuisce alla dimensione complessiva del campo virtuale.

Puoi utilizzare la popolazione e la lunghezza del beacon per determinare se i campi di origine di un campo virtuale preservano la sicurezza del tuo set di dati. La popolazione è il numero previsto di valori univoci in un campo. Non è necessario che la tua popolazione sia precisa. Per informazioni sulla stima della popolazione di un campo, vedi Stima della popolazione.

Considera il seguente esempio mentre esamini la sicurezza dei tuoi campi virtuali.

- Beacon1 è costruito da. `FieldA` `FieldA`ha una popolazione superiore a $2^{(\text{lunghezza Beacon1})}$.
- Beacon2 è costruito da`VirtualField`, che è costruito da`FieldA`, `FieldB``FieldC`, e. `FieldD` Insieme, `FieldB``FieldC`, e `FieldD` hanno una popolazione superiore a 2^{N}

Beacon2 preserva la sicurezza sia di Beacon1 che di Beacon2 se le seguenti affermazioni sono vere:

```
N ≥ (Beacon1 length)/2
```

e

```
N ≥ (Beacon2 length)/2
```

Configurazione dei beacon composti

I beacon composti combinano stringhe letterali in chiaro e beacon standard per eseguire operazioni complesse sul database, ad esempio interrogare due diversi tipi di record da un singolo indice o eseguire una query su una combinazione di campi con una chiave di ordinamento. I beacon composti

possono essere costruiti da ENCRYPT_AND_SIGN e da campi. SIGN_ONLY È necessario creare un beacon standard per ogni campo crittografato incluso nel beacon composto.

Esempio di sintassi di configurazione

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

Per configurare un beacon composto, fornisci i seguenti valori.

Nome del faro

Il nome che usi per interrogare un campo crittografato.

Il nome di un beacon può avere lo stesso nome di un campo crittografato o virtuale, ma non può avere lo stesso nome di un campo non crittografato. Due beacon non possono avere lo stesso nome. Per aiutarti a determinare il nome del beacon migliore per la tua implementazione, vedi Scelta del nome del beacon.

Dividi carattere

Il personaggio usato per separare le parti che compongono il tuo faro composto.

Il carattere diviso non può apparire nei valori in chiaro di nessuno dei campi da cui è costruito il beacon composto.

Elenco delle parti crittografato

Identifica i ENCRYPT_AND_SIGN campi inclusi nel beacon composto.

Ogni parte deve includere un nome e un prefisso. Il nome della parte deve essere il nome del beacon standard costruito dal campo crittografato. Il prefisso può essere qualsiasi stringa, ma deve essere univoco. Una parte crittografata non può avere lo stesso prefisso di una parte firmata. Si consiglia di utilizzare un valore breve che distingua la parte dalle altre parti servite dal faro composto. Per semplificare le query sui beacon, ti consigliamo anche di identificare una parte

con lo stesso prefisso in ogni beacon in cui è inclusa ed evitare di utilizzare lo stesso prefisso per identificare parti diverse.

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

Elenco delle parti firmate (opzionale)

Identifica i `SIGN_ONLY` campi inclusi nel beacon composto.

Ogni parte deve includere un nome, un'origine e un prefisso. L'origine è il `SIGN_ONLY` campo identificato dalla parte. L'origine deve essere un nome di campo o un indice che si riferisce al valore di un campo annidato. Se il nome della parte identifica la fonte, puoi omettere la fonte e AWS Database Encryption SDK utilizzerà automaticamente il nome come origine. Si consiglia di specificare la fonte come nome della parte ogni volta che è possibile. Il prefisso può essere qualsiasi stringa, ma deve essere univoco. Una parte firmata non può avere lo stesso prefisso di una parte crittografata. Si consiglia di utilizzare un valore breve che distingua la parte dalle altre parti servite dal faro composto. Per semplificare le query sui beacon, ti consigliamo anche di identificare una parte con lo stesso prefisso in ogni beacon in cui è inclusa ed evitare di utilizzare lo stesso prefisso per identificare parti diverse.

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

Elenco costruttori (opzionale)

Identifica i costruttori che definiscono i diversi modi in cui le parti crittografate e firmate possono essere assemblate dal beacon composto.

Se non specificate un elenco di costruttori, AWS Database Encryption SDK assembla il beacon composto con il seguente costruttore predefinito.

- Tutte le parti firmate nell'ordine in cui sono state aggiunte all'elenco delle parti firmate

- Tutte le parti crittografate nell'ordine in cui sono state aggiunte all'elenco delle parti crittografate
- Tutte le parti sono obbligatorie

Costruttori

Ogni costruttore è un elenco ordinato di parti del costruttore che definisce un modo in cui il faro composto può essere assemblato. Le parti del costruttore vengono unite nell'ordine in cui vengono aggiunte all'elenco, con ciascuna parte separata dal carattere di divisione specificato.

Ogni parte del costruttore nomina una parte crittografata o una parte firmata e definisce se tale parte è obbligatoria o facoltativa all'interno del costruttore. Ad esempio, se desideri interrogare un beacon composto su`Field1`, e `Field1.Field2``Field1.Field2.Field3`, contrassegna e `Field3` come opzionale `Field2` e crea un costruttore.

Ogni costruttore deve avere almeno una parte richiesta. Ti consigliamo di creare la prima parte di ogni costruttore richiesto in modo da poter utilizzare l'`BEGINS_WITH`operatore nelle tue domande.

Un costruttore ha successo se nel record sono presenti tutte le parti necessarie. Quando si scrive un nuovo record, il beacon composto utilizza l'elenco dei costruttori per determinare se il beacon può essere assemblato dai valori forniti. Tenta di assemblare il beacon nell'ordine in cui i costruttori sono stati aggiunti all'elenco dei costruttori e utilizza il primo costruttore che riesce. Se nessun costruttore riesce, il beacon non viene scritto nel record.

Tutti i lettori e gli autori devono specificare lo stesso ordine di costruttori per garantire che i risultati delle loro query siano corretti.

Utilizzate le seguenti procedure per specificare il vostro elenco di costruttori.

1. Create una parte costruttore per ogni parte crittografata e parte firmata per definire se quella parte è necessaria o meno.

   Il nome della parte del costruttore deve essere il nome del faro standard o del campo firmato che rappresenta.

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

2. Crea un costruttore per ogni possibile modo in cui il faro composto può essere assemblato utilizzando le parti del costruttore che hai creato nel passaggio 1.

Ad esempio, se si desidera eseguire una query su `Field1.Field2.Field3` and`Field4.Field2.Field3`, è necessario creare due costruttori. `Field1`e `Field4` possono essere entrambi richiesti perché sono definiti in due costruttori separati.

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

3. Crea un elenco di costruttori che includa tutti i costruttori che hai creato nel passaggio 2.

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

4. Specifica `constructorList` quando crei il tuo beacon firmato.

Configurazioni di esempio

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Gli esempi seguenti mostrano come configurare beacon standard e composti. Le seguenti configurazioni non forniscono lunghezze dei beacon. Per informazioni su come determinare la

lunghezza del beacon appropriata per la configurazione, consulta Scegliere una lunghezza del beacon.

Per vedere esempi di codice completi che dimostrano come configurare e utilizzare i beacon, consulta la directory searchableencryption del repository -dynamodb-java su. aws-database-encryption-sdk GitHub

Argomenti

- Fari standard
- Fari composti

Fari standard

Se desideri interrogare il `inspector_id_last4` campo per verificare le corrispondenze esatte, crea un beacon standard utilizzando la seguente configurazione.

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

Fari composti

Se desideri interrogare il `UnitInspection` database su `inspector_id_last4` e`inspector_id_last4.unit`, crea un beacon composto con la seguente configurazione. Questo beacon composto richiede solo parti crittografate.

```
// 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();
```

Utilizzo dei beacon

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

I beacon consentono di cercare record crittografati senza decrittografare l'intero database oggetto della query. I beacon sono progettati per essere implementati in nuovi database non popolati. Qualsiasi beacon configurato in un database esistente mapperà solo i nuovi record scritti nel database. I beacon vengono calcolati in base al valore di testo in chiaro di un campo, una volta che il campo è crittografato, non è possibile per il beacon di mappare i dati esistenti. Dopo aver scritto

nuovi record con un beacon, non è possibile aggiornare la configurazione del beacon. Tuttavia, puoi aggiungere nuovi beacon per i nuovi campi che aggiungi al tuo record.

Dopo aver configurato i beacon, è necessario completare i passaggi seguenti prima di iniziare a popolare il database ed eseguire query sui beacon.

1. Crea un portachiavi AWS KMS gerarchico

   Per utilizzare la crittografia ricercabile, è necessario utilizzare il portachiavi AWS KMS gerarchico per generare, crittografare e decrittografare le chiavi dati utilizzate per proteggere i record.

   Dopo aver configurato i beacon, assembla i prerequisiti del portachiavi gerarchico e crea il tuo portachiavi gerarchico.

   Per ulteriori dettagli sul motivo per cui è necessario il portachiavi gerarchico, vedere Utilizzo del portachiavi gerarchico per la crittografia ricercabile.

2.

   Definire la versione del beacon

   Specifica il tuo`keyStore`,`keySource`, un elenco di tutti i beacon standard che hai configurato, un elenco di tutti i beacon composti che hai configurato e una versione del beacon. È necessario specificare 1 la versione del beacon. Per indicazioni su come definire il tuo`keySource`, vediDefinizione della fonte della chiave del beacon.

   Il seguente esempio di Java definisce la versione beacon per un database a tenant singolo. Per informazioni sulla definizione della versione beacon per un database multitenant, consulta Crittografia ricercabile per database multitenant.

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

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

3. Configurazione degli indici secondari

   Dopo aver configurato i beacon, è necessario configurare un indice secondario che rifletta ogni beacon prima di poter eseguire la ricerca nei campi crittografati. Per ulteriori informazioni, consulta Configurazione degli indici secondari con i beacon.

4. Definisci le tue azioni crittografiche

   Tutti i campi utilizzati per costruire un faro standard devono essere contrassegnati. `ENCRYPT_AND_SIGN` Tutti gli altri campi utilizzati per costruire i beacon devono essere contrassegnati. `SIGN_ONLY`

5. Configurazione di un AWS client SDK per la crittografia del database

   Per configurare un client AWS Database Encryption SDK che protegge gli elementi della tabella nella tabella DynamoDB, consulta la libreria di crittografia lato client Java per DynamoDB.

Interrogazione dei beacon

Il tipo di beacon configurato determina il tipo di query che è possibile eseguire. I beacon standard utilizzano espressioni di filtro per eseguire ricerche di uguaglianza. I beacon composti combinano stringhe letterali in chiaro e beacon standard per eseguire query complesse. Quando interroghi dati crittografati, esegui una ricerca in base al nome del beacon.

Non è possibile confrontare i valori di due beacon standard, anche se contengono lo stesso testo in chiaro sottostante. I due beacon standard produrranno due diversi tag HMAC per gli stessi valori di testo in chiaro. Di conseguenza, i beacon standard non possono eseguire le seguenti domande.

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

I beacon composti possono eseguire le seguenti domande.

- `BEGINS_WITH(a)`, dove *a* riflette l'intero valore del campo da cui inizia il beacon composto assemblato. Non è possibile utilizzare l'`BEGINS_WITH`operatore per identificare un valore che inizia con una particolare sottostringa. Tuttavia, è possibile utilizzare`BEGINS_WITH(S_)`, where *S_* riflette il prefisso di una parte con cui inizia il faro composto assemblato.
- `CONTAINS(a)`, dove *a* riflette l'intero valore di un campo contenuto nel beacon composto assemblato. Non è possibile utilizzare l'`CONTAINS`operatore per identificare un record che contiene una particolare sottostringa o un valore all'interno di un set.

  Ad esempio, non è possibile eseguire una query `CONTAINS(path, "a"` in cui *a* rifletta il valore di un set.
- È possibile confrontare parti firmate di beacon composti. Quando si confrontano parti firmate, è possibile aggiungere facoltativamente il prefisso di una parte crittografata a una o più parti firmate, ma non è possibile includere il valore di un campo crittografato in nessuna query.

  Ad esempio, puoi confrontare parti firmate ed eseguire una query su *signedField1* = *signedField2* o*value* `IN (`*signedField1*`, `*signedField2*`, ...)`.

  È inoltre possibile confrontare le parti firmate e il prefisso di una parte crittografata mediante interrogazione. *signedField1.A_* = *signedField2.B_*
- *field* `BETWEEN` *a* `AND` *b*, dove *a* e *b* sono parti firmate. Facoltativamente, è possibile aggiungere il prefisso di una parte crittografata a una o più parti firmate, ma non è possibile includere il valore di un campo crittografato in nessuna query.

È necessario includere il prefisso per ogni parte inclusa in una query su un beacon composto. Ad esempio, se hai creato un beacon composto,`compoundBeacon`, da due campi `encryptedField` e`signedField`, devi includere i prefissi configurati per quelle due parti quando interroghi il beacon.

```
compoundBeacon = E_encryptedFieldValue.S_signedFieldValue
```

Crittografia ricercabile per database multitenant

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce ancora informazioni sul client di crittografia DynamoDB.

Per implementare la crittografia ricercabile nel tuo database, devi utilizzare un portachiavi AWS KMSgerarchico. Il portachiavi AWS KMS gerarchico genera, crittografa e decrittografa le chiavi dati utilizzate per proteggere i tuoi record. Crea anche la chiave beacon utilizzata per generare i beacon. Quando si utilizza il portachiavi AWS KMS gerarchico con database multitenant, esistono una chiave di diramazione e una chiave beacon distinte per ogni tenant. Per interrogare dati crittografati in un database multitenant, è necessario identificare i materiali chiave del beacon utilizzati per generare il beacon che si sta interrogando.

Quando definisci la versione del beacon per un database multitenant, specifica un elenco di tutti i beacon standard che hai configurato, un elenco di tutti i beacon composti che hai configurato, una versione del beacon e un. `keySource` È necessario definire la fonte della chiave beacon come e includere un `MultiKeyStore` `keyFieldName` tempo di cache per la cache delle chiavi beacon locale e la dimensione massima della cache per la cache delle chiavi beacon locale.

Se hai configurato dei beacon firmati, devono essere inclusi nel tuo. `compoundBeaconList` I beacon firmati sono un tipo di beacon composto che indicizza ed esegue query complesse sui campi. `SIGN_ONLY`

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

keyFieldName

`keyFieldName`Definisce il nome del campo che memorizza la chiave `branch-key-id` associata alla chiave beacon utilizzata per generare i beacon per un determinato tenant.

Quando si scrivono nuovi record nel database, in `branch-key-id` questo campo viene memorizzata la chiave beacon utilizzata per generare eventuali beacon per quel record.

Per impostazione predefinita, `keyField` è un campo concettuale che non è memorizzato in modo esplicito nel database. Il AWS Database Encryption SDK identifica la chiave `branch-key-id` di dati crittografata nella descrizione del materiale e memorizza il valore concettuale `keyField` a cui fare riferimento nei beacon composti e nei beacon firmati. Poiché la descrizione del materiale è firmata, il concettuale `keyField` è considerato una parte firmata.

Puoi anche includere il `keyField` nelle tue azioni crittografiche come campo per archiviare esplicitamente il `SIGN_ONLY` campo nel tuo database. In tal caso, è necessario includere manualmente `keyField` ogni volta che si scrive un record nel database. `branch-key-id`

Interrogazione dei beacon in un database multitenant

Per interrogare un beacon, è necessario includere `keyField` nella richiesta i materiali chiave del beacon appropriati per ricalcolare il beacon. È necessario specificare la chiave `branch-key-id` associata alla chiave beacon utilizzata per generare i beacon per un record. Non è possibile specificare il nome identificativo di un inquilino `branch-key-id` nell'ID del fornitore della filiale. Puoi includerli `keyField` nelle tue domande nei seguenti modi.

Fari composti

Indipendentemente dal fatto che tu li memorizzi esplicitamente `keyField` nei tuoi registri o meno, puoi includerli `keyField` direttamente nei tuoi beacon composti come parte firmata. La parte `keyField` firmata deve essere obbligatoria.

Ad esempio, se si desidera costruire un faro composto, da due campi `encryptedField` e `compoundBeaconsignedField`, è necessario includere anche la parte `keyField` come parte firmata. Ciò consente di eseguire la seguente interrogazione su`compoundBeacon`.

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

Fari firmati

Il AWS Database Encryption SDK utilizza beacon standard e composti per fornire soluzioni di crittografia ricercabili. Questi beacon devono includere almeno un campo crittografato. Tuttavia, AWS Database Encryption SDK supporta anche beacon firmati che possono essere configurati interamente da campi di testo non crittografato. `SIGN_ONLY`

I beacon firmati possono essere costruiti a partire da una singola parte. Che tu lo memorizzi esplicitamente `keyField` nei tuoi record o meno, puoi costruire un beacon firmato da `keyField` e utilizzarlo per creare query composte che combinano una query sul beacon `keyField` firmato con una query su uno degli altri beacon. Ad esempio, è possibile eseguire la seguente query.

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

Per assistenza nella configurazione dei beacon firmati, consulta Creazione di beacon firmati

Interroga direttamente sul **`keyField`**

Se hai specificato il `keyField` nelle tue azioni crittografiche e memorizzi esplicitamente il campo nel tuo record, puoi creare una query composta che combini una query sul tuo beacon con una query sul. `keyField` Puoi scegliere di interrogare direttamente su `keyField` se desideri interrogare un beacon standard. Ad esempio, è possibile eseguire la seguente query.

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

AWSSDK di crittografia del database per DynamoDB

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Il AWS Database Encryption SDK per DynamoDB è una libreria software che consente di includere la crittografia lato client nella progettazione di Amazon DynamoDB. L'SDK per la crittografia del AWS database per DynamoDB fornisce la crittografia a livello di attributo e consente di specificare quali elementi crittografare e quali elementi includere nelle firme che garantiscono l'autenticità dei dati. La crittografia dei dati sensibili in transito e a riposo aiuta a garantire che i dati in testo non crittografato non siano disponibili a terzi, ad esempio. AWS

> **Note**
>
> I seguenti argomenti si concentrano sulla versione 3. x della libreria di crittografia lato client Java per DynamoDB.
> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK.
> Il AWS Database Encryption SDK continua a supportare le versioni precedenti di DynamoDB Encryption Client.

In DynamoDB, una tabella è una raccolta di elementi. E ogni item è una raccolta di attributi. Ogni attributo ha un nome e un valore. L'SDK di crittografia del AWS database per DynamoDB crittografa i valori degli attributi. Quindi calcola una firma sugli attributi. È necessario specificare quali valori di attributo crittografare e quali includere nella firma nelle azioni crittografiche.

Gli argomenti di questo capitolo forniscono una panoramica del AWS Database Encryption SDK per DynamoDB, inclusi i campi crittografati, indicazioni sull'installazione e la configurazione del client ed esempi Java per aiutarti a iniziare.

Argomenti

- Crittografia lato client e lato server
- Quali campi sono crittografati e firmati?
- Java

- Client di crittografia DynamoDB Legacy

Crittografia lato client e lato server

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

L'SDK per la crittografia del AWS database per DynamoDB supporta la crittografia lato client, in cui è possibile crittografare i dati della tabella prima di inviarli al database. Tuttavia, DynamoDB fornisce una funzionalità di crittografia a riposo lato server che crittografa in modo trasparente la tabella quando viene conservata su disco e la decrittografa quando si accede alla tabella.

La scelta degli strumenti dipende dal livello di riservatezza dei dati e dai requisiti di sicurezza dell'applicazione. Puoi utilizzare sia il AWS Database Encryption SDK per DynamoDB che la crittografia a riposo. Quando invii elementi crittografati e firmati a DynamoDB, DynamoDB non li riconosce come protetti. Rileva soltanto gli item tipici della tabella con valori di attributo binari.

Crittografia dei dati inattivi lato server

DynamoDB supporta la crittografia a riposo, una funzionalità di crittografia lato server in cui DynamoDB crittografa in modo trasparente le tabelle per te quando la tabella viene conservata su disco e le decrittografa quando accedi ai dati della tabella.

Quando utilizzi un AWS SDK per interagire con DynamoDB, per impostazione predefinita, i tuoi dati vengono crittografati in transito su una connessione HTTPS, decrittografati sull'endpoint DynamoDB e quindi ricritrittografati prima di essere archiviati in DynamoDB.

- Crittografia predefinita. DynamoDB crittografa e decrittografa in modo trasparente tutte le tabelle quando vengono scritte. Non c'è alcuna opzione per abilitare o disabilitare la crittografia dei dati inattivi.
- DynamoDB crea e gestisce le chiavi crittografiche.La chiave univoca per ogni tabella è protetta da un messaggio AWS KMS keyche non lascia mai AWS Key Management Service(AWS KMS) non crittografato. Per impostazione predefinita, DynamoDB utilizza un Chiave di proprietà di AWSaccount del servizio DynamoDB, ma puoi scegliere una Chiave gestita da AWSchiave gestita dal cliente nel tuo account per proteggere alcune o tutte le tabelle.

- Tutti i dati della tabella sono crittografati su disco.Quando una tabella crittografata viene salvata su disco, DynamoDB crittografa tutti i dati della tabella, inclusa la chiave primaria e gli indici secondari locali e globali. Se la tabella dispone di chiavi di ordinamento, alcune di queste che contrassegnano i limiti dell'intervallo sono archiviate come testo non crittografato nei metadati della tabella.
- Anche gli oggetti correlati alle tabelle sono crittografati. La crittografia a riposo protegge i flussi, le tabelle globali e i backup di DynamoDB ogni volta che vengono scritti su supporti durevoli.
- I tuoi articoli vengono decifrati quando li accedi.Quando accedi alla tabella, DynamoDB decripta la parte della tabella che include l'elemento di destinazione e ti restituisce l'elemento in testo normale.

AWSSDK di crittografia del database per DynamoDB

La crittografia lato client fornisce protezione completa dei dati, in transito e inattivi, dall'origine all'archiviazione in DynamoDB. I dati di testo non crittografato non vengono mai esposti a terze parti, compreso AWS. Puoi utilizzare il AWS Database Encryption SDK per DynamoDB con nuove tabelle DynamoDB oppure puoi migrare le tabelle Amazon DynamoDB esistenti alla versione 3. x della libreria di crittografia lato client Java per DynamoDB.

- I dati sono protetti sia quando sono in transito sia quando sono inattivi. Non è mai esposto a terzi, inclusiAWS.
- Puoi firmare gli item della tabella. Puoi indirizzare il AWS Database Encryption SDK per DynamoDB a calcolare una firma su tutto o parte di un elemento della tabella, inclusi gli attributi della chiave primaria. Tramite le firme, puoi rilevare modifiche non autorizzate all'item nel suo insieme, tra cui l'aggiunta o l'eliminazione di attributi o lo scambio dei valori di attributo.
- Puoi determinare come vengono protetti i tuoi dati selezionando un portachiavi. Il tuo portachiavi determina le chiavi di avvolgimento che proteggono le tue chiavi dati e, in ultima analisi, i tuoi dati. Usa le chiavi di imballaggio più sicure e pratiche per il tuo compito.
- L'SDK per la crittografia del AWS database per DynamoDB non crittografa l'intera tabella. Sei tu a scegliere quali attributi devono essere crittografati nei tuoi articoli. L'SDK per la crittografia del AWS database per DynamoDB non crittografa un intero elemento. Non crittografa i nomi di attributo o i nomi o i valori degli attributi della chiave primaria (chiave di partizione e chiave di ordinamento).

AWS Encryption SDK

Se stai crittografando i dati archiviati in DynamoDB, ti consigliamo il AWS Database Encryption SDK per DynamoDB.

AWS Encryption SDK è una libreria di crittografia lato client che ti consente di crittografare e decrittografare i dati generici. Anche se è in grado di proteggere qualsiasi tipo di dati, non è stato progettato per funzionare con i dati strutturati, come i record di database. A differenza del AWS Database Encryption SDK per DynamoDB, AWS Encryption SDK non è in grado di fornire un controllo dell'integrità a livello di elemento e non ha alcuna logica per riconoscere gli attributi o impedire la crittografia delle chiavi primarie.

Se usi AWS Encryption SDK per crittografare qualsiasi elemento della tua tabella, ricorda che non è compatibile con il AWS Database Encryption SDK per DynamoDB. Non puoi utilizzare due librerie diverse per la crittografia e la decrittografia.

Quali campi sono crittografati e firmati?

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Il AWS Database Encryption SDK per DynamoDB è una libreria di crittografia lato client progettata appositamente per le applicazioni Amazon DynamoDB. Amazon DynamoDB archivia i dati in tabelle, che sono una raccolta di elementi. E ogni item è una raccolta di attributi. Ogni attributo ha un nome e un valore. Il AWS Database Encryption SDK per DynamoDB crittografa i valori degli attributi. Quindi calcola una firma sugli attributi. Puoi specificare quali valori degli attributi crittografare e quali includere nelle firma.

La crittografia protegge la riservatezza del valore degli attributi. La firma assicura l'integrità di tutti gli attributi firmati e la loro relazione reciproca e fornisce l'autenticazione. Consente di rilevare modifiche non autorizzate all'item nel suo insieme, come l'aggiunta o l'eliminazione di attributi o la sostituzione di un valore crittografato con un altro.

In un elemento crittografato, alcuni dati rimangono in testo normale, tra cui il nome della tabella, tutti i nomi degli attributi, i valori degli attributi che non vengono crittografati, i nomi e i valori degli attributi della chiave primaria (chiave di partizione e chiave di ordinamento) e i tipi di attributo. Non archiviare dati sensibili in questi campi.

Per ulteriori informazioni su come funziona il AWS Database Encryption SDK per DynamoDB, consulta. Come funziona il AWS Database Encryption SDK

> **Note**
>
> Tutti i riferimenti alle azioni degli attributi negli argomenti AWS Database Encryption SDK per DynamoDB si riferiscono alle azioni crittografiche.

Argomenti

Crittografia dei valori degli attributi

L'SDK per la crittografia del AWS database per DynamoDB crittografa i valori (ma non il nome o il tipo di attributo) degli attributi specificati. Per determinare quali valori attributo vengono crittografati, utilizza le operazioni di attributo.

Ad esempio, questo item include gli attributi `example` e `test`.

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

Se effettui la crittografia dell'attributo `example`, ma non dell'attributo `test`, i risultati saranno simili a quelli riportati di seguito. Il valore dell'attributo `example` sono dati binari e non una stringa.

```
'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'
...
```

Gli attributi chiave primari (chiave di partizione e chiave di ordinamento) di ogni elemento devono rimanere in testo normale perché DynamoDB li utilizza per trovare l'elemento nella tabella. Devono essere firmati, ma non crittografati.

Il AWS Database Encryption SDK per DynamoDB identifica gli attributi della chiave primaria per te e garantisce che i relativi valori siano firmati, ma non crittografati. Se individui la tua chiave primaria e tenti di crittografarla, il client genera un'eccezione.

Il client memorizza la descrizione del materiale in un nuovo attributo (`aws_dbe_head`) che aggiunge all'articolo. La descrizione del materiale descrive come l'articolo è stato crittografato e firmato. Il client utilizza l'informazione per verificare e decrittografare l'item. Il campo in cui è memorizzata la descrizione del materiale non è crittografato.

Firma dell'item

Dopo aver crittografato i valori degli attributi specificati, il AWS Database Encryption SDK per DynamoDB calcola i codici di autenticazione dei messaggi basati su hash (HMAC) e una firma digitale sulla base della canonicalizzazione della descrizione del materiale, del contesto di crittografia e di ogni campo contrassegnato o nelle azioni degli attributi. `ENCRYPT_AND_SIGNSIGN_ONLY`
Le firme ECDSA sono abilitate per impostazione predefinita, ma non sono obbligatorie. Il client memorizza gli HMAC e le firme in un nuovo attributo (`aws_dbe_foot`) che aggiunge all'elemento.

Java

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Questo argomento spiega come installare e utilizzare la versione 3. x della libreria di crittografia lato client Java per DynamoDB. Per dettagli sulla programmazione con AWS Database Encryption SDK per DynamoDB, consulta la directory degli esempi del repository aws-database-encryption-sdk - dynamodb-java su. GitHub

> **Note**
>
> I seguenti argomenti si concentrano sulla versione 3. x della libreria di crittografia lato client Java per DynamoDB.
> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Il AWS Database Encryption SDK continua a supportare le versioni precedenti di DynamoDB Encryption Client.

Argomenti

- Prerequisiti

- Installazione
- Utilizzo della libreria di crittografia lato client Java per DynamoDB
- Esempi di Java
- Aggiornamento del modello di dati
- Configurare una tabella DynamoDB esistente per utilizzare il AWS Database Encryption SDK per DynamoDB
- Esegui la migrazione alla versione 3.x della libreria di crittografia lato client Java per DynamoDB

Prerequisiti

Prima di installare la versione 3. x della libreria di crittografia lato client Java per DynamoDB, assicurati di avere i seguenti prerequisiti.

Un ambiente di sviluppo Java

È necessario Java 8 o versioni successive. Nel sito Web di Oracle, accedi alla pagina Java SE Download, quindi scarica e installa Java SE Development Kit (JDK).

Se utilizzi Oracle JDK, devi scaricare e installare anche Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files.

AWS SDK for Java 2.x

L'SDK di crittografia del AWS database per DynamoDB richiede il modulo DynamoDB Enhanced Client di. AWS SDK for Java 2.x Puoi installare l'intero SDK o solo questo modulo.

Per informazioni sull'aggiornamento della versione in uso diAWS SDK for Java, vedere Migrazione dalla versione 1.x alla 2.x di. AWS SDK for Java

AWS SDK for JavaÈ disponibile tramite Apache Maven. Puoi dichiarare una dipendenza per l'intero AWS SDK for Java modulo o solo per il `dynamodb-enhanced` modulo.

Installa AWS SDK for Java utilizzando Apache Maven

- Per importare l'intero AWS SDK for Java come dipendenza, dichiaralo nel file `pom.xml`.
- Per creare una dipendenza solo per il modulo Amazon DynamoDB nelAWS SDK for Java, segui le istruzioni per specificare moduli particolari. Imposta il `groupId` to `software.amazon.awssdk` e il `artifactID` to`dynamodb-enhanced`.

> **Note**
>
> Se si utilizza il AWS KMS portachiavi o il portachiavi AWS KMS gerarchico, è necessario creare anche una dipendenza per il modulo. AWS KMS Imposta il `groupId` to `software.amazon.awssdk` e il `artifactID` to`kms`.

Installazione

Puoi installare la versione 3. x della libreria di crittografia lato client Java per DynamoDB nei seguenti modi.

Utilizzo di Apache Maven

Il client di crittografia Amazon DynamoDB per Java è disponibile tramite Apache Maven con la seguente definizione di dipendenza.

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

Usare Gradle Kotlin

Puoi usare Gradle per dichiarare una dipendenza da The Amazon DynamoDB Encryption Client for Java aggiungendo quanto segue alla sezione delle dipendenze del tuo progetto Gradle.

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

Manualmente

Per installare la libreria di crittografia lato client Java per DynamoDB, clona o scarica il repository - dynamodb-java. aws-database-encryption-sdk GitHub

Dopo aver installato l'SDK, inizia guardando il codice di esempio in questa guida e la directory degli esempi del repository aws-database-encryption-sdk -dynamodb-java su. GitHub

Utilizzo della libreria di crittografia lato client Java per DynamoDB

La nostra libreria di crittografia lato client è stata rinominata Database Encryption SDK. AWS Questa guida per sviluppatori fornisce ancora informazioni sul DynamoDB Encryption Client.

Questo argomento spiega alcune delle funzioni e delle classi di supporto della versione 3. x della libreria di crittografia lato client Java per DynamoDB.

Per informazioni dettagliate sulla programmazione con la libreria di crittografia lato client Java per DynamoDB, consulta gli esempi Java, la directory examples del repository -dynamodb-java on. aws-database-encryption-sdk GitHub

Argomenti

Componenti di crittografia dell'item

Fondamentalmente, il AWS Database Encryption SDK per DynamoDB è un cifratore di elementi. È possibile utilizzare la versione 3. x della libreria di crittografia lato client Java per DynamoDB per crittografare, firmare, verificare e decrittografare gli elementi della tabella DynamoDB nei seguenti modi.

Il client avanzato per DynamoDB

È possibile configurare il DynamoDB Enhanced Client per crittografare e firmare automaticamente `DynamoDbEncryptionInterceptor` gli elementi lato client con le richieste DynamoDB. `PutItem` Con DynamoDB Enhanced Client, puoi definire le azioni degli attributi utilizzando una classe di dati annotata. Consigliamo di utilizzare il DynamoDB Enhanced Client ogni volta che è possibile.

> **Note**
>
> Il AWS Database Encryption SDK non supporta le annotazioni sugli attributi annidati.

L'API DynamoDB di basso livello

Puoi configurare l'API DynamoDB di basso livello per crittografare e firmare automaticamente gli elementi lato client con `DynamoDbEncryptionInterceptor` le tue richieste DynamoDB. `PutItem`

Il livello inferiore **`DynamoDbItemEncryptor`**

Il livello inferiore crittografa e firma o decrittografa e verifica `DynamoDbItemEncryptor` direttamente gli elementi della tabella senza chiamare DynamoDB. Non crea DynamoDB o `PutItem` richieste`GetItem`. Ad esempio, puoi utilizzare il livello inferiore per `DynamoDbItemEncryptor` decrittografare e verificare direttamente un elemento DynamoDB che hai già recuperato.

Il livello inferiore non supporta la crittografia ricercabile. `DynamoDbItemEncryptor`

Azioni relative agli attributi nel AWS Database Encryption SDK per DynamoDB

Le operazioni di attributo determinano quali valori attributo sono crittografati e firmati, quali solo firmati e quali ignorati.

Se utilizzi l'API DynamoDB di basso livello o quella di `DynamoDbItemEncryptor` livello inferiore, devi definire manualmente le azioni degli attributi. Se si utilizza il DynamoDB Enhanced Client, è possibile definire manualmente le azioni relative agli attributi oppure utilizzare una classe di dati annotata per generare un. `TableSchema` Per semplificare il processo di configurazione, consigliamo di utilizzare una classe di dati annotata. Quando utilizzate una classe di dati annotata, dovete modellare l'oggetto una sola volta.

> **Note**
>
> Dopo aver definito le azioni relative agli attributi, è necessario definire quali attributi sono esclusi dalle firme. Per semplificare l'aggiunta di nuovi attributi non firmati in futuro, consigliamo di scegliere un prefisso distinto (ad esempio `":"`) per identificare gli attributi non firmati. Includi questo prefisso nel nome dell'attributo per tutti gli attributi contrassegnati durante `DO_NOTHING` la definizione dello schema DynamoDB e delle azioni degli attributi.

Utilizza una classe di dati annotata

Utilizza una classe di dati annotata per specificare le azioni degli attributi con DynamoDB Enhanced Client e. `DynamoDbEncryptionInterceptor` Il AWS Database Encryption SDK per DynamoDB utilizza le annotazioni standard degli attributi DynamoDB che definiscono il tipo di attributo per determinare come proteggere un attributo. Per impostazione predefinita, tutti gli attributi sono crittografati e firmati, tranne le chiavi primarie, che sono firmate ma non crittografate.

SimpleClassConsulta.java nel repository aws-database-encryption-sdk -dynamodb-java su per GitHub ulteriori indicazioni sulle annotazioni di DynamoDB Enhanced Client.

Per impostazione predefinita, gli attributi della chiave primaria sono firmati ma non crittografati (`SIGN_ONLY`) e tutti gli altri attributi sono crittografati e firmati (). `ENCRYPT_AND_SIGN` Per specificare le eccezioni, utilizzate le annotazioni di crittografia definite nella libreria di crittografia lato client Java per DynamoDB. Ad esempio, se desideri che un particolare attributo venga firmato, utilizza l'annotazione. `@DynamoDbEncryptionSignOnly` Se vuoi che un particolare attributo non sia né firmato né crittografato (`DO_NOTHING`), usa l'`@DynamoDbEncryptionDoNothing`annotazione.

> **Note**
>
> Il AWS Database Encryption SDK non supporta le annotazioni sugli attributi annidati.

L'esempio seguente mostra le annotazioni utilizzate per definire le azioni degli attributi.

```
@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;
    }
}
```

Usa la tua classe di dati annotata per creare il file `TableSchema` come mostrato nel seguente frammento.

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

Definisci manualmente le azioni degli attributi

Per specificare manualmente le azioni degli attributi, create un `Map` oggetto in cui le coppie nome-valore rappresentino i nomi degli attributi e le azioni specificate.

Specificate `ENCRYPT_AND_SIGN` di crittografare e firmare un attributo. `SIGN_ONLY`Specificare di firmare, ma non crittografare, un attributo. Non è possibile crittografare un attributo senza firmarlo. `DO_NOTHING`Specificare di ignorare un attributo.

```
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);
```

Configurazione della crittografia nel AWS Database Encryption SDK per DynamoDB

Quando si utilizza AWS Database Encryption SDK, è necessario definire in modo esplicito una configurazione di crittografia per la tabella DynamoDB. I valori richiesti nella configurazione di crittografia dipendono dal fatto che le azioni degli attributi siano state definite manualmente o con una classe di dati annotata.

Il seguente frammento definisce una configurazione di crittografia delle tabelle DynamoDB utilizzando il DynamoDB Enhanced Client e gli attributi non firmati consentiti definiti da un prefisso `TableSchema`distinto.

```
final Map<String, DynamoDbEnhancedTableEncryptionConfig> tableConfigs = new
 HashMap<>();
tableConfigs.put(ddbTableName,
        DynamoDbEnhancedTableEncryptionConfig.builder()
            .logicalTableName(ddbTableName)
            .keyring(kmsKeyring)
            .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
            .schemaOnEncrypt(tableSchema)
            // Optional: only required if you use beacons
```

```
            .search(SearchConfig.builder()
                    .writeVersion(1) // MUST be 1
                    .versions(beaconVersions)
                    .build())
            .build());
```

Nome della tabella logica

Un nome di tabella logica per la tabella DynamoDB.

Il nome della tabella logica è associato crittograficamente a tutti i dati memorizzati nella tabella per semplificare le operazioni di ripristino di DynamoDB. Consigliamo vivamente di specificare il nome della tabella DynamoDB come nome della tabella logica quando si definisce per la prima volta la configurazione di crittografia. È necessario specificare sempre lo stesso nome di tabella logica. Affinché la decrittografia abbia esito positivo, il nome della tabella logica deve corrispondere al nome specificato nella crittografia. Nel caso in cui il nome della tabella DynamoDB cambi dopo il ripristino della tabella DynamoDB da un backup, il nome della tabella logica assicura che l'operazione di decrittografia riconosca ancora la tabella.

Attributi non firmati consentiti

Gli attributi contrassegnati `DO_NOTHING` nelle azioni relative agli attributi.

Gli attributi non firmati consentiti indicano al client quali attributi sono esclusi dalle firme. Il client presume che tutti gli altri attributi siano inclusi nella firma. Quindi, durante la decrittografia di un record, il client determina quali attributi deve verificare e quali ignorare tra gli attributi non firmati consentiti specificati. Non è possibile rimuovere un attributo dagli attributi non firmati consentiti.

È possibile definire gli attributi non firmati consentiti in modo esplicito creando un array che elenca tutti gli attributi. `DO_NOTHING` È inoltre possibile specificare un prefisso distinto quando si assegnano nomi `DO_NOTHING` agli attributi e utilizzare il prefisso per indicare al client quali attributi non sono firmati. Consigliamo vivamente di specificare un prefisso distinto perché semplifica il processo di aggiunta di un nuovo `DO_NOTHING` attributo in futuro. Per ulteriori informazioni, consulta Aggiornamento del modello di dati.

Se non si specifica un prefisso per tutti gli `DO_NOTHING` attributi, è possibile configurare un `allowedUnsignedAttributes` array che elenchi in modo esplicito tutti gli attributi che il client dovrebbe aspettarsi che non siano firmati quando li incontra durante la decrittografia. È necessario definire in modo esplicito gli attributi non firmati consentiti solo se assolutamente necessario.

Configurazione della ricerca (opzionale)

`SearchConfig`definisce la versione del beacon.

È `SearchConfig` necessario specificare il per utilizzare la crittografia ricercabile o i beacon firmati.

Algorithm Suite (opzionale)

`algorithmSuiteId`Definisce la suite di algoritmi utilizzata da AWS Database Encryption SDK.

A meno che non si specifichi esplicitamente una suite di algoritmi alternativa, AWS Database Encryption SDK utilizza la suite di algoritmi predefinita. La suite di algoritmi predefinita utilizza l'algoritmo AES-GCM con derivazione delle chiavi, firme digitali e impegno delle chiavi. Sebbene la suite di algoritmi predefinita sia probabilmente adatta alla maggior parte delle applicazioni, è possibile scegliere una suite di algoritmi alternativa. Ad esempio, alcuni modelli di fiducia sarebbero soddisfatti da una suite di algoritmi senza firme digitali. Per informazioni sulle suite di algoritmi supportate da AWS Database Encryption SDK, consulta. Suite di algoritmi supportate nel AWS Database Encryption SDK

Per selezionare la suite di algoritmi AES-GCM senza firme digitali, includi il seguente frammento nella configurazione di crittografia delle tabelle.

```
.algorithmSuiteId(

 DBEAlgorithmSuiteId.ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY_ECDSA_P384_SYMSIG_HMAC_SHA384)
```

Crittografia ricercabile in DynamoDB

Per configurare le tabelle Amazon DynamoDB per una crittografia ricercabile, devi utilizzare AWS KMS il portachiavi gerarchico per generare, crittografare e decrittografare le chiavi dati utilizzate per proteggere i tuoi articoli. È necessario utilizzare il DynamoDB Enhanced Client o l'API DynamoDB di basso livello per crittografare, firmare, verificare e decrittografare gli elementi della tabella. Il livello inferiore non supporta la crittografia ricercabile. `DynamoDBItemEncryptor` È inoltre necessario includere la configurazione di `SearchConfig`crittografia nella tabella.

Dopo aver configurato i beacon, è necessario configurare un indice secondario che rifletta ogni beacon prima di poter eseguire la ricerca negli attributi crittografati.

Configurazione degli indici secondari con i beacon

Quando si configura un beacon standard o composto, AWS Database Encryption SDK aggiunge il `aws_dbe_b_` prefisso al nome del beacon in modo che il server possa identificare facilmente i beacon. Ad esempio, se si assegna un nome a un beacon composto, in realtà il nome completo del beacon è. `compoundBeacon` `aws_dbe_b_compoundBeacon` Se si desidera configurare indici secondari che includano un beacon standard o composto, è necessario includere il prefisso quando si identifica il nome del `aws_dbe_b_` beacon.

Chiavi di partizione e ordinamento

Non è possibile crittografare i valori della chiave primaria. Le chiavi di partizione e di ordinamento devono essere. `SIGN_ONLY` I valori della chiave primaria non possono essere un beacon standard o composto.

I valori della chiave primaria possono essere i beacon firmati. Se hai configurato beacon firmati distinti per ciascuno dei valori della chiave primaria, devi specificare il nome dell'attributo che identifica il valore della chiave primaria come nome del beacon firmato. Tuttavia, AWS Database Encryption SDK non aggiunge il prefisso ai beacon firmati. `aws_dbe_b_` Anche se hai configurato beacon firmati distinti per i valori della chiave primaria, devi specificare i nomi degli attributi per i valori della chiave primaria solo quando configuri un indice secondario.

Indici secondari locali

La chiave di ordinamento per un indice secondario locale può essere un beacon.

Se si specifica un beacon per la chiave di ordinamento, il tipo deve essere String. Se specificate un beacon standard o composto per la chiave di ordinamento, dovete includere il `aws_dbe_b_` prefisso quando specificate il nome del beacon. Se specificate un beacon firmato, specificate il nome del beacon senza alcun prefisso.

Indici secondari globali

Le chiavi di partizione e di ordinamento per un indice secondario globale possono essere entrambe beacon.

Se si specifica un beacon per la chiave di partizione o di ordinamento, il tipo deve essere String. Se specificate un beacon standard o composto per la chiave di ordinamento, dovete includere il `aws_dbe_b_` prefisso quando specificate il nome del beacon. Se specificate un beacon firmato, specificate il nome del beacon senza alcun prefisso.

Proiezioni di attributi

Una proiezione è l'insieme di attributi copiato da una tabella in un indice secondario. La chiave di partizione e la chiave di ordinamento della tabella vengono sempre proiettati nell'indice; è possibile proiettare altri attributi per supportare i requisiti di query dell'applicazione. DynamoDB offre tre diverse opzioni per le proiezioni `KEYS_ONLY` degli attributi:, e. `INCLUDE` `ALL`

Se si utilizza la proiezione dell'attributo INCLUDE per cercare su un beacon, è necessario specificare i nomi di tutti gli attributi da cui è costruito il beacon e il nome del beacon con il prefisso. `aws_dbe_b_` Ad esempio, se avete configurato un faro composto, from, and`compoundBeacon`, dovete specificare`field1`, `field2``field3`, e nella proiezione. `aws_dbe_b_compoundBeacon` `field1` `field2` `field3`

Un indice secondario globale può utilizzare solo gli attributi specificati esplicitamente nella proiezione, ma un indice secondario locale può utilizzare qualsiasi attributo.

Esempi di Java

La nostra libreria di crittografia lato client è stata rinominata Database Encryption SDK. AWS Questa guida per sviluppatori fornisce ancora informazioni sul DynamoDB Encryption Client.

Gli esempi seguenti mostrano come utilizzare la libreria di crittografia lato client Java per DynamoDB per proteggere gli elementi della tabella nell'applicazione. Puoi trovare altri esempi (e contribuire con i tuoi) nella directory examples del repository -dynamodb-java su. aws-database-encryption-sdk GitHub

Gli esempi seguenti mostrano come configurare la libreria di crittografia lato client Java per DynamoDB in una nuova tabella Amazon DynamoDB non popolata. Se desideri configurare le tabelle Amazon DynamoDB esistenti per la crittografia lato client, consulta. Aggiungi la versione 3.x a una tabella esistente

Argomenti

- Utilizzo del client avanzato DynamoDB
- Utilizzo dell'API DynamoDB di basso livello
- Utilizzo del livello inferiore DynamoDbItemEncryptor

Utilizzo del client avanzato DynamoDB

L'esempio seguente mostra come utilizzare il DynamoDB Enhanced Client `DynamoDbEncryptionInterceptor` e AWS KMS un portachiavi per crittografare gli elementi della tabella DynamoDB come parte delle chiamate API DynamoDB.

Puoi utilizzare qualsiasi portachiavi supportato con DynamoDB Enhanced Client, ma consigliamo di utilizzare uno dei AWS KMS portachiavi quando possibile.

Guarda l'esempio di codice completo: .java EnhancedPutGetExample

Fase 1: Creare il portachiavi AWS KMS

L'esempio seguente utilizza la creazione `CreateAwsKmsMrkMultiKeyring` di un AWS KMS portachiavi con una chiave KMS di crittografia simmetrica. Il `CreateAwsKmsMrkMultiKeyring` metodo garantisce che il portachiavi gestisca correttamente sia le chiavi a regione singola che quelle a più regioni.

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

Passaggio 2: creare uno schema tabellare dalla classe di dati annotata

L'esempio seguente utilizza la classe di dati annotati per creare il. `TableSchema`

Questo esempio presuppone che la classe di dati annotata e le azioni degli attributi siano state definite utilizzando .java. SimpleClass Per ulteriori informazioni sull'annotazione delle azioni relative agli attributi, consulta. Utilizza una classe di dati annotata

> **Note**
>
> Il AWS Database Encryption SDK non supporta le annotazioni sugli attributi annidati.

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

Fase 3: Definire quali attributi sono esclusi dalle firme

L'esempio seguente presuppone che tutti `DO_NOTHING` gli attributi condividano il prefisso distinto `":"` e utilizza il prefisso per definire gli attributi non firmati consentiti. Il client presuppone che qualsiasi nome di attributo con il prefisso `":"` sia escluso dalle firme. Per ulteriori informazioni, consulta Attributi non firmati consentiti.

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

Fase 4: Creare la configurazione di crittografia

L'esempio seguente definisce una `tableConfigs` mappa che rappresenta la configurazione di crittografia per la tabella DynamoDB.

Questo esempio specifica il nome della tabella DynamoDB come nome della tabella logica. Consigliamo vivamente di specificare il nome della tabella DynamoDB come nome della tabella logica quando si definisce per la prima volta la configurazione di crittografia. Per ulteriori informazioni, consulta Configurazione della crittografia nel AWS Database Encryption SDK per DynamoDB.

> **Note**
>
> Per utilizzare la crittografia ricercabile o i beacon firmati, è necessario includerli anche nella configurazione di crittografia. `SearchConfig`

```
final Map<String, DynamoDbEnhancedTableEncryptionConfig> tableConfigs = new
 HashMap<>();
tableConfigs.put(ddbTableName,
    DynamoDbEnhancedTableEncryptionConfig.builder()
        .logicalTableName(ddbTableName)
        .keyring(kmsKeyring)
        .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
        .schemaOnEncrypt(tableSchema)
        .build());
```

Fase 5: Crea il **`DynamoDbEncryptionInterceptor`**

L'esempio seguente ne crea uno nuovo `DynamoDbEncryptionInterceptor` con il `tableConfigs` passo 4.

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

Fase 6: Creare un nuovo client AWS SDK DynamoDB

L'esempio seguente crea un nuovo client AWS SDK DynamoDB utilizzando **`interceptor`** lo Step 5.

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

Fase 7: Creare il DynamoDB Enhanced Client e creare una tabella

L'esempio seguente crea il DynamoDB Enhanced Client utilizzando il client DynamoDB AWS SDK creato nel passaggio 6 e crea una tabella utilizzando la classe di dati annotati.

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

Fase 8: Crittografare e firmare un elemento della tabella

L'esempio seguente inserisce un elemento nella tabella DynamoDB utilizzando il DynamoDB Enhanced Client. L'elemento viene crittografato e firmato lato client prima di essere inviato a DynamoDB.

```
final SimpleClass item = new SimpleClass();
```

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

table.putItem(item);
```

Utilizzo dell'API DynamoDB di basso livello

L'esempio seguente mostra come utilizzare l'API DynamoDB di basso livello con AWS KMS un portachiavi per crittografare e firmare automaticamente gli elementi lato client con le richieste DynamoDB. `PutItem`

Puoi utilizzare qualsiasi portachiavi supportato, ma ti consigliamo di utilizzare uno dei portachiavi quando possibile. AWS KMS

Guarda l'esempio di codice completo: .java BasicPutGetExample

Fase 1: Creare il portachiavi AWS KMS

L'esempio seguente utilizza la creazione `CreateAwsKmsMrkMultiKeyring` di un AWS KMS portachiavi con una chiave KMS di crittografia simmetrica. Il `CreateAwsKmsMrkMultiKeyring` metodo garantisce che il portachiavi gestisca correttamente sia le chiavi a regione singola che quelle a più regioni.

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

Passaggio 2: configura le azioni relative agli attributi

L'esempio seguente definisce una `attributeActionsOnEncrypt` mappa che rappresenta azioni di esempio relative agli attributi per un elemento della tabella.

```
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);
```

Fase 3: Definire quali attributi sono esclusi dalle firme

L'esempio seguente presuppone che tutti `DO_NOTHING` gli attributi condividano il prefisso distinto "`:`«e utilizza il prefisso per definire gli attributi non firmati consentiti. Il client presuppone che qualsiasi nome di attributo con il prefisso "`:`" sia escluso dalle firme. Per ulteriori informazioni, consulta Attributi non firmati consentiti.

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

Fase 4: Definire la configurazione di crittografia delle tabelle DynamoDB

L'esempio seguente definisce una `tableConfigs` mappa che rappresenta la configurazione di crittografia per questa tabella DynamoDB.

Questo esempio specifica il nome della tabella DynamoDB come nome della tabella logica. Consigliamo vivamente di specificare il nome della tabella DynamoDB come nome della tabella logica quando si definisce per la prima volta la configurazione di crittografia. Per ulteriori informazioni, consulta Configurazione della crittografia nel AWS Database Encryption SDK per DynamoDB.

> **Note**
>
> Per utilizzare la crittografia ricercabile o i beacon firmati, è necessario includerli anche nella configurazione di crittografia. `SearchConfig`

```
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);
```

Fase 5: Creare il **`DynamoDbEncryptionInterceptor`**

L'esempio seguente crea l'`DynamoDbEncryptionInterceptor`utilizzo del `tableConfigs` dal passaggio 4.

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

Fase 6: Creare un nuovo client AWS SDK DynamoDB

L'esempio seguente crea un nuovo client AWS SDK DynamoDB utilizzando **`interceptor`** lo Step 5.

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

Fase 7: Crittografare e firmare un elemento della tabella DynamoDB

L'esempio seguente definisce una `item` mappa che rappresenta un elemento della tabella di esempio e inserisce l'elemento nella tabella DynamoDB. L'elemento viene crittografato e firmato lato client prima di essere inviato a DynamoDB.

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

final PutItemRequest putRequest = PutItemRequest.builder()
```

```
        .tableName(ddbTableName)
        .item(item)
        .build();

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

Utilizzo del livello inferiore DynamoDbItemEncryptor

L'esempio seguente mostra come utilizzare il livello inferiore `DynamoDbItemEncryptor` con un AWS KMS portachiavi per crittografare e firmare direttamente gli elementi della tabella. Non `DynamoDbItemEncryptor` inserisce l'elemento nella tabella DynamoDB.

Puoi utilizzare qualsiasi portachiavi supportato con DynamoDB Enhanced Client, ma consigliamo di utilizzare uno dei AWS KMS portachiavi quando possibile.

> **Note**
>
> Il livello inferiore `DynamoDbItemEncryptor` non supporta la crittografia ricercabile. Utilizzalo `DynamoDbEncryptionInterceptor` con l'API DynamoDB di basso livello o con il DynamoDB Enhanced Client per utilizzare la crittografia ricercabile.

Guarda l'esempio di codice completo: .java ItemEncryptDecryptExample

Fase 1: Creare il portachiavi AWS KMS

L'esempio seguente utilizza la creazione `CreateAwsKmsMrkMultiKeyring` di un AWS KMS portachiavi con una chiave KMS di crittografia simmetrica. Il `CreateAwsKmsMrkMultiKeyring` metodo garantisce che il portachiavi gestisca correttamente sia le chiavi a regione singola che quelle a più regioni.

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

Passaggio 2: configura le azioni relative agli attributi

L'esempio seguente definisce una `attributeActionsOnEncrypt` mappa che rappresenta azioni di esempio relative agli attributi per un elemento della tabella.

```
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);
```

Fase 3: Definire quali attributi sono esclusi dalle firme

L'esempio seguente presuppone che tutti `DO_NOTHING` gli attributi condividano il prefisso distinto "**:**«e utilizza il prefisso per definire gli attributi non firmati consentiti. Il client presuppone che qualsiasi nome di attributo con il prefisso "**:**" sia escluso dalle firme. Per ulteriori informazioni, consulta Attributi non firmati consentiti.

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

Fase 4: Definire la configurazione **`DynamoDbItemEncryptor`**

L'esempio seguente definisce la configurazione per`DynamoDbItemEncryptor`.

Questo esempio specifica il nome della tabella DynamoDB come nome della tabella logica. Consigliamo vivamente di specificare il nome della tabella DynamoDB come nome della tabella logica quando si definisce per la prima volta la configurazione di crittografia. Per ulteriori informazioni, consulta Configurazione della crittografia nel AWS Database Encryption SDK per DynamoDB.

```
final DynamoDbItemEncryptorConfig config = DynamoDbItemEncryptorConfig.builder()
        .logicalTableName(ddbTableName)
        .partitionKeyName("partition_key")
        .sortKeyName("sort_key")
        .attributeActionsOnEncrypt(attributeActionsOnEncrypt)
        .keyring(kmsKeyring)
        .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
        .build();
```

Fase 5: Creare il **`DynamoDbItemEncryptor`**

L'esempio seguente ne crea uno nuovo `DynamoDbItemEncryptor` utilizzando il `config` tratto dal passaggio 4.

```
final DynamoDbItemEncryptor itemEncryptor = DynamoDbItemEncryptor.builder()
        .DynamoDbItemEncryptorConfig(config)
        .build();
```

Fase 6: Crittografare e firmare direttamente un elemento della tabella

L'esempio seguente crittografa e firma direttamente un elemento utilizzando il. `DynamoDbItemEncryptor` Non `DynamoDbItemEncryptor` inserisce l'elemento nella tabella DynamoDB.

```
final Map<String, AttributeValue> originalItem = new HashMap<>();
originalItem.put("partition_key",
 AttributeValue.builder().s("ItemEncryptDecryptExample").build());
originalItem.put("sort_key", AttributeValue.builder().n("0").build());
originalItem.put("attribute1", AttributeValue.builder().s("encrypt and sign
 me!").build());
originalItem.put("attribute2", AttributeValue.builder().s("sign me!").build());
originalItem.put(":attribute3", AttributeValue.builder().s("ignore me!").build());

final Map<String, AttributeValue> encryptedItem = itemEncryptor.EncryptItem(
        EncryptItemInput.builder()
                .plaintextItem(originalItem)
                .build()
).encryptedItem();
```

Aggiornamento del modello di dati

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Quando configuri la libreria di crittografia lato client Java per DynamoDB, fornisci azioni relative agli attributi. Per quanto riguarda la crittografia, AWS Database Encryption SDK utilizza le azioni degli attributi per identificare quali attributi crittografare e firmare, quali firmare (ma non crittografare) e

quali ignorare. Definisci anche gli attributi non firmati consentiti per indicare esplicitamente al client quali attributi sono esclusi dalle firme. Durante la decrittografia, AWS Database Encryption SDK utilizza gli attributi non firmati consentiti definiti dall'utente per identificare quali attributi non sono inclusi nelle firme. Le azioni degli attributi non vengono salvate nell'elemento crittografato e AWS Database Encryption SDK non aggiorna automaticamente le azioni degli attributi.

Scegli attentamente le operazioni di attributo. In caso di dubbio, usa Encrypt and sign (Crittografa e firma). Dopo aver utilizzato AWS Database Encryption SDK per proteggere i tuoi articoli, non puoi modificare un `SIGN_ONLY` attributo `ENCRYPT_AND_SIGN` o un attributo esistente in. `DO_NOTHING` Tuttavia, puoi apportare in sicurezza le seguenti modifiche.

- Aggiungi nuovi `ENCRYPT_AND_SIGNSIGN_ONLY` attributi
- Rimuovi gli `DO_NOTHING` attributi`ENCRYPT_AND_SIGN`,`SIGN_ONLY`, esistenti
- Cambia un `ENCRYPT_AND_SIGN` attributo esistente in `SIGN_ONLY`
- Cambia un `SIGN_ONLY` attributo esistente in `ENCRYPT_AND_SIGN`
- Aggiungi un nuovo `DO_NOTHING` attributo

Considerazioni sulla crittografia ricercabile

Prima di aggiornare il tuo modello di dati, valuta attentamente l'impatto degli aggiornamenti sui beacon che hai creato a partire dagli attributi. Dopo aver scritto nuovi record con un beacon, non è possibile aggiornare la configurazione del beacon. Non è possibile aggiornare le azioni degli attributi associate agli attributi utilizzati per costruire i beacon. Se rimuovi un attributo esistente e il relativo beacon associato, non sarai in grado di interrogare i record esistenti utilizzando quel beacon. Puoi creare nuovi beacon per i nuovi campi che aggiungi al tuo record, ma non puoi aggiornare i beacon esistenti per includere il nuovo campo.

Aggiungi nuovi **`ENCRYPT_AND_SIGNSIGN_ONLY`** attributi

Per aggiungere un nuovo `SIGN_ONLY` attributo `ENCRYPT_AND_SIGN` or, definisci il nuovo attributo nelle azioni degli attributi.

Non è possibile rimuovere un `DO_NOTHING` attributo esistente e aggiungerlo nuovamente come `SIGN_ONLY` attributo `ENCRYPT_AND_SIGN` or.

Utilizzo di una classe di dati annotata

Se hai definito le azioni degli attributi con un`TableSchema`, aggiungi il nuovo attributo alla tua classe di dati annotata. Se non specificate un'annotazione sull'azione dell'attributo per il nuovo attributo,

il client cripterà e firmerà il nuovo attributo per impostazione predefinita (a meno che l'attributo non faccia parte della chiave primaria). Se si desidera firmare solo il nuovo attributo, è necessario aggiungere il nuovo attributo con l'`@DynamoDBEncryptionSignOnly`annotazione.

Utilizzo di un modello a oggetti

Se avete definito manualmente le azioni degli attributi, aggiungete il nuovo attributo alle azioni degli attributi nel modello a oggetti e specificate `ENCRYPT_AND_SIGN` o `SIGN_ONLY` come azione dell'attributo.

Rimuovi gli **`DO_NOTHING`** attributi**`ENCRYPT_AND_SIGN`**,**`SIGN_ONLY`**, esistenti

Se decidi di non aver più bisogno di un attributo, puoi interrompere la scrittura dei dati su quell'attributo oppure puoi rimuoverlo formalmente dalle azioni relative agli attributi. Quando interrompi la scrittura di nuovi dati su un attributo, l'attributo viene comunque visualizzato nelle azioni dell'attributo. Questo può essere utile se è necessario ricominciare a utilizzare l'attributo in futuro. La rimozione formale dell'attributo dalle azioni dell'attributo non lo rimuove dal set di dati. Il tuo set di dati conterrà comunque gli elementi che includono quell'attributo.

Per rimuovere formalmente un `DO_NOTHING` attributo o esistente `ENCRYPT_AND_SIGNSIGN_ONLY`, aggiorna le azioni dell'attributo.

Se rimuovi un `DO_NOTHING` attributo, non devi rimuoverlo dagli attributi non firmati consentiti. Anche se non stai più scrivendo nuovi valori per quell'attributo, il client deve comunque sapere che l'attributo non è firmato per leggere gli elementi esistenti che contengono l'attributo.

Utilizzo di una classe di dati annotata

Se hai definito le azioni degli attributi con un`TableSchema`, rimuovi l'attributo dalla tua classe di dati annotata.

Utilizzo di un modello a oggetti

Se avete definito manualmente le azioni degli attributi, rimuovete l'attributo dalle azioni degli attributi nel modello a oggetti.

Cambia un **`ENCRYPT_AND_SIGN`** attributo esistente in **`SIGN_ONLY`**

Per modificare un `ENCRYPT_AND_SIGN` attributo esistente in`SIGN_ONLY`, è necessario aggiornare le azioni degli attributi. Dopo aver distribuito l'aggiornamento, il client sarà in grado di verificare e decrittografare i valori esistenti scritti nell'attributo, ma firmerà solo i nuovi valori scritti nell'attributo.

Utilizzo di una classe di dati annotata

Se hai definito le azioni degli attributi con un`TableSchema`, aggiorna l'attributo esistente per includere l'`@DynamoDBEncryptionSignOnly`annotazione nella tua classe di dati annotata.

Utilizzo di un modello a oggetti

Se avete definito manualmente le azioni degli attributi, aggiornate l'azione dell'attributo associata all'attributo esistente da `ENCRYPT_AND_SIGN` a `SIGN_ONLY` nel modello a oggetti.

Cambia un **`SIGN_ONLY`** attributo esistente in **`ENCRYPT_AND_SIGN`**

Per modificare un `SIGN_ONLY` attributo esistente in`ENCRYPT_AND_SIGN`, è necessario aggiornare le azioni degli attributi. Dopo aver distribuito l'aggiornamento, il client sarà in grado di verificare i valori esistenti scritti nell'attributo, crittografare e firmare i nuovi valori scritti nell'attributo.

Utilizzo di una classe di dati annotata

Se hai definito le azioni degli attributi con un`TableSchema`, rimuovi l'`@DynamoDBEncryptionSignOnly`annotazione dall'attributo esistente. `SIGN_ONLY`

Utilizzo di un modello a oggetti

Se avete definito manualmente le azioni degli attributi, aggiornate l'azione dell'attributo associata all'attributo da `SIGN_ONLY` a `ENCRYPT_AND_SIGN` nel modello a oggetti.

Aggiungi un nuovo **`DO_NOTHING`** attributo

Per ridurre il rischio di errori durante l'aggiunta di un nuovo `DO_NOTHING` attributo, si consiglia di specificare un prefisso distinto per denominare gli attributi e quindi di utilizzare tale prefisso per definire gli `DO_NOTHING` attributi non firmati consentiti.

Non è possibile rimuovere un `SIGN_ONLY` attributo `ENCRYPT_AND_SIGN` or esistente dalla classe di dati annotata e quindi aggiungere nuovamente l'attributo come attributo. `DO_NOTHING` Puoi aggiungere solo `DO_NOTHING` attributi completamente nuovi.

I passaggi da eseguire per aggiungere un nuovo `DO_NOTHING` attributo dipendono dal fatto che gli attributi non firmati consentiti siano stati definiti esplicitamente in un elenco o con un prefisso.

Utilizzo di un prefisso consentito per gli attributi non firmati

Se hai definito le azioni degli attributi con un`TableSchema`, aggiungi il nuovo `DO_NOTHING` attributo alla classe di dati annotata con l'annotazione. `@DynamoDBEncryptionDoNothing` Se hai definito manualmente le azioni degli attributi, aggiorna le azioni degli attributi per includere il nuovo attributo. Assicurati di configurare in modo esplicito il nuovo attributo con l'azione dell'`DO_NOTHING`attributo. È necessario includere lo stesso prefisso distinto nel nome del nuovo attributo.

Utilizzo di un elenco di attributi non firmati consentiti

1. Aggiungi il nuovo `DO_NOTHING` attributo all'elenco degli attributi non firmati consentiti e distribuisci l'elenco aggiornato.
2. Implementa la modifica a partire dalla Fase 1.

   Non è possibile passare al passaggio 3 finché la modifica non si è propagata a tutti gli host che devono leggere questi dati.
3. Aggiungi il nuovo `DO_NOTHING` attributo alle azioni degli attributi.

   a. Se hai definito le azioni degli attributi con un`TableSchema`, aggiungi il nuovo `DO_NOTHING` attributo alla classe di dati annotata con l'annotazione. `@DynamoDBEncryptionDoNothing`

   b. Se hai definito manualmente le azioni degli attributi, aggiorna le azioni degli attributi per includere il nuovo attributo. Assicurati di configurare in modo esplicito il nuovo attributo con l'azione dell'`DO_NOTHING`attributo.
4. Implementa la modifica a partire dalla Fase 3.

Configurare una tabella DynamoDB esistente per utilizzare il AWS Database Encryption SDK per DynamoDB

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Con la versione 3. x della libreria di crittografia lato client Java per DynamoDB, puoi configurare le tabelle Amazon DynamoDB esistenti per la crittografia lato client. Questo argomento fornisce indicazioni sui tre passaggi da eseguire per aggiungere la versione 3. x su una tabella DynamoDB esistente e popolata.

Prerequisiti

Versione 3. x della libreria di crittografia lato client Java per DynamoDB richiede il DynamoDB Enhanced Client fornito in. AWS SDK for Java 2.x Se usi ancora DynamoDBMapper, devi eseguire la migrazione per utilizzare il DynamoDB Enhanced AWS SDK for Java 2.x Client.

Segui le istruzioni per la migrazione dalla versione 1.x alla 2.x di. AWS SDK for Java

Quindi, segui le istruzioni per iniziare a utilizzare l'API DynamoDB Enhanced Client.

Prima di configurare la tabella per utilizzare la libreria di crittografia lato client Java per DynamoDB, è necessario generare una classe di dati `TableSchema` con annotazioni e creare un client avanzato.

Passaggio 1: Prepararsi a leggere e scrivere elementi crittografati

Completa i seguenti passaggi per preparare il tuo client AWS Database Encryption SDK a leggere e scrivere elementi crittografati. Dopo aver implementato le seguenti modifiche, il client continuerà a leggere e scrivere elementi in testo non crittografato. Non crittograferà né firmerà i nuovi elementi scritti nella tabella, ma sarà in grado di decrittografare gli elementi crittografati non appena vengono visualizzati. Queste modifiche preparano il client a iniziare a crittografare i nuovi elementi. Le seguenti modifiche devono essere implementate su ciascun lettore prima di procedere al passaggio successivo.

1. Definisci le azioni degli attributi

   Aggiorna la tua classe di dati annotata per includere azioni di attributo che definiscono quali valori degli attributi verranno crittografati e firmati, quali saranno solo firmati e quali verranno ignorati.

   Consulta il SimpleClassfile .java nel repository aws-database-encryption-sdk -dynamodb-java su GitHub per ulteriori indicazioni sulle annotazioni di DynamoDB Enhanced Client.

   Per impostazione predefinita, gli attributi della chiave primaria sono firmati ma non crittografati (`SIGN_ONLY`) e tutti gli altri attributi sono crittografati e firmati (`ENCRYPT_AND_SIGN`).
   Per specificare le eccezioni, utilizza le annotazioni di crittografia definite nella libreria di crittografia lato client Java per DynamoDB. Ad esempio, se desideri che un particolare attributo sia segno, usa solo l'`@DynamoDbEncryptionSignOnly`annotazione. Se desideri che un particolare attributo non sia né firmato né crittografato (`DO_NOTHING`), usa l'`@DynamoDbEncryptionDoNothing`annotazione.

   Ad esempio per le annotazioni, vedere. Utilizza una classe di dati annotata

2. Definire quali attributi verranno esclusi dalle firme

   L'esempio seguente presuppone che tutti `DO_NOTHING` gli attributi condividano il prefisso distinto "`:`«e utilizza il prefisso per definire gli attributi non firmati consentiti. Il client presumerà che qualsiasi nome di attributo con il prefisso `:` "" sia escluso dalle firme. Per ulteriori informazioni, consulta Attributi non firmati consentiti.

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

3. Crea un portachiavi

   L'esempio seguente crea un AWS KMSportachiavi. Il AWS KMS portachiavi utilizza la crittografia simmetrica o RSA asimmetrica AWS KMS keys per generare, crittografare e decrittografare le chiavi dati.

   Questo esempio consente `CreateMrkMultiKeyring` di creare un AWS KMS portachiavi con una chiave KMS di crittografia simmetrica. Il `CreateAwsKmsMrkMultiKeyring` metodo garantisce che il portachiavi gestisca correttamente sia le chiavi monoregionali che quelle multiregionali.

   ```
   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. Definire la configurazione di crittografia delle tabelle DynamoDB

   L'esempio seguente definisce una `tableConfigs` mappa che rappresenta la configurazione di crittografia per questa tabella DynamoDB.

   Questo esempio specifica il nome della tabella DynamoDB come nome della tabella logica. Ti consigliamo vivamente di specificare il nome della tabella DynamoDB come nome della tabella logica quando definisci per la prima volta la configurazione di crittografia. Per ulteriori informazioni, consulta Configurazione della crittografia nel AWS Database Encryption SDK per DynamoDB.

   ```
   final Map<String, DynamoDbTableEncryptionConfig> tableConfigs = new HashMap<>();
   ```

```
final DynamoDbTableEncryptionConfig config = DynamoDbTableEncryptionConfig.builder()
        .logicalTableName(ddbTableName)
        .partitionKeyName("partition_key")
        .sortKeyName("sort_key")
        .schemaOnEncrypt(tableSchema)
        .keyring(kmsKeyring)
        .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
        .build();
tableConfigs.put(ddbTableName, config);
```

5. Creazione del **`DynamoDbEncryptionInterceptor`**

   L'esempio seguente crea l'`DynamoDbEncryptionInterceptor`utilizzo del modulo `tableConfigs` Step 3. È necessario specificare `FORCE_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT` come sostituto del testo in chiaro. Questa politica continua a leggere e scrivere elementi di testo non crittografato, a leggere elementi crittografati e a preparare il client a scrivere elementi crittografati.

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

    .plaintextOverride(PlaintextOverride.FORCE_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT)
                   .build())
           .build();
   ```

Fase 2: Scrivere elementi crittografati e firmati

Aggiorna la politica di testo in chiaro nella tua `DynamoDbEncryptionInterceptor` configurazione per consentire al client di scrivere elementi crittografati e firmati. Dopo aver implementato la seguente modifica, il client crittograferà e firmerà i nuovi elementi in base alle azioni degli attributi configurate nel passaggio 1. Il cliente sarà in grado di leggere elementi in testo normale e elementi crittografati e firmati.

Prima di procedere al passaggio 3, è necessario crittografare e firmare tutti gli elementi di testo non crittografato esistenti nella tabella. Non è possibile eseguire alcuna metrica o query per crittografare rapidamente gli elementi di testo in chiaro esistenti. Usa il processo più adatto al tuo sistema. Ad esempio, è possibile utilizzare un processo asincrono che analizza lentamente la tabella e riscrive gli elementi utilizzando le azioni degli attributi e la configurazione di crittografia che hai definito. Per identificare gli elementi in testo non crittografato nella tabella, ti consigliamo di eseguire la scansione

di tutti gli elementi che non contengono gli `aws_dbe_foot` attributi `aws_dbe_head` e che AWS Database Encryption SDK aggiunge agli elementi quando sono crittografati e firmati.

L'esempio seguente crea lo stesso `DynamoDbEncryptionInterceptor` utilizzando lo stesso `tableConfigs` contenuto nel passaggio 1. È necessario aggiornare l'override del testo in chiaro con. `FORBID_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT` Questa politica continua a leggere gli elementi di testo in chiaro, ma anche a leggere e scrivere elementi crittografati.

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

 .plaintextOverride(PlaintextOverride.FORBID_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT)
                .build())
        .build();
```

Passaggio 3: leggi solo gli articoli crittografati e firmati

Dopo aver crittografato e firmato tutti i tuoi articoli, aggiorna l'override in chiaro nella tua `DynamoDbEncryptionInterceptor` configurazione per consentire al client di leggere e scrivere solo gli elementi crittografati e firmati. Dopo aver implementato la seguente modifica, il client crittograferà e firmerà i nuovi elementi in base alle azioni degli attributi configurate nel passaggio 1. Il client sarà in grado di leggere solo elementi crittografati e firmati.

L'esempio seguente crea lo stesso `DynamoDbEncryptionInterceptor` utilizzando lo stesso `tableConfigs` contenuto nel passaggio 1. Puoi aggiornare la sovrascrittura del testo in chiaro `FORBID_WRITE_PLAINTEXT_FORBID_READ_PLAINTEXT` o rimuovere la politica del testo in chiaro dalla tua configurazione. Per impostazione predefinita, il client legge e scrive solo elementi crittografati e firmati.

```
DynamoDbEncryptionInterceptor interceptor = DynamoDbEncryptionInterceptor.builder()
        .config(DynamoDbTablesEncryptionConfig.builder()
                .tableEncryptionConfigs(tableConfigs)
                 // Optional: you can also remove the plaintext policy from your
 configuration

 .plaintextOverride(PlaintextOverride.FORBID_WRITE_PLAINTEXT_FORBID_READ_PLAINTEXT)
                .build())
        .build();
```

Esegui la migrazione alla versione 3.x della libreria di crittografia lato client Java per DynamoDB

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Versione 3. x della libreria di crittografia lato client Java per DynamoDB è un'importante riscrittura della 2. codice base x. Include molti aggiornamenti, come un nuovo formato di dati strutturati, un supporto multitenancy migliorato, modifiche allo schema senza interruzioni e supporto per la crittografia ricercabile. Questo argomento fornisce indicazioni su come migrare il codice alla versione 3. x.

Migrazione dalla versione 1.x alla 2.x

Esegui la migrazione alla versione 2. x prima della migrazione alla versione 3. x. Versione 2. x ha cambiato il simbolo del provider più recente da `MostRecentProvider` a`CachingMostRecentProvider`. Se attualmente utilizzi la versione 1. x della libreria di crittografia lato client Java per DynamoDB con il `MostRecentProvider` simbolo, devi aggiornare il nome del simbolo nel codice a. `CachingMostRecentProvider` Per ulteriori informazioni, consulta Aggiornamenti al provider più recente.

Migrazione dalla versione 2.x alla 3.x

Le procedure seguenti descrivono come migrare il codice dalla versione 2. x alla versione 3. x della libreria di crittografia lato client Java per DynamoDB.

Fase 1. Preparati a leggere gli elementi nel nuovo formato

Completa i seguenti passaggi per preparare il tuo client AWS Database Encryption SDK a leggere gli elementi nel nuovo formato. Dopo aver implementato le seguenti modifiche, il client continuerà a comportarsi come nella versione 2. x. Il tuo cliente continuerà a leggere e scrivere elementi nella versione 2. formato x, ma queste modifiche preparano il client a leggere gli elementi nel nuovo formato.

Aggiorna il tuo AWS SDK for Java alla versione 2.x

Versione 3. x della libreria di crittografia lato client Java per DynamoDB richiede il DynamoDB Enhanced Client. Il DynamoDB Enhanced Client sostituisce il DynamoDBMapper utilizzato nelle versioni precedenti. Per utilizzare il client avanzato, è necessario utilizzareAWS SDK for Java 2.x.

Segui le istruzioni per la migrazione dalla versione 1.x alla 2.x di. AWS SDK for Java

Per ulteriori informazioni sui AWS SDK for Java 2.x moduli necessari, vederePrerequisiti.

Configura il tuo client per leggere gli elementi crittografati dalle versioni precedenti

Le procedure seguenti forniscono una panoramica dei passaggi illustrati nell'esempio di codice riportato di seguito.

1. Crea un portachiavi.

   I portachiavi e i gestori dei materiali crittografici sostituiscono i fornitori di materiali crittografici utilizzati nelle versioni precedenti della libreria di crittografia lato client Java per DynamoDB.

   > **Important**
   >
   > Le chiavi di avvolgimento specificate durante la creazione di un portachiavi devono essere le stesse chiavi di avvolgimento utilizzate con il fornitore di materiali crittografici nella versione 2. x.

2. Crea uno schema di tabella sulla tua classe annotata.

   Questo passaggio definisce le azioni relative agli attributi che verranno utilizzate quando inizierete a scrivere elementi nel nuovo formato.

   Per indicazioni sull'utilizzo del nuovo client avanzato di DynamoDB, consulta la sezione Genera un `TableSchema` nella Guida per gli AWS SDK for Javasviluppatori.

   L'esempio seguente presuppone che tu abbia aggiornato la tua classe annotata dalla versione 2. x utilizzando le nuove annotazioni sulle azioni degli attributi. Per ulteriori indicazioni sull'annotazione delle azioni degli attributi, vedere. Utilizza una classe di dati annotata

3. Definire quali attributi sono esclusi dalla firma.
4. Configura una mappa esplicita delle azioni degli attributi configurate nella classe modellata della tua versione 2.x.

Questo passaggio definisce le azioni degli attributi utilizzate per scrivere gli elementi nel vecchio formato.

5. Configura quello `DynamoDBEncryptor` che hai usato nella versione 2. x della libreria di crittografia lato client Java per DynamoDB.
6. Configura il comportamento precedente.
7. Creare un `DynamoDbEncryptionInterceptor`.
8. Crea un nuovo client AWS SDK DynamoDB.
9. Crea `DynamoDBEnhancedClient` e crea una tabella con la tua classe modellata.

   Per ulteriori informazioni sul client avanzato di DynamoDB, consulta creare un client avanzato.

```
public class MigrationExampleStep1 {

    public static void MigrationStep1(String kmsKeyId, String ddbTableName, int
 sortReadValue) {
        // 1. Create a Keyring.
        //    This example creates an AWS KMS Keyring that specifies the
        //    same kmsKeyId previously used in the version 2.x configuration.
        //    It uses the 'CreateMrkMultiKeyring' method to create the
        //    keyring, so that the keyring can correctly handle both single
        //    region and Multi-Region KMS Keys.
        //    Note that this example uses the AWS SDK for Java v2 KMS client.
        final MaterialProviders matProv = MaterialProviders.builder()
                .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
                .build();
        final CreateAwsKmsMrkMultiKeyringInput keyringInput =
 CreateAwsKmsMrkMultiKeyringInput.builder()
                .generator(kmsKeyId)
                .build();
        final IKeyring kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);

        // 2. Create a Table Schema over your annotated class.
        //    For guidance on using the new attribute actions
        //    annotations, see SimpleClass.java in the
        //    aws-database-encryption-sdk-dynamodb-java GitHub repository.
        //    All primary key attributes must be signed but not encrypted
        //    (SIGN_ONLY) and by default all non-primary key attributes
        //    are encrypted and signed (ENCRYPT_AND_SIGN).
```

```
        //    If you want a particular non-primary key attribute to be signed but
        //    not encrypted, use the 'DynamoDbEncryptionSignOnly' annotation.
        //    If you want a particular attribute to be neither signed nor encrypted
        //    (DO_NOTHING), use the 'DynamoDbEncryptionDoNothing' annotation.
        final TableSchema<SimpleClass> schemaOnEncrypt =
TableSchema.fromBean(SimpleClass.class);

        // 3. Define which attributes the client should expect to be excluded
        //    from the signature when reading items.
        //    This value represents all unsigned attributes across the entire
        //    dataset.
        final List<String> allowedUnsignedAttributes = Arrays.asList("attribute3");

        // 4. Configure an explicit map of the attribute actions configured
        //    in your version 2.x modeled class.
        final Map<String, CryptoAction> legacyActions = new HashMap<>();
        legacyActions.put("partition_key", CryptoAction.SIGN_ONLY);
        legacyActions.put("sort_key", CryptoAction.SIGN_ONLY);
        legacyActions.put("attribute1", CryptoAction.ENCRYPT_AND_SIGN);
        legacyActions.put("attribute2", CryptoAction.SIGN_ONLY);
        legacyActions.put("attribute3", CryptoAction.DO_NOTHING);

        // 5. Configure the DynamoDBEncryptor that you used in version 2.x.
        final AWSKMS kmsClient = AWSKMSClientBuilder.defaultClient();
        final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kmsClient,
kmsKeyId);
        final DynamoDBEncryptor oldEncryptor = DynamoDBEncryptor.getInstance(cmp);

        // 6. Configure the legacy behavior.
        //    Input the DynamoDBEncryptor and attribute actions created in
        //    the previous steps. For Legacy Policy, use
        //    'FORCE_LEGACY_ENCRYPT_ALLOW_LEGACY_DECRYPT'. This policy continues to
read
        //    and write items using the old format, but will be able to read
        //    items written in the new format as soon as they appear.
        final LegacyOverride legacyOverride = LegacyOverride
                .builder()
                .encryptor(oldEncryptor)
                .policy(LegacyPolicy.FORCE_LEGACY_ENCRYPT_ALLOW_LEGACY_DECRYPT)
                .attributeActionsOnEncrypt(legacyActions)
                .build();

        // 7. Create a DynamoDbEncryptionInterceptor with the above configuration.
```

```
        final Map<String, DynamoDbEnhancedTableEncryptionConfig> tableConfigs = new
 HashMap<>();
        tableConfigs.put(ddbTableName,
                DynamoDbEnhancedTableEncryptionConfig.builder()
                        .logicalTableName(ddbTableName)
                        .keyring(kmsKeyring)
                        .allowedUnsignedAttributes(allowedUnsignedAttributes)
                        .schemaOnEncrypt(tableSchema)
                        .legacyOverride(legacyOverride)
                        .build());
        final DynamoDbEncryptionInterceptor interceptor =
                DynamoDbEnhancedClientEncryption.CreateDynamoDbEncryptionInterceptor(
                        CreateDynamoDbEncryptionInterceptorInput.builder()
                                .tableEncryptionConfigs(tableConfigs)
                                .build()
                );

        // 8. Create a new AWS SDK DynamoDb client using the
        //    interceptor from Step 7.
        final DynamoDbClient ddb = DynamoDbClient.builder()
                .overrideConfiguration(
                        ClientOverrideConfiguration.builder()
                                .addExecutionInterceptor(interceptor)
                                .build())
                .build();

        // 9. Create the DynamoDbEnhancedClient using the AWS SDK DynamoDb client
        //    created in Step 8, and create a table with your modeled class.
        final DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();
        final DynamoDbTable<SimpleClass> table = enhancedClient.table(ddbTableName,
 tableSchema);
    }
}
```

Fase 2. Scrivere elementi nel nuovo formato

Dopo aver implementato le modifiche dello Step 1 a tutti i lettori, completa i seguenti passaggi per configurare il tuo client AWS Database Encryption SDK per scrivere gli elementi nel nuovo formato. Dopo aver implementato le seguenti modifiche, il client continuerà a leggere gli elementi nel vecchio formato e inizierà a scrivere e leggere gli elementi nel nuovo formato.

Le procedure seguenti forniscono una panoramica dei passaggi illustrati nell'esempio di codice riportato di seguito.

1. Continua a configurare il tuo portachiavi, lo schema della tabella, **`allowedUnsignedAttributes`** le azioni degli attributi precedenti e **`DynamoDBEncryptor`** come hai fatto nel passaggio 1.
2. Aggiorna il tuo comportamento precedente per scrivere solo nuovi elementi utilizzando il nuovo formato.
3. Creazione di una destinazione `DynamoDbEncryptionInterceptor`
4. Crea un nuovo client AWS SDK DynamoDB.
5. Crea `DynamoDBEnhancedClient` e crea una tabella con la tua classe modellata.

   Per ulteriori informazioni sul client avanzato di DynamoDB, consulta creare un client avanzato.

```
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);
    }
}
```

Dopo aver implementato le modifiche della Fase 2, è necessario crittografare nuovamente tutti i vecchi elementi della tabella con il nuovo formato prima di poter continuare con il passaggio 3. Non è possibile eseguire alcuna metrica o query per crittografare rapidamente gli elementi esistenti. Usa il processo più adatto al tuo sistema. Ad esempio, è possibile utilizzare un processo asincrono che analizza lentamente la tabella e riscrive gli elementi utilizzando le nuove azioni degli attributi e la configurazione di crittografia che hai definito.

Fase 3. Leggi e scrivi solo elementi nel nuovo formato

Dopo aver ricrittografato tutti gli elementi della tabella con il nuovo formato, puoi rimuovere il comportamento precedente dalla tua configurazione. Completa i seguenti passaggi per configurare il tuo client in modo che legga e scriva solo elementi nel nuovo formato.

Le procedure seguenti forniscono una panoramica dei passaggi illustrati nell'esempio di codice riportato di seguito.

1. Continua a configurare il tuo portachiavi, lo schema della tabella e `allowedUnsignedAttributes` come hai fatto nel passaggio 1. Rimuovi le azioni degli attributi precedenti e `DynamoDBEncryptor` dalla tua configurazione.
2. Creare un `DynamoDbEncryptionInterceptor`.
3. Crea un nuovo client AWS SDK DynamoDB.
4. Crea `DynamoDBEnhancedClient` e crea una tabella con la tua classe modellata.

   Per ulteriori informazioni sul client avanzato di DynamoDB, consulta creare un client avanzato.

```
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);
    }
}
```

Client di crittografia DynamoDB Legacy

Il 9 giugno 2023, la nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Il AWS Database Encryption SDK continua a supportare le versioni precedenti di DynamoDB Encryption Client. Per ulteriori informazioni sulle diverse parti della libreria di crittografia lato client che sono state modificate con la ridenominazione, vedere. Rinomina del client di crittografia Amazon DynamoDB

Per eseguire la migrazione alla versione più recente della libreria di crittografia lato client Java per DynamoDB, vedere. Esegui la migrazione alla versione 3.x

Argomenti

- AWSSupporto per la versione di Database Encryption SDK per DynamoDB
- Come funziona il client di crittografia DynamoDB
- Concetti relativi ai client di crittografia Amazon DynamoDB
- Fornitore di materiali crittografici
- Linguaggi di programmazione disponibili per Amazon DynamoDB Encryption Client
- Modifica del modello di dati
- Risoluzione dei problemi nell'applicazione DynamoDB Encryption Client

AWSSupporto per la versione di Database Encryption SDK per DynamoDB

Gli argomenti del capitolo Legacy forniscono informazioni sulle versioni 1. x —2. x del client di crittografia DynamoDB per Java e versioni 1. x —3. x del client di crittografia DynamoDB per Python.

La tabella seguente elenca le lingue e le versioni che supportano la crittografia lato client in Amazon DynamoDB.

Linguaggio di programmazione	Versione	Fase del ciclo di vita della versione principale dell'SDK
Java	Versioni 1. x	Fase di fine del supporto, a partire da luglio 2022
Java	Versioni 2. x	Disponibilità generale (GA)
Java	Versione 3. x	Disponibilità generale (GA)
Python	Versioni 1. x	Fase di fine del supporto, a partire da luglio 2022
Python	Versioni 2. x	Fase di fine del supporto, a partire da luglio 2022
Python	Versioni 3. x	Disponibilità generale (GA)

Come funziona il client di crittografia DynamoDB

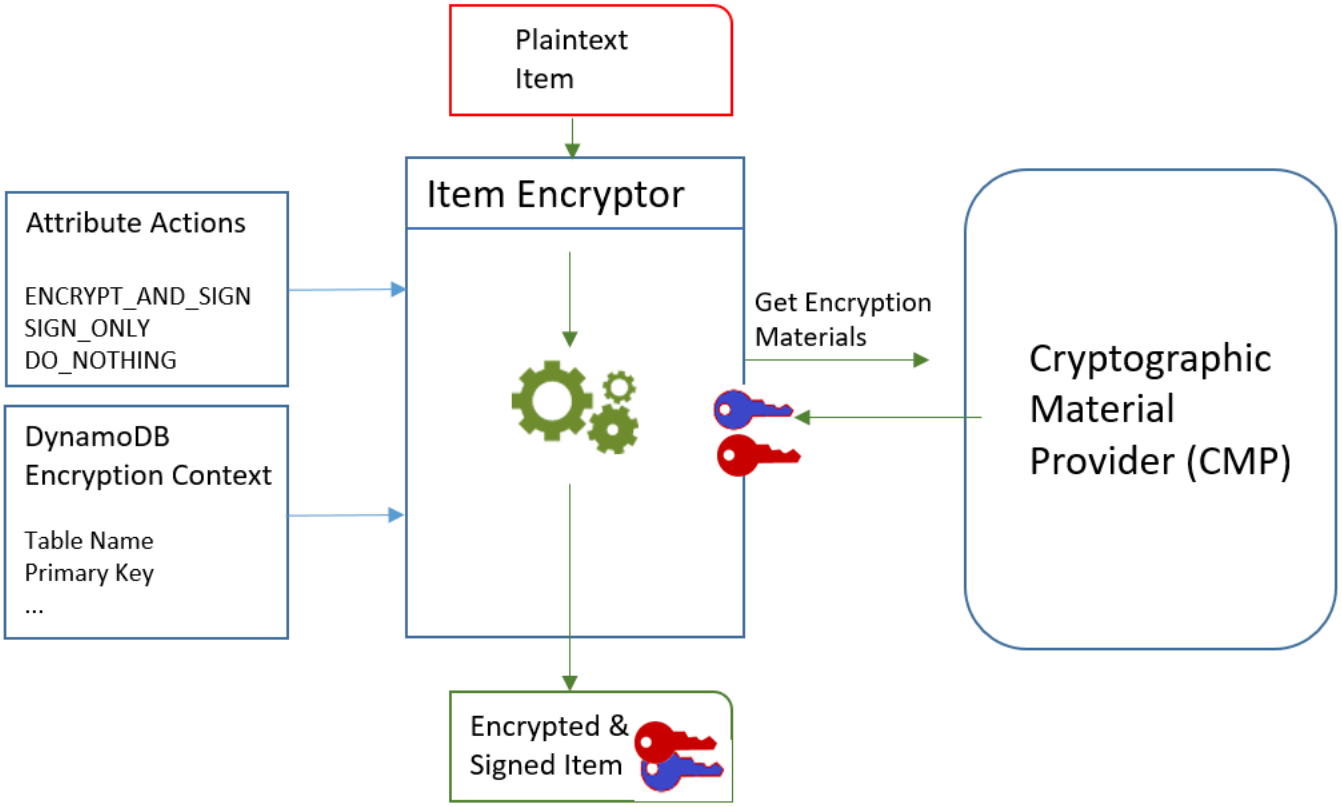
> **Note**
>
> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. L'argomento seguente fornisce informazioni sulle versioni 1. x —2. x del client di crittografia DynamoDB per Java e versioni 1. x —3. x del client di crittografia DynamoDB per Python. Per ulteriori informazioni, consulta AWSDatabase Encryption SDK per il supporto delle versioni di DynamoDB.

Il client di crittografia DynamoDB è progettato specificamente per proteggere i dati archiviati in DynamoDB. Le librerie includono implementazioni sicure che puoi estendere o utilizzare senza modificarle. Inoltre, la maggior parte degli elementi sono rappresentati da elementi astratti per consentirti di creare e utilizzare componenti personalizzati compatibili.

Crittografia e firma degli item della tabella

Alla base del DynamoDB Encryption Client c'è un criptatore di elementi che crittografa, firma, verifica e decrittografa gli elementi della tabella. Carica informazioni sugli elementi della tabella e istruzioni su quali item criptare e firmare. Ottiene i materiali di crittografia e le istruzioni su come utilizzarli da un provider di materiali crittografici da te selezionato e configurato.

Il seguente diagramma mostra una vista generale di questo processo:



Per crittografare e firmare un elemento della tabella, il client di crittografia DynamoDB necessita di:

- Informazioni sulla tabella. Ottiene informazioni sulla tabella da un contesto di crittografia DynamoDB fornito dall'utente. Alcuni assistenti ottengono le informazioni richieste da DynamoDB e creano automaticamente il contesto di crittografia DynamoDB.

> **Note**
>
> Il contesto di crittografia DynamoDB nel client di crittografia DynamoDB non è correlato al contesto di crittografia in AWS Key Management Service () AWS KMS e. AWS Encryption SDK

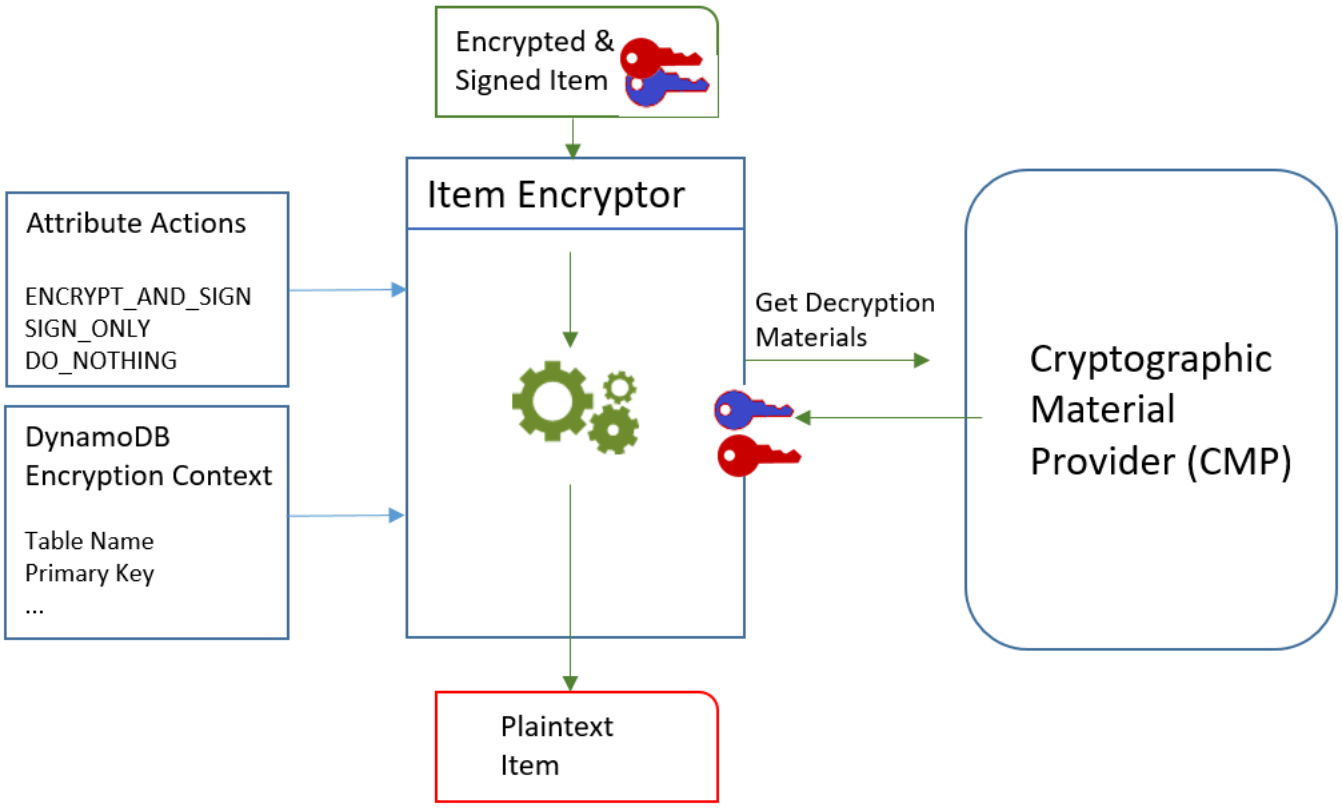
- Informazioni su quali attributi crittografare e firmare. Ottiene queste informazioni dalle operazioni di attributo da te fornite.
- Materiali di crittografia, incluse le chiavi di crittografia e di firma. Li ottiene da un provider di materiali crittografici (CMP) da te selezionato e configurato.
- Istruzioni per la crittografia e la firma dell'item. Il CMP aggiunge istruzioni per l'utilizzo dei materiali di crittografia, inclusi gli algoritmi di crittografia e firma, alla descrizione dei materiali effettivi.

Il componente di crittografia dell'item utilizza tutti questi elementi per crittografare e firmare l'item. Il componente di crittografia dell'item, inoltre, aggiunge due attributi all'item: un attributo di descrizione del materiale, contenente le istruzioni per la crittografia e la firma (la descrizione dei materiali effettivi), e un attributo che contiene la firma. Puoi interagire direttamente con il componente di crittografia dell'item o utilizzare le funzionalità helper che interagiscono con il componente di crittografia dell'item per implementare un comportamento predefinito sicuro.

Il risultato è un item DynamoDB contenente dati crittografati e firmati.

Verifica e decrittografia degli item della tabella

Anche questi componenti operano insieme per verificare e decrittografare l'item, come mostrato nel diagramma riportato di seguito.



Per verificare e decrittografare un elemento, il client di crittografia DynamoDB necessita degli stessi componenti, componenti con la stessa configurazione o componenti appositamente progettati per decrittografare gli elementi, come segue:

- Informazioni sulla tabella dal contesto di crittografia DynamoDB.
- Quali attributi verificare e decrittografare. ottenute dalle operazioni di attributo.
- Materiali di decrittografia, incluse le chiavi di verifica e decrittografia, ottenute dal CMP da te selezionato e configurato.

  L'item crittografato non include alcun record del CMP che è stato utilizzato per crittografarlo. Devi fornire lo stesso CMP, un CMP con la stessa configurazione o un CMP che è stato designato per decrittografare gli item.
- Informazioni sui modi in cui l'item è stato crittografato e firmato, inclusi gli algoritmi di crittografia e firma. Il client le ottiene dall'attributo di descrizione del materiale nell'item.

Il componente di crittografia dell'item utilizza tutti questi elementi per verificare e decrittografare l'item. Inoltre, rimuove gli attributi di descrizione del materiale e di firma. Il risultato è un elemento DynamoDB in testo semplice.

Concetti relativi ai client di crittografia Amazon DynamoDB

> **Note**
>
> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. L'argomento seguente fornisce informazioni sulle versioni 1. x —2. x del client di crittografia DynamoDB per Java e versioni 1. x —3. x del client di crittografia DynamoDB per Python. Per ulteriori informazioni, consulta AWSDatabase Encryption SDK per il supporto delle versioni di DynamoDB.

Questo argomento spiega i concetti e la terminologia utilizzati nel client di crittografia Amazon DynamoDB.

Per scoprire come interagiscono i componenti del client di crittografia DynamoDB, consulta. Come funziona il client di crittografia DynamoDB

Argomenti

- Provider di materiali crittografici
- Componenti di crittografia dell'item
- Operazioni di attributo
- Descrizione dei materiali
- Contesto di crittografia DynamoDB
- Archivio provider

Provider di materiali crittografici

Quando si implementa il client di crittografia DynamoDB, una delle prime attività è selezionare un fornitore di materiali crittografici (CMP) (noto anche come fornitore di materiali crittografici). La scelta determina gran parte del processo di implementazione rimanente.

Un CMP raccoglie, assembla e restituisce i materiali crittografici che il componente di crittografia dell'item utilizza per crittografare e firmare gli item della tabella. Il CMP determina gli algoritmi di crittografia da utilizzare e come generare e proteggere le chiavi di crittografia e firma.

Il CMP interagisce con il componente di crittografia dell'item. Il componente di crittografia dell'item richiede i materiali di crittografia e decrittografia al CMP e il CMP li restituisce al componente di crittografia dell'item. Quindi, il componente di crittografia dell'item utilizza i materiali crittografici per crittografare e firmare l'item o per verificarlo e decrittografarlo.

Il CMP viene specificato quando si configura il client. Puoi creare un CMP personalizzato compatibile o utilizzare uno dei molti CMP contenuti nella libreria. La maggior parte dei CMP è disponibile per più linguaggi di programmazione.

Componenti di crittografia dell'item

L'item encryptor è un componente di livello inferiore che esegue operazioni crittografiche per il client di crittografia DynamoDB. Richiede materiali crittografici a un CMP, quindi utilizza i materiali che il CMP restituisce per crittografare e firmare l'item della tabella o per verificarlo e decrittografarlo.

Puoi interagire direttamente con il componente di crittografia dell'item o utilizzare gli helper forniti dalla libreria. Ad esempio, il client di crittografia DynamoDB per Java include una classe di `AttributeEncryptor` supporto che è possibile utilizzare con`DynamoDBMapper`, anziché interagire direttamente con l'elemento crittografico. `DynamoDBEncryptor` La libreria Python include le classi helper `EncryptedTable`, `EncryptedClient` ed `EncryptedResource` che interagiscono con il componente di crittografia dell'item per te.

Operazioni di attributo

Le operazioni di attributo comunicano al componente di crittografia dell'item quali operazioni effettuare su ciascun attributo dell'item.

È possibile utilizzare i seguenti valori per le operazioni di attributo:

- Crittografa e firma: crittografa il valore dell'attributo. Includi l'attributo (nome e valore) nella firma dell'item.
- Solo firma: includi l'attributo nella firma dell'articolo.
- Non fare nulla: non crittografare o firmare l'attributo.

Utilizza Encrypt and sign (Crittografa e firma) per tutti gli attributi in grado di memorizzare dati sensibili. Per gli attributi delle chiavi primarie (chiave di partizione e chiave di ordinamento), utilizza Sign only (Firma soltanto). L'attributo di descrizione del materiale e l'attributo di firma non sono firmati o crittografati. Non è necessario specificare operazioni di attributo per questi attributi.

Scegli attentamente le operazioni di attributo. In caso di dubbio, usa Encrypt and sign (Crittografa e firma). Dopo aver utilizzato il client di crittografia DynamoDB per proteggere gli elementi della tabella, non è possibile modificare l'azione di un attributo senza rischiare un errore di convalida della firma. Per informazioni dettagliate, consultare Modifica del modello di dati.

> ⚠ Warning
>
> Non crittografare gli attributi che vengono usati per la chiave primaria. Devono rimanere in testo normale in modo che DynamoDB possa trovare l'elemento senza eseguire una scansione completa della tabella.

Se il contesto di crittografia DynamoDB identifica gli attributi della chiave primaria, il client genererà un errore se si tenta di crittografarli.

La tecnica da utilizzare per specificare le operazioni di attributo è diversa per ogni linguaggio di programmazione. Potrebbe anche essere specifica per le classi helper che utilizzi.

Per ulteriori informazioni, consulta la documentazione relativa al tuo linguaggio di programmazione.

- Python
- Java

Descrizione dei materiali

La descrizione dei materiali per un item della tabella crittografato comprende informazioni, come gli algoritmi di crittografia, sui modi in cui l'item della tabella viene crittografato e firmato. Il CMP registra la descrizione dei materiali poiché assembla i materiali crittografici per la crittografia e la firma. Successivamente, quando deve assemblare i materiali crittografici per verificare e decrittografare l'item, utilizza la descrizione dei materiali come guida.

Nel DynamoDB Encryption Client, la descrizione del materiale si riferisce a tre elementi correlati:

Descrizione dei materiali richiesti

Alcuni CMP consentono di specificare opzioni avanzate, come l'algoritmo di crittografia. Per indicare le tue scelte, aggiungi coppie nome-valore alla proprietà di descrizione del materiale del contesto di crittografia DynamoDB nella tua richiesta di crittografare un elemento della tabella. Questo elemento è noto come descrizione dei materiali richiesti. I valori validi nella descrizione dei materiali richiesti sono definiti dal CMP scelto.

> **Note**
>
> Poiché la descrizione dei materiali può sovrascrivere i valori predefiniti, ti consigliamo di omettere la descrizione dei materiali richiesti se non hai un valido motivo per utilizzarla.

Descrizione dei materiali effettivi

La descrizione dei materiali che i CMP restituiscono è nota come descrizione dei materiali effettivi. Descrive i valori effettivi che il CMP ha utilizzato quando ha assemblato i materiali crittografici. In genere comprende l'eventuale descrizione dei materiali richiesti, con aggiunte e modifiche.

Attributo di descrizione del materiale

Il client salva la descrizione dei materiali effettivi nell'attributo di descrizione del materiale dell'item crittografato. Il nome dell'attributo di descrizione del materiale è `amzn-ddb-map-desc` e il suo valore è la descrizione dei materiali effettivi. Il client utilizza i valori dell'attributo di descrizione del materiale per verificare e decrittografare l'item.

Contesto di crittografia DynamoDB

Il contesto di crittografia DynamoDB fornisce informazioni sulla tabella e sull'elemento al fornitore di materiali crittografici (CMP). Nelle implementazioni avanzate, il contesto di crittografia DynamoDB può includere una descrizione del materiale richiesta.

Quando si crittografano gli elementi della tabella, il contesto di crittografia DynamoDB è legato crittograficamente ai valori degli attributi crittografati. Quando si decrittografa, se il contesto di crittografia DynamoDB non corrisponde esattamente, con distinzione tra maiuscole e minuscole, al contesto di crittografia DynamoDB utilizzato per la crittografia, l'operazione di decrittografia ha esito negativo. Se interagisci direttamente con il crittografo degli elementi, devi fornire un contesto di crittografia DynamoDB quando chiami un metodo di crittografia o decrittografia. La maggior parte degli helper crea il contesto di crittografia DynamoDB per te.

> **Note**
>
> Il contesto di crittografia DynamoDB nel client di crittografia DynamoDB non è correlato al contesto di crittografia in AWS Key Management Service () AWS KMS e. AWS Encryption SDK

Il contesto di crittografia DynamoDB può includere i seguenti campi. Tutti i campi e i valori sono facoltativi.

- Nome tabella
- Nome della chiave di partizione
- Nome della chiave di ordinamento
- Coppie nome/valore degli attributi
- Descrizione dei materiali richiesti

Archivio provider

Un archivio provider è un componente che restituisce CMP. L'archivio provider può creare i CMP o ottenerli da un'altra origine, ad esempio un altro archivio provider. L'archivio provider memorizza le versioni dei CMP che crea in un dispositivo storage persistente in cui ogni CMP archiviato viene identificato del nome dei materiali del richiedente e dal numero di versione.

Il provider più recente del client di crittografia DynamoDB ottiene i propri CMP da un archivio del provider, ma è possibile utilizzare l'archivio del provider per fornire CMP a qualsiasi componente. Ciascun provider più recente è associato a un singolo archivio provider, ma un archivio provider può fornire CMP a molti richiedenti tramite più host.

L'archivio provider crea nuove versioni di CMP on demand e restituisce versioni nuove ed esistenti. Restituisce inoltre l'ultimo numero di versione per un dato nome di materiale. Questo consente al richiedente di sapere quando l'archivio provider dispone di una nuova versione del suo CMP che è possibile richiedere.

Il client di crittografia DynamoDB include un MetaStorearchivio di provider che crea Wrapped CMP con chiavi archiviate in DynamoDB e crittografate utilizzando un client di crittografia DynamoDB interno.

Ulteriori informazioni:

- Archivio provider: Java, Python
- MetaStore: Java, Python

Fornitore di materiali crittografici

> **Note**
>
> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. L'argomento seguente fornisce informazioni sulle versioni 1. x —2. x del client di crittografia DynamoDB per Java e versioni 1. x —3. x del client di crittografia DynamoDB per Python. Per ulteriori informazioni, consulta AWSDatabase Encryption SDK per il supporto delle versioni di DynamoDB.

Una delle decisioni più importanti da prendere quando si utilizza il client di crittografia DynamoDB è la selezione di un fornitore di materiali crittografici (CMP). Il CMP raccoglie e restituisce i materiali crittografici al componente di crittografia dell'item. Determina inoltre la modalità di generazione delle chiavi di crittografia e di firma, gli algoritmi di crittografia e firma utilizzati e se i nuovi materiali di chiave devono essere creati o riutilizzati per ciascun item.

Puoi scegliere una CMP tra le implementazioni fornite nelle librerie DynamoDB Encryption Client o creare una CMP personalizzata compatibile. La scelta del CMP potrebbe inoltre dipendere dal linguaggio di programmazione utilizzato.

Questo argomento descrive i CMP più comuni con dei consigli per aiutarti a fare la scelta migliore per la tua applicazione.

Provider di materiali KMS diretto

Direct KMS Materials Provider protegge gli articoli della tua tavola con un sistema AWS KMS keyche non esce mai AWS Key Management Service(AWS KMS) in modo non crittografato. Non è necessario che l'applicazione generi o gestisca i materiali crittografici. Dal momento che utilizza la AWS KMS key per generare chiavi di crittografia e firma univoche per ciascun item, questo provider richiama AWS KMS ogni volta che crittografa o decrittografa un item.

Questo provider rappresenta una scelta adeguata se utilizzi AWS KMS e se per la tua applicazione è utile una chiamata AWS KMS per transazione.

Per informazioni dettagliate, consultare Provider di materiali KMS diretto.

Provider di materiali di sottoposti a wrapping (CMP di sottoposti a wrapping)

Il Wrapped Materials Provider (Wrapped CMP) consente di generare e gestire le chiavi di wrapping e firma all'esterno del client di crittografia DynamoDB.

Il CMP di sottoposti a wrapping genera una chiave di crittografia univoca per ciascun item. Utilizza quindi le chiavi di firma e di crittografia (o di annullamento della crittografia) da te fornite. In questo modo, puoi determinare la modalità di generazione delle chiavi di crittografia e di firma e se sono univoche o se vengono riutilizzate per ciascun item. Il CMP di sottoposti a wrapping è un'alternativa sicura al provider KMS diretto per le applicazioni che non utilizzano AWS KMS ed è in grado di gestire in modo sicuro i materiali crittografici.

Per informazioni dettagliate, consultare Provider di materiali sottoposti a wrapping.

Provider più recente

Il provider più recente è un provider di materiali crittografici (CMP) progettato per funzionare con gli archivi del provider. Ottiene i CMP dall'archivio del provider e recupera i materiali crittografici che restituisce dai CMP. Il Provider più recente in genere utilizza ciascun CMP per soddisfare più richieste di materiali crittografici, ma puoi utilizzare le funzioni dell'archivio del provider per controllare in quale misura vengono riutilizzati i materiali, determinare la frequenza di rotazione del CMP e persino modificare il tipo di CMP utilizzato senza cambiare il Provider più recente.

Puoi utilizzare il Provider più recente con qualsiasi archivio del provider compatibile. Il client di crittografia DynamoDB include unMetaStore, che è un negozio di provider che restituisce Wrapped CMP.

Il Provider più recente rappresenta una scelta adeguata per le applicazioni che devono ridurre al minimo le chiamate alla relativa origine crittografica e per le applicazioni che possono riutilizzare alcuni materiali crittografici senza violare i requisiti di sicurezza. Ad esempio, ti consente di proteggere i tuoi materiali crittografici con un AWS KMS keyin AWS Key Management Service(AWS KMS) senza chiamare AWS KMS ogni volta che crittografi o decrittografi un elemento.

Per informazioni dettagliate, consultare Provider più recente.

Provider di materiali statici

Static Materials Provider è progettato per test, proof-of-concept dimostrazioni e compatibilità con versioni precedenti. Non genera nessun materiale crittografico univoco per gli item. Restituisce

le stesse chiavi di crittografia e di firma da te fornite che vengono utilizzate direttamente per crittografare, decrittografare e firmare gli item della tabella.

> **Note**
>
> Il Provider di statici asimmetrico nella libreria Java non è un provider di statici. Fornisce soltanto costruttori alternativi per il CMP di sottoposti a wrapping. Puoi utilizzarlo nell'ambiente di produzione, ma ti consigliamo di utilizzare direttamente il CMP di sottoposti a wrapping ogni qualvolta sia possibile.

Argomenti

- Provider di materiali KMS diretto
- Provider di materiali sottoposti a wrapping
- Provider più recente
- Provider di materiali statici

Provider di materiali KMS diretto

> **Note**
>
> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. L'argomento seguente fornisce informazioni sulle versioni 1. x —2. x del client di crittografia DynamoDB per Java e versioni 1. x —3. x del client di crittografia DynamoDB per Python. Per ulteriori informazioni, consulta AWSDatabase Encryption SDK per il supporto delle versioni di DynamoDB.

Il Direct KMS Materials Provider (Direct KMS Provider) protegge gli elementi della tabella con un sistema AWS KMS keyche non esce mai AWS Key Management Service(AWS KMS) in modo non crittografato. Questo provider di materiali crittografici restituisce una chiave di crittografia e una chiave di firma univoche per ogni item della tabella. A questo scopo, richiama AWS KMS ogni volta che effettui la crittografia o la decrittografia di un item.

Se stai elaborando elementi DynamoDB ad alta frequenza e su larga scala, potresti superare i AWS KMS requests-per-secondlimiti, causando ritardi nell'elaborazione. Se devi superare un limite, crea

un caso nel AWS SupportCentro. Potresti anche prendere in considerazione l'utilizzo di un fornitore di materiali crittografici con un riutilizzo limitato delle chiavi, come il fornitore più recente.

Per utilizzare il Direct KMS Provider, il chiamante deve disporreAccount AWS, almeno unoAWS KMS key, dell'autorizzazione per chiamare GenerateDataKeye decrittografare le operazioni su. AWS KMS key AWS KMS keyDeve essere una chiave di crittografia simmetrica; il client di crittografia DynamoDB non supporta la crittografia asimmetrica. Se utilizzi una tabella globale DynamoDB, potresti voler specificare una chiave AWS KMSmultiregionale. Per informazioni dettagliate, consultare Come utilizzarlo.

> **Note**
>
> Quando utilizzi Direct KMS Provider, i nomi e i valori degli attributi della chiave primaria vengono visualizzati in chiaro nel contesto di AWS KMS crittografia e nei AWS CloudTrail registri delle operazioni correlate. AWS KMS Tuttavia, il client di crittografia DynamoDB non espone mai il testo in chiaro di alcun valore di attributo crittografato.

Il Direct KMS Provider è uno dei numerosi fornitori di materiali crittografici (CMP) supportati dal client di crittografia DynamoDB. Per informazioni sugli altri CMP, consulta Fornitore di materiali crittografici.

Per il codice di esempio, consulta:

- Java: AwsKmsEncryptedItem
- Pitone: aws-kms-encrypted-table, aws-kms-encrypted-item

Argomenti

- Come utilizzarlo
- Come funziona

Come utilizzarlo

Per creare un provider KMS diretto, utilizza il parametro key ID per specificare una chiave KMS di crittografia simmetrica nel tuo account. Il valore del parametro key ID può essere l'ID chiave, l'ARN della chiave, il nome dell'alias o l'alias ARN di. AWS KMS key Per i dettagli sugli identificatori chiave, consulta Identificatori chiave nella Guida per gli sviluppatori. AWS Key Management Service

Il Direct KMS Provider richiede una chiave KMS di crittografia simmetrica. Non è possibile utilizzare una chiave KMS asimmetrica. Tuttavia, puoi utilizzare una chiave KMS multiregionale, una chiave KMS con materiale chiave importato o una chiave KMS in un archivio chiavi personalizzato. È necessario disporre delle autorizzazioni kms: GenerateDataKey e kms:Decrypt sulla chiave KMS. Pertanto, è necessario utilizzare una chiave gestita dal cliente, non una chiave KMS AWS gestita o AWS di proprietà.

Il client di crittografia DynamoDB per Python determina la regione per la chiamata AWS KMS dalla regione nel valore del parametro ID chiave, se ne include una. Altrimenti, utilizza la regione nel AWS KMS client, se ne viene specificata una, o la regione configurata inAWS SDK for Python (Boto3). Per informazioni sulla selezione della regione in Python, consulta Configurazione nel riferimento API AWS SDK for Python (Boto3).

Il client di crittografia DynamoDB per Java determina la regione per le chiamate AWS KMS dalla regione del AWS KMS client, se il client specificato include una regione. Altrimenti, utilizza la regione configurata inAWS SDK for Java. Per informazioni sulla selezione della regione inAWS SDK for Java, consulta la Regione AWSselezione nella Guida per gli AWS SDK for Java sviluppatori.

Java

```
// Replace the example key ARN and Region with valid values for your application
final String keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
final String region = 'us-west-2'

final AWSKMS kms = AWSKMSClientBuilder.standard().withRegion(region).build();
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, keyArn);
```

Python

L'esempio seguente utilizza la chiave ARN per specificare ilAWS KMS key. Se l'identificatore di chiave non include unRegione AWS, il client di crittografia DynamoDB ottiene la regione dalla sessione Botocore configurata, se presente, o dai valori predefiniti di Boto.

```
# Replace the example key ID with a valid value
kms_key = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key)
```

Se utilizzi tabelle globali Amazon DynamoDB, ti consigliamo di crittografare i dati con una chiave multiregionale. AWS KMS Le chiavi multiregionali sono diverse Regioni AWS e possono essere utilizzate AWS KMS keys in modo intercambiabile perché hanno lo stesso ID chiave e lo stesso materiale chiave. Per i dettagli, consulta Utilizzo di chiavi multiregionali nella Guida per gli AWS Key Management Service sviluppatori.

> **Note**
>
> Se si utilizza la versione 2017.11.29 delle tabelle globali, è necessario impostare le azioni relative agli attributi in modo che i campi di replica riservati non siano crittografati o firmati. Per informazioni dettagliate, consultare Problemi con le tabelle globali delle versioni precedenti.

Per utilizzare una chiave multiregionale con il client di crittografia DynamoDB, crea una chiave multiregionale e replicala nelle regioni in cui viene eseguita l'applicazione. Quindi configura il Direct KMS Provider per utilizzare la chiave multiregionale nella regione in cui chiama il DynamoDB Encryption Client. AWS KMS

L'esempio seguente configura il client di crittografia DynamoDB per crittografare i dati nella regione Stati Uniti orientali (Virginia settentrionale) (us-east-1) e decrittografarli nella regione Stati Uniti occidentali (Oregon) (us-west-2) utilizzando una chiave multiregionale.

Java

In questo esempio, il client di crittografia DynamoDB ottiene la regione per le chiamate AWS KMS dalla regione del AWS KMS client. Il `keyArn` valore identifica una chiave multiregionale nella stessa regione.

```
// Encrypt in us-east-1

// Replace the example key ARN and Region with valid values for your application
final String usEastKey = 'arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab'
final String region = 'us-east-1'

final AWSKMS kms = AWSKMSClientBuilder.standard().withRegion(region).build();
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, usEastKey);
```

```
// Decrypt in us-west-2
```

```
// Replace the example key ARN and Region with valid values for your application
final String usWestKey = 'arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab'
final String region = 'us-west-2'

final AWSKMS kms = AWSKMSClientBuilder.standard().withRegion(region).build();
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, usWestKey);
```

Python

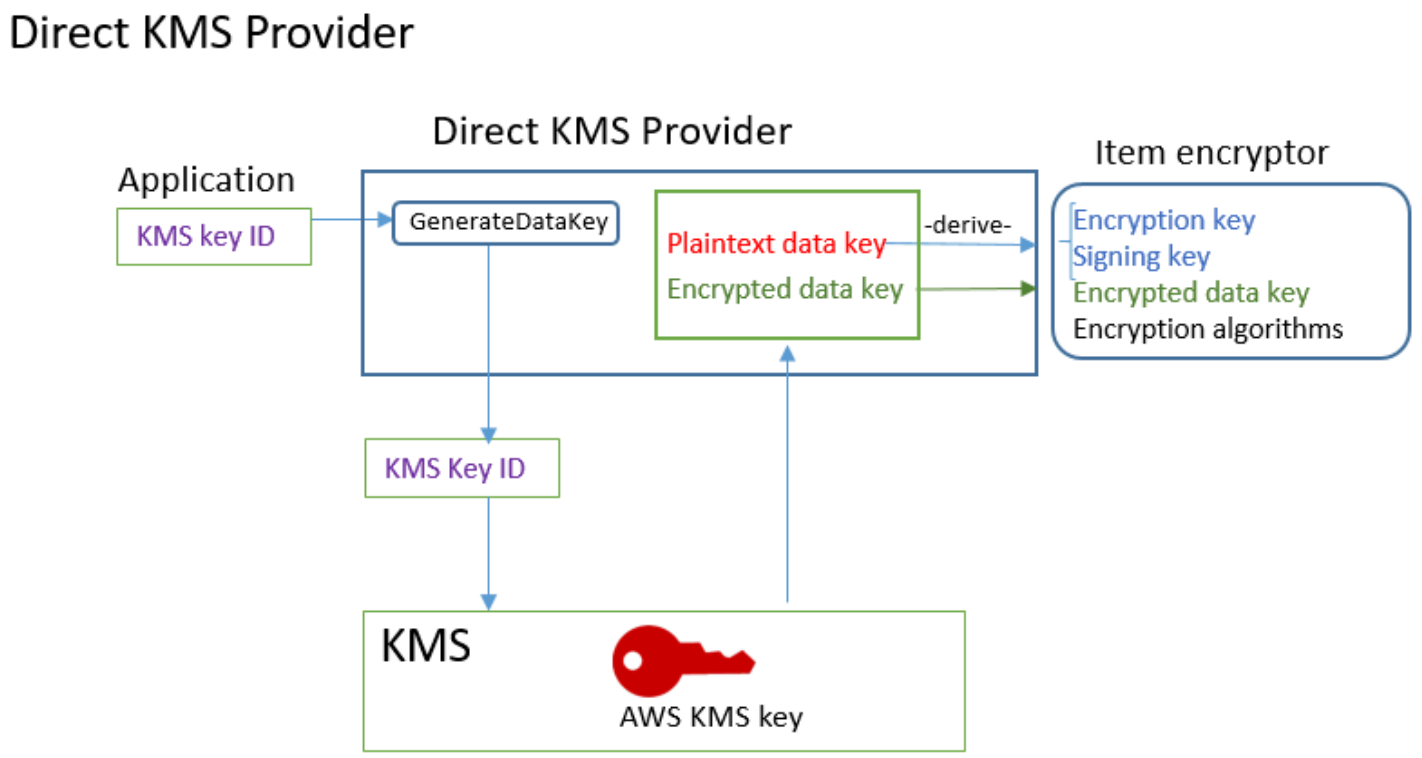
In questo esempio, il client di crittografia DynamoDB ottiene la regione per le chiamate AWS KMS dalla regione nella chiave ARN.

```
# 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)
```

Come funziona

Direct KMS Provider restituisce chiavi di crittografia e firma protette da un codice AWS KMS key specificato dall'utente, come illustrato nel diagramma seguente.



- Per generare materiali di crittografia, Direct KMS Provider richiede di AWS KMS generare una chiave dati univoca per ogni elemento utilizzando una AWS KMS key chiave specificata dall'utente. Deriva le chiavi di crittografia e di firma dell'item dalla copia di testo non crittografato della chiave di dati e le restituisce, insieme alla chiave di dati crittografata, archiviata nell'attributo di descrizione del materiale dell'item.

  Il componente di crittografia dell'item utilizza le chiavi di crittografia e di firma e le rimuove dalla memoria il prima possibile. Nell'item crittografato viene salvata soltanto la copia crittografata della chiave di dati da cui queste chiavi sono state derivate.

- Per generare materiali di decrittografia, il Provider KMS diretto chiede a AWS KMS di decrittografare la chiave di dati crittografata. Quindi, deriva le chiavi di verifica e di firma dalla chiave di dati con testo non crittografato e le restituisce al componente di crittografia dell'item.

  Il componente di crittografia dell'item effettua la verifica dell'item e, in assenza di errori, decrittografa i valori crittografati. Quindi, rimuove le chiavi dalla memoria il prima possibile.

Ottenere materiali di crittografia

Questa sezione descrive nei dettagli gli input, gli output e l'elaborazione del provider KMS diretto al momento della ricezione di una richiesta di materiali di crittografia dal componente di crittografia dell'item.

Input (dall'applicazione)

- L'ID chiave di unAWS KMS key.

Input (dal componente di crittografia dell'item)

- Contesto di crittografia DynamoDB

Output (sul componente di crittografia dell'item)

- Chiave di crittografia (testo non crittografato)
- Chiave di firma
- Nella descrizione dei materiali effettivi: questi valori vengono salvati nell'attributo di descrizione del materiale aggiunto all'item dal client.
  - amzn-ddb-env-key: chiave dati codificata in Base64 crittografata da AWS KMS key
  - amzn-ddb-env-alg: algoritmo di crittografia, di default AES/256
  - amzn-ddb-sig-alg: algoritmo di firma, per impostazione predefinita, hmacSHA256/256
  - amzn-ddb-wrap-alg: km

Elaborazione

1. Il Direct KMS Provider invia AWS KMS una richiesta per utilizzare quanto specificato AWS KMS key per generare una chiave dati univoca per l'articolo. L'operazione restituisce una chiave di testo non crittografato e una copia che viene crittografata con la AWS KMS key. Quest'ultima è nota come materiale di chiave iniziale.

   La richiesta include i valori seguenti sotto forma di testo non crittografato nel contesto di crittografia di AWS KMS. Questi valori non segreti sono crittograficamente legati all'oggetto crittografato; pertanto, per effettuare la decrittografia è necessario lo stesso contesto di crittografia. Puoi utilizzare questi valori per identificare la chiamata a AWS KMS in AWS CloudTrail logs.

   - amzn-ddb-env-alg— Algoritmo di crittografia, di default AES/256

- amzn-ddb-sig-alg— Algoritmo di firma, di default hmacSHA256/256
- (Facoltativo) aws-kms-table — *nome della tabella*
- (Facoltativo) *nome della chiave di partizione*: *valore della chiave di partizione* (i valori binari sono codificati in Base64)
- (Facoltativo) *nome chiave di ordinamento*: *valore della chiave di ordinamento* (i valori binari sono codificati in Base64)

Il provider Direct KMS ottiene i valori per il contesto di AWS KMS crittografia dal contesto di crittografia DynamoDB per l'elemento. Se il contesto di crittografia DynamoDB non include un valore, ad esempio il nome della tabella, quella coppia nome-valore viene omessa dal contesto di crittografia. AWS KMS

2. Il Provider KMS diretto deriva la chiave di crittografia e la chiave di firma simmetriche dalla chiave di dati. Per impostazione predefinita, utilizza il Secure Hash Algorithm (SHA) 256 e la funzione di derivazione della chiave RFC5869 basata su HMAC per derivare una chiave di crittografia simmetrica AES a 256 bit e una chiave di firma HMAC-SHA-256 a 256 bit.
3. Il Provider KMS diretto restituisce l'output al componente di crittografia dell'item.
4. Il componente di crittografia dell'item utilizza la chiave di crittografia per crittografare gli attributi specificati e la chiave di firma per firmarli, tramite gli algoritmi specificati nella descrizione dei materiali effettivi. Rimuove le chiavi di testo non crittografato dalla memoria il prima possibile.

Ottenere materiali di decrittografia

Questa sezione descrive nei dettagli gli input, gli output e l'elaborazione del provider KMS diretto al momento della ricezione di una richiesta di materiali di decrittografia dal componente di crittografia dell'item.

Input (dall'applicazione)

- L'ID chiave di unAWS KMS key.

  Il valore dell'ID chiave può essere l'ID chiave, l'ARN della chiave, il nome dell'alias o l'alias ARN di. AWS KMS key Tutti i valori che non sono inclusi nell'ID chiave, ad esempio la regione, devono essere disponibili nel profilo AWS indicato. La chiave ARN fornisce tutti i valori AWS KMS necessari.

Input (dal componente di crittografia dell'item)

- Una copia del contesto di crittografia DynamoDB che contiene il contenuto dell'attributo di descrizione del materiale.

Output (sul componente di crittografia dell'item)

- Chiave di crittografia (testo non crittografato)
- Chiave di firma

Elaborazione

1. Il provider Direct KMS ottiene la chiave di dati crittografata dall'attributo di descrizione del materiale nell'elemento crittografato.
2. Chiede a AWS KMS di utilizzare la AWS KMS key specificata per decrittografare la chiave di dati crittografata. L'operazione restituisce una chiave di testo non crittografato.

   Questa richiesta deve utilizzare lo stesso contesto di crittografia di AWS KMS utilizzato per generare e crittografare la chiave di dati.

   - aws-kms-table— *`nome della tabella`*
   - *`nome della chiave di partizione`* — *`valore della chiave di partizione`* (i valori binari sono codificati in Base64)
   - (Facoltativo) *`nome chiave di ordinamento`*: *`valore della chiave di ordinamento`* (i valori binari sono codificati in Base64)
   - amzn-ddb-env-alg— Algoritmo di crittografia, di default AES/256
   - amzn-ddb-sig-alg— Algoritmo di firma, di default hmacSHA256/256
3. Il provider KMS diretto utilizza il Secure Hash Algorithm (SHA) 256 e la funzione di derivazione della chiave RFC5869 basata su HMAC per derivare dalla chiave di dati una chiave di crittografia simmetrica AES a 256 bit e una chiave di firma HMAC-SHA-256 a 256 bit.
4. Il Provider KMS diretto restituisce l'output al componente di crittografia dell'item.
5. Il componente di crittografia dell'item utilizza la chiave di firma per verificare l'item. Se l'operazione riesce, utilizza la chiave di crittografia simmetrica per decrittografare i valori di attributo crittografati. Queste operazioni utilizzano gli algoritmi di crittografia e di firma specificati nella descrizione dei materiali effettivi. Il componente di crittografia dell'item rimuove le chiavi di testo non crittografato dalla memoria il prima possibile.

Provider di materiali sottoposti a wrapping

> **Note**
>
> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. L'argomento seguente fornisce informazioni sulle versioni 1. x —2. x del client di crittografia DynamoDB per Java e versioni 1. x —3. x del client di crittografia DynamoDB per Python. Per ulteriori informazioni, consulta AWSDatabase Encryption SDK per il supporto delle versioni di DynamoDB.

Il Wrapped Materials Provider (Wrapped CMP) ti consente di utilizzare chiavi di wrapping e firma da qualsiasi fonte con il client di crittografia DynamoDB. La Wrapped CMP non dipende da alcun AWS servizio. Tuttavia, devi generare e gestire le chiavi di wrapping e firma all'esterno del client, nonché fornire le chiavi corrette per verificare e decrittografare l'item.

Il CMP di sottoposti a wrapping genera una chiave di crittografia item univoca per ciascun item. Esegue il wrapping della chiave di crittografia dell'item con la chiave di wrapping che hai fornito e salva la chiave di crittografia dell'item sottoposta a wrapping nell'attributo di descrizione del materiale dell'item. Poiché sei tu a fornire le chiavi di wrapping e di firma, puoi capire come sono generate le chiavi di wrapping e firma e se sono univoche per ciascun item o riutilizzate.

Il CMP di sottoposti a wrapping è un'implementazione sicura e un'ottima scelta per le applicazioni in grado di gestire i materiali crittografici.

Wrapped CMP è uno dei numerosi fornitori di materiali crittografici (CMP) supportati dal client di crittografia DynamoDB. Per informazioni sugli altri CMP, consulta Fornitore di materiali crittografici.

Per il codice di esempio, consulta:

- Java: AsymmetricEncryptedItem
- Pitone: wrapped-rsa-encrypted-table, wrapped-symmetric-encrypted-table

Argomenti

Come utilizzarlo

Per creare un CMP di sottoposti a wrapping, specifica una chiave di wrapping (necessaria per la crittografia), una chiave di annullamento del wrapping (necessaria per la decrittografia) e una chiave di firma. Le chiavi devono essere fornite al momento di crittografare e decrittografare gli item.

Le chiavi di wrapping, annullamento del wrapping e firma possono essere chiavi simmetriche o coppie di chiavi asimmetriche.

Java

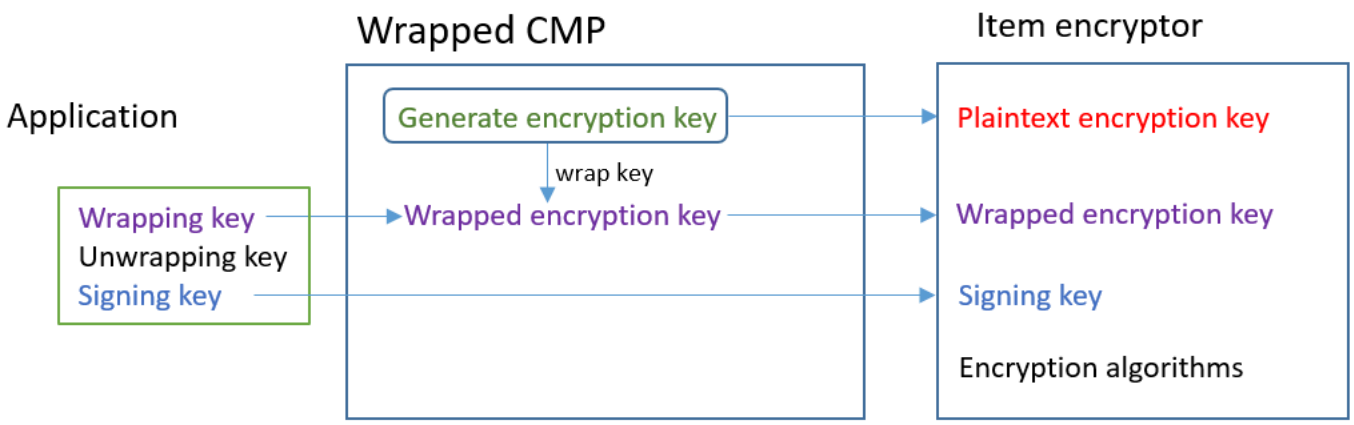
```
// This example uses asymmetric wrapping and signing key pairs
final KeyPair wrappingKeys = ...
final KeyPair signingKeys = ...

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
)
```

Come funziona

Il CMP di sottoposti a wrapping genera una nuova chiave di crittografia item per ciascun item. Utilizza le chiavi di wrapping, annullamento del wrapping e firma da te fornite, come mostrato nel diagramma mostrato di seguito.



Ottenere materiali di crittografia

In questa sezione vengono descritti nei dettagli gli input, gli output e l'elaborazione eseguita dal provider di materiali sottoposti a wrapping (CMP di sottoposti a wrapping) al momento della ricezione di una richiesta di materiali di crittografia.

Input (dall'applicazione)

- Chiave di wrapping: una chiave AES simmetrica o una chiave pubblica RSA. Necessaria se alcuni valori degli attributi sono crittografati. In caso contrario, è facoltativa e viene ignorata.
- Chiave di annullamento del wrapping: facoltativa e ignorata.
- Chiave di firma

Input (dal componente di crittografia dell'item)

- Contesto di crittografia DynamoDB

Output (sul componente di crittografia dell'item)

- Chiave di crittografia dell'item di testo normale
- Chiave di firma (invariata)
- Descrizione dei materiali effettivi: questi valori vengono salvati nell'attributo di descrizione del materiale che il client aggiunge all'item.
  - `amzn-ddb-env-key`: chiave di crittografia item sottoposta a wrapping Base64-encoded
  - `amzn-ddb-env-alg`: algoritmo di crittografia utilizzato per crittografare l'item. Il valore predefinito è AES-256-CBC.

- `amzn-ddb-wrap-alg`: l'algoritmo di wrapping che il CMP di sottoposti a wrapping ha utilizzato per eseguire il wrapping della chiave di crittografia item. Se la chiave di wrapping è una chiave AES, viene sottoposta a wrapping utilizzando `AES-Keywrap` senza riempimenti come indicato in RFC 3394. Se la chiave di wrapping è una chiave RSA, viene crittografata utilizzando RSA OAEP con l'aggiunta di MGF1.

Elaborazione

La crittografia di un item richiede una chiave di wrapping e una chiave di firma. La chiave di annullamento del wrapping è facoltativa e viene ignorata.

1. Il CMP di sottoposti a wrapping genera una chiave di crittografia item simmetrica univoca per l'item della tabella.
2. Utilizza la chiave di wrapping da te specificata per eseguire il wrapping della chiave di crittografia item. Quindi, la rimuove dalla memoria il prima possibile.
3. Restituisce la chiave di crittografia dell'item in testo normale, la chiave di firma da te fornita e una descrizione dei materiali effettivi che include la chiave di crittografia dell'item sottoposta a wrapping e gli algoritmi di crittografia e wrapping.
4. Il componente di crittografia dell'item utilizza la chiave di crittografia testo normale per crittografare l'item. Utilizza la chiave di firma da te fornita per firmare l'item. Quindi, rimuove le chiavi di testo normale dalla memoria il prima possibile. Copia i campi della descrizione dei materiali effettivi, inclusa la chiave di crittografia sottoposta a wrapping (`amzn-ddb-env-key`) nell'attributo di descrizione del materiale dell'item.

Ottenere materiali di decrittografia

In questa sezione vengono descritti nei dettagli gli input, gli output e l'elaborazione eseguita dal provider di materiali sottoposti a wrapping (CMP di sottoposti a wrapping) al momento della ricezione di una richiesta di materiali di decrittografia.

Input (dall'applicazione)

- Chiave wrapping: facoltativa e ignorata.
- Chiave di annullamento del wrapping: la stessa chiave AES simmetrica o la chiave privata RSA corrispondente alla chiave pubblica RSA utilizzata per la crittografia. Necessaria se alcuni valori degli attributi sono crittografati. In caso contrario, è facoltativa e viene ignorata.
- Chiave di firma

Input (dal componente di crittografia dell'item)

- Una copia del contesto di crittografia DynamoDB che contiene il contenuto dell'attributo di descrizione del materiale.

Output (sul componente di crittografia dell'item)

- Chiave di crittografia dell'item di testo normale
- Chiave di firma (invariata)

Elaborazione

La decrittografia di un item richiede una chiave di annullamento del wrapping e una chiave di firma. La chiave di wrapping è facoltativa e viene ignorata.

1. Il CMP di sottoposti a wrapping ottiene la chiave di crittografia item sottoposta a wrapping dall'attributo di descrizione del materiale dell'item.
2. Utilizza la chiave e l'algoritmo di annullamento del wrapping per annullare il wrapping della chiave di crittografia item.
3. Restituisce la chiave di crittografia item di testo normale, la chiave di firma e gli algoritmi di crittografia e firma al componente di crittografia dell'item.
4. Il componente di crittografia dell'item utilizza la chiave di firma per verificare l'item. Se la verifica riesce, utilizza la chiave di crittografia item per decrittografare l'item. Quindi, rimuove le chiavi di testo normale dalla memoria il prima possibile.

Provider più recente

> **Note**
>
> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. L'argomento seguente fornisce informazioni sulle versioni 1. x —2. x del client di crittografia DynamoDB per Java e versioni 1. x —3. x del client di crittografia DynamoDB per Python. Per ulteriori informazioni, consulta AWSDatabase Encryption SDK per il supporto delle versioni di DynamoDB.

Il provider più recente è un provider di materiali crittografici (CMP) progettato per funzionare con gli archivi del provider. Ottiene i CMP dall'archivio del provider e recupera i materiali crittografici che restituisce dai CMP. In genere utilizza ciascun CMP per soddisfare più richieste di materiali crittografici. Ma puoi utilizzare le funzioni del suo archivio provider per controllare in quale misura i materiali vengono riutilizzati, stabilire la frequenza di rotazione del CMP e persino cambiare il tipo di CMP utilizzato senza cambiare il provider più recente.

> **Note**
>
> Il codice associato al `MostRecentProvider` simbolo del provider più recente potrebbe archiviare materiali crittografici in memoria per tutta la durata del processo. Potrebbe consentire a un chiamante di utilizzare chiavi che non è più autorizzato a utilizzare. Il `MostRecentProvider` simbolo è obsoleto nelle versioni precedenti supportate del client di crittografia DynamoDB e rimosso dalla versione 2.0.0. È sostituito dal `CachingMostRecentProvider` simbolo. Per informazioni dettagliate, consultare Aggiornamenti al provider più recente.

Il provider più recente è una buona scelta per le applicazioni che devono ridurre al minimo le chiamate all'archivio provider e all'origine crittografica e per le applicazioni che possono riutilizzare alcuni materiali crittografici senza violare i requisiti di sicurezza. Ad esempio, ti consente di proteggere i tuoi materiali crittografici con un AWS KMS keyin AWS Key Management Service(AWS KMS) senza chiamare AWS KMS ogni volta che crittografi o decrittografi un elemento.

L'archivio provider che hai scelto determina il tipo di CMP utilizzato dal provider più recente e la frequenza con cui questi ottiene un nuovo CMP. Puoi utilizzare qualsiasi archivio provider compatibile con il provider più recente, inclusi quelli personalizzati che hai progettato.

Il client di crittografia DynamoDB include un programma MetaStoreche crea e restituisce Wrapped Materials Provider (Wrapped CMP). MetaStoreSalva più versioni dei Wrapped CMP che genera in una tabella DynamoDB interna e le protegge con la crittografia lato client tramite un'istanza interna del DynamoDB Encryption Client.

Puoi configurarlo MetaStore per utilizzare qualsiasi tipo di CMP interno per proteggere i materiali nella tabella, incluso un Direct KMS Provider che genera materiali crittografici protetti dall'utenteAWS KMS key, una CMP Wrapped che utilizza le chiavi di wrapping e firma fornite o una CMP personalizzata compatibile progettata da te.

Per il codice di esempio, consulta:

- Java: MostRecentEncryptedItem
- Python: most_recent_provider_encrypted_table

Argomenti

- Come utilizzarlo
- Come funziona
- Aggiornamenti al provider più recente

Come utilizzarlo

Per creare un provider più recente devi creare e configurare un archivio provider e quindi creare un provider più recente che lo utilizzi.

Gli esempi seguenti mostrano come creare un provider più recente che utilizza MetaStore e protegge le versioni nella sua tabella DynamoDB interna con materiali crittografici di un Direct KMS Provider. In questi esempi viene utilizzato il `CachingMostRecentProvider`simbolo.

Ogni provider più recente ha un nome che identifica i suoi CMP nella MetaStore tabella, un'impostazione time-to-live(TTL) e un'impostazione della dimensione della cache che determina quante voci può contenere la cache. Questi esempi impostano la dimensione della cache su 1000 voci e un TTL di 60 secondi.

Java

```
// Set the name for MetaStore's internal table
final String keyTableName = 'metaStoreTable'

// Set the Region and AWS KMS key
final String region = 'us-west-2'
final String keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

// Set the TTL and cache size
final long ttlInMillis = 60000;
final long cacheSize = 1000;

// Name that identifies the MetaStore's CMPs in the provider store
final String materialName = 'testMRP'
```

```
// Create an internal DynamoDB client for the MetaStore
final AmazonDynamoDB ddb =
 AmazonDynamoDBClientBuilder.standard().withRegion(region).build();

// Create an internal Direct KMS Provider for the MetaStore
final AWSKMS kms = AWSKMSClientBuilder.standard().withRegion(region).build();
final DirectKmsMaterialProvider kmsProv = new DirectKmsMaterialProvider(kms,
 keyArn);

// Create an item encryptor for the MetaStore,
// including the Direct KMS Provider
final DynamoDBEncryptor keyEncryptor = DynamoDBEncryptor.getInstance(kmsProv);

// Create the MetaStore
final MetaStore metaStore = new MetaStore(ddb, keyTableName, keyEncryptor);

//Create the Most Recent Provider
final CachingMostRecentProvider cmp = new CachingMostRecentProvider(metaStore,
 materialName, ttlInMillis, cacheSize);
```

Python

```
# Designate an AWS KMS key
kms_key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

# Set the name for MetaStore's internal table
meta_table_name = 'metaStoreTable'

# Name that identifies the MetaStore's CMPs in the provider store
material_name = 'testMRP'

# Create an internal DynamoDB table resource for the MetaStore
meta_table = boto3.resource('dynamodb').Table(meta_table_name)

# Create an internal Direct KMS Provider for the MetaStore
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key_id)

# Create the MetaStore with the Direct KMS Provider
meta_store = MetaStore(
    table=meta_table,
    materials_provider=kms_cmp
)
```

```
# Create a Most Recent Provider using the MetaStore
#    Sets the TTL (in seconds) and cache size (# entries)
most_recent_cmp = MostRecentProvider(
    provider_store=meta_store,
    material_name=material_name,
    version_ttl=60.0,
    cache_size=1000
)
```

Come funziona

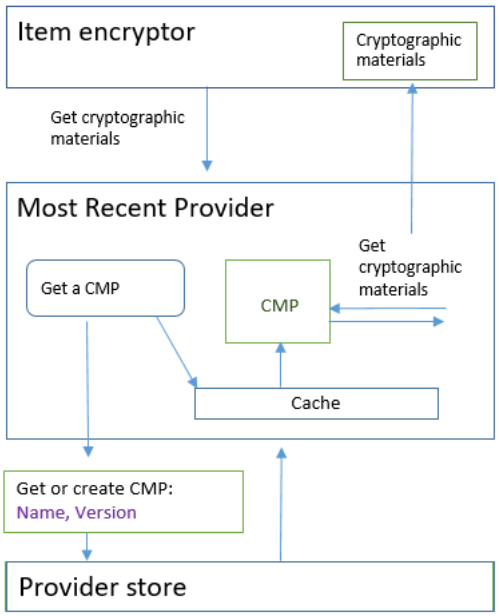
Il provider più recente riceve i CMP da un archivio provider. Quindi utilizza il CMP per generare i materiali crittografici che restituisce al componente di crittografia dell'item.

Informazioni sul provider più recente

Il provider più recente ottiene un provider di materiali crittografici (CMP) da un archivio provider. Utilizza quindi il CMP per generare i materiali crittografici che restituisce. Ciascun provider più recente è associato a un archivio provider, ma un archivio provider può fornire CMP a più provider su più host.

Il provider più recente può utilizzare qualsiasi CMP compatibile di qualsiasi archivio provider. Richiede materiali di crittografia o decrittografia dal CMP e restituisce l'output al criptatore dell'elemento. Non esegue alcuna operazione crittografica.

Per richiedere un CMP al suo archivio provider, il provider più recente fornisce il nome del materiale e la versione di un CMP esistente che desidera utilizzare. Per i materiali di crittografia, il provider più recente richiede sempre la versione massima ("più recente"). Per i materiali di decrittografia, richiede la versione del CMP che è stata utilizzata per creare i materiali di crittografia, come mostrato nel diagramma seguente.



Il provider più recente salva le versioni dei CMP che l'archivio provider restituisce nella cache degli item utilizzati meno di recente (LRU) in memoria. La cache consente al provider più recente di ottenere i CMP di cui ha bisogno senza chiamare l'archivio provider per ogni item. Puoi cancellare la cache on-demand.

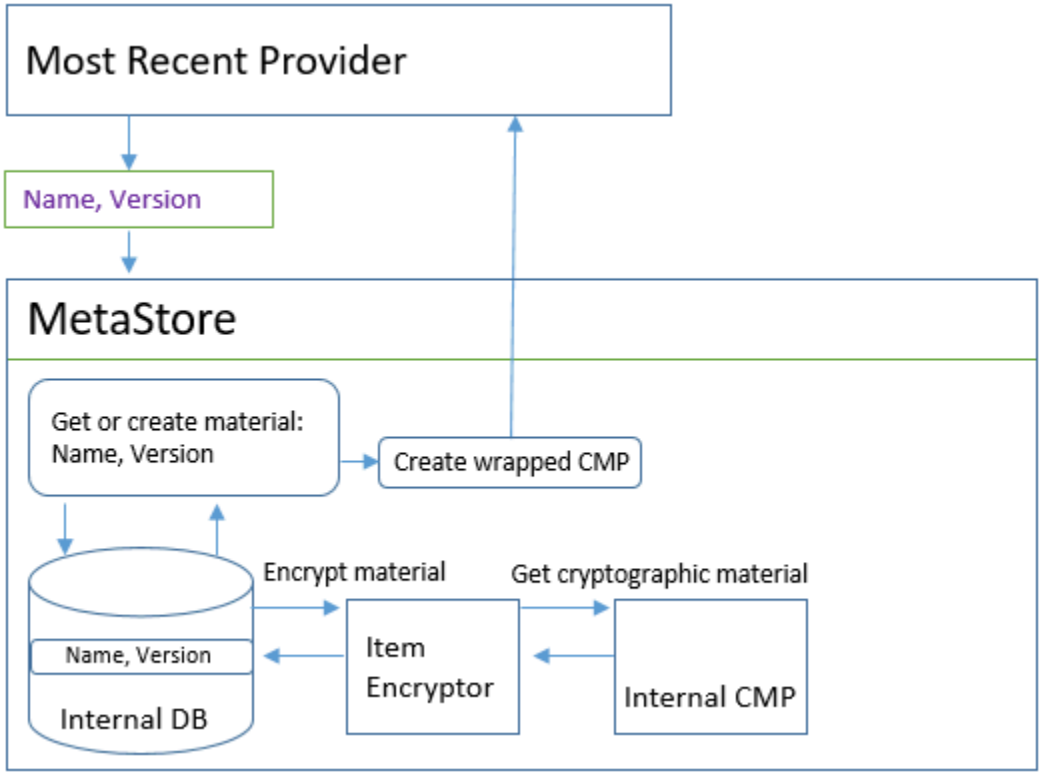
Il provider più recente utilizza un time-to-livevalore configurabile che è possibile regolare in base alle caratteristiche dell'applicazione.

Informazioni su MetaStore

Puoi utilizzare un provider più recente con qualsiasi archivio provider, anche un archivio provider personalizzato compatibile. Il client di crittografia DynamoDB include un'MetaStoreimplementazione sicura che puoi configurare e personalizzare.

A MetaStoreè un provider store che crea e restituisce i Wrapped CMP configurati con la chiave di wrapping, la chiave di unwrapping e la chiave di firma richieste da Wrapped CMP. A MetaStore è un'opzione sicura per un provider più recente perché i Wrapped CMP generano sempre chiavi di crittografia degli articoli univoche per ogni elemento. Vengono riutilizzate solo la chiave di wrapping che protegge la chiave di crittografia degli item e le chiavi di firma.

Il diagramma seguente mostra i componenti MetaStore e come interagisce con il provider più recente.



MetaStoreGenera i Wrapped CMP e li archivia (in forma crittografata) in una tabella DynamoDB interna. La chiave di partizione è il nome del materiale del fornitore più recente; la chiave di ordinamento è il numero di versione. I materiali della tabella sono protetti da un client di crittografia DynamoDB interno, che include un criptatore di elementi e un fornitore interno di materiali crittografici (CMP).

Puoi utilizzare qualsiasi tipo di CMP interna nella tuaMetaStore, incluso un Direct KMS Provider, una CMP Wrapped con materiali crittografici che fornisci o una CMP personalizzata compatibile. Se la CMP interna del tuo MetaStore è un Direct KMS Provider, le tue chiavi di wrapping e firma riutilizzabili sono protette da un AWS KMS keyin (). AWS Key Management ServiceAWS KMS Le MetaStore chiamate AWS KMS ogni volta che aggiunge una nuova versione CMP alla tabella interna o ottiene una versione CMP dalla tabella interna.

Impostazione di un time-to-live valore

Puoi impostare un valore time-to-live (TTL) per ogni provider più recente che crei. In generale, utilizzate il valore TTL più basso possibile per la vostra applicazione.

L'uso del valore TTL viene modificato nel `CachingMostRecentProvider` simbolo del provider più recente.

> **Note**
>
> Il `MostRecentProvider` simbolo del provider più recente è obsoleto nelle versioni precedenti supportate del client di crittografia DynamoDB e rimosso dalla versione 2.0.0. È sostituito dal `CachingMostRecentProvider` simbolo. Ti consigliamo di aggiornare il codice il prima possibile. Per informazioni dettagliate, consultare Aggiornamenti al provider più recente.

`CachingMostRecentProvider`

`CachingMostRecentProvider`Utilizza il valore TTL in due modi diversi.

- Il TTL determina la frequenza con cui il provider più recente verifica la presenza di una nuova versione della CMP nell'archivio del provider. Se è disponibile una nuova versione, il provider più recente sostituisce il suo CMP e aggiorna i materiali crittografici. Altrimenti, continua a utilizzare il suo attuale materiale CMP e crittografico.
- Il TTL determina per quanto tempo possono essere utilizzati i CMP nella cache. Prima di utilizzare una CMP memorizzata nella cache per la crittografia, il provider più recente valuta il tempo trascorso nella cache. Se il tempo di cache CMP supera il TTL, il CMP viene rimosso dalla cache e il provider più recente ottiene una nuova CMP più recente dall'archivio del provider.

`MostRecentProvider`

Nel`MostRecentProvider`, il TTL determina la frequenza con cui il provider più recente verifica la presenza di una nuova versione della CMP nell'archivio del provider. Se è disponibile una nuova versione, il provider più recente sostituisce il suo CMP e aggiorna i materiali crittografici. Altrimenti, continua a utilizzare il suo attuale materiale CMP e crittografico.

Il TTL non determina la frequenza con cui viene creata una nuova versione CMP. Si creano nuove versioni CMP ruotando i materiali crittografici.

Il valore TTL ideale varia in base all'applicazione e ai suoi obiettivi di latenza e disponibilità. Un TTL inferiore migliora il profilo di sicurezza riducendo il tempo di archiviazione dei materiali crittografici in memoria. Inoltre, un TTL inferiore aggiorna le informazioni critiche più frequentemente. Ad esempio, se il tuo CMP interno è un Direct KMS Provider, verifica più frequentemente che il chiamante sia ancora autorizzato a utilizzare un. AWS KMS key

Tuttavia, se il TTL è troppo breve, le frequenti chiamate all'archivio del provider possono aumentare i costi e far sì che il negozio del provider limiti le richieste provenienti dall'applicazione e da altre applicazioni che condividono l'account del servizio. Potresti anche trarre vantaggio dal coordinamento del TTL con la velocità con cui ruotate i materiali crittografici.

Durante i test, varia il TTL e la dimensione della cache in base a diversi carichi di lavoro fino a trovare una configurazione adatta alla tua applicazione e ai tuoi standard di sicurezza e prestazioni.

Rotazione dei materiali crittografici

Quando un Most Recent Provider necessita di materiale crittografato, utilizza sempre la versione più recente della sua CMP di cui è a conoscenza. La frequenza con cui verifica la presenza di una versione più recente è determinata dal valore time-to-live(TTL) impostato durante la configurazione del provider più recente.

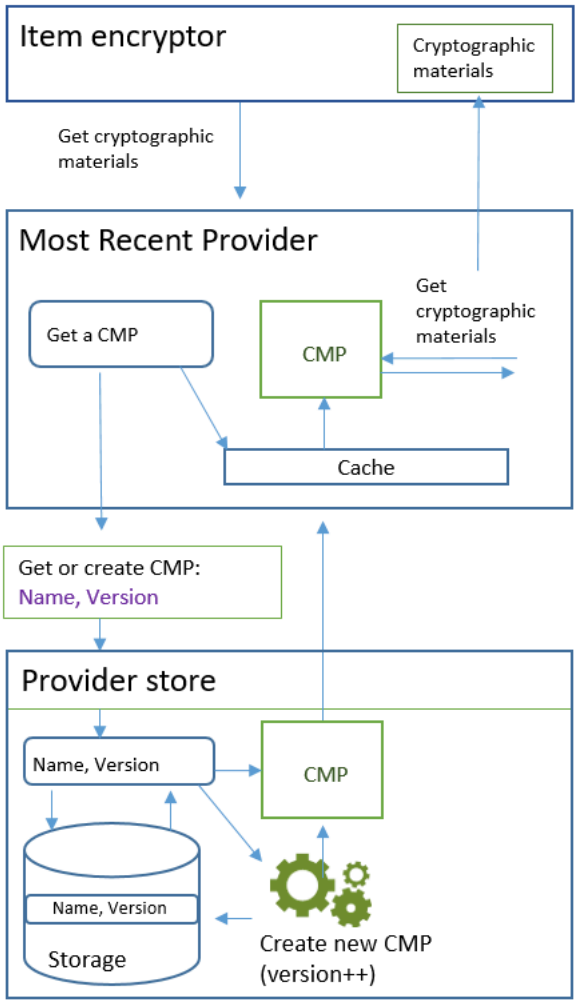
Alla scadenza del TTL, il provider più recente verifica la presenza di una versione più recente della CMP nell'archivio del provider. Se disponibile, il provider più recente lo ottiene e sostituisce il CMP nella sua cache. Utilizza questa CMP e il suo materiale crittografico fino a quando non scopre che il provider store ha una versione più recente.

Per indicare all'archivio provider di creare una nuova versione di un CMP per un provider più recente, richiama l'operazione Create New Provider (Crea nuovo provider) dell'archivio provider con il nome del materiale del provider più recente. L'archivio provider crea un nuovo CMP e salva una copia crittografata nel suo storage interno con un numero di versione superiore: Restituisce anche un CMP, ma puoi ignorarlo. Di conseguenza, la prossima volta che il provider più recente interroga l'archivio del provider per il numero massimo di versioni dei suoi CMP, ottiene il nuovo numero di versione maggiore e lo utilizza nelle successive richieste allo store per verificare se è stata creata una nuova versione della CMP.

Puoi programmare le chiamate Create New Provider (Crea nuovo provider) sulla base del tempo, del numero di item o di attributi elaborati o su qualsiasi altro parametro rilevante per la tua applicazione.

Ottenere materiali di crittografia

Il provider più recente utilizza il seguente processo, mostrato in questo diagramma, per ottenere i materiali di crittografia che restituisce al componente di crittografia dell'item. L'output dipende dal tipo di CMP restituito dall'archivio provider. Il provider più recente può utilizzare qualsiasi provider store compatibile, incluso MetaStore quello incluso nel client di crittografia DynamoDB.



Quando si crea un provider più recente utilizzando il `CachingMostRecentProvider`simbolo, si specifica un archivio provider, un nome per il provider più recente e un valore time-to-live(TTL). È inoltre possibile specificare una dimensione della cache, che determina il numero massimo di materiali crittografici che possono esistere nella cache.

Quando il componente di crittografia dell'item chiede al provider più recente i materiali di crittografia, il provider più recente inizia a cercare nella sua cache la versione più recente del suo CMP.

- Se trova la versione CMP più recente nella sua cache e la CMP non ha superato il valore TTL, il provider più recente utilizza la CMP per generare materiali di crittografia. Quindi restituisce i materiali di crittografia al componente di crittografia dell'item. Questa operazione non richiede una chiamata dell'archivio provider.
- Se la versione più recente della CMP non è nella cache o se si trova nella cache ma ha superato il suo valore TTL, il provider più recente richiede una CMP dall'archivio del provider. La richiesta

include il nome del materiale del provider più recente e il numero della versione massima che conosce.

1. L'archivio provider restituisce un CMP dal suo storage persistente. Se l'archivio del provider è unMetaStore, ottiene una CMP Wrapped crittografata dalla sua tabella DynamoDB interna utilizzando il nome del materiale del provider più recente come chiave di partizione e il numero di versione come chiave di ordinamento. MetaStoreUtilizza il suo criptatore interno per gli elementi e la CMP interna per decrittografare la Wrapped CMP. Quindi restituisce il CMP come testo normale al provider più recente. Se il CMP interno è un provider KMS diretto, questa fase prevede una chiamata a AWS Key Management Service (AWS KMS).
2. Il CMP aggiunge il campo `amzn-ddb-meta-id` alla descrizione dei materiali effettivi. Il suo valore è il nome del materiale e la versione del CMP nella sua tabella interna. L'archivio provider restituisce al provider più recente il CMP come testo normale.
3. Il provider più recente archivia il CMP nella cache.
4. Il provider più recente utilizza il CMP per generare i materiali di crittografia. Quindi restituisce i materiali di crittografia al componente di crittografia dell'item.

Ottenere materiali di decrittografia

Quando il componente di crittografia dell'item chiede al provider più recente i materiali di decrittografia, il provider più recente utilizza il seguente processo per ottenerli e restituirli.

1. Il provider più recente chiede all'archivio provider il numero di versione dei materiali crittografici utilizzati per crittografare l'item. Passa la descrizione dei materiali effettivi dall'attributo di descrizione del materiale dell'item.
2. L'archivio provider riceve il numero di versione del CMP di crittografia dal campo `amzn-ddb-meta-id` nella descrizione dei materiali effettivi e lo restituisce al provider più recente.
3. Il provider più recente ricerca nella sua cache la versione del CMP utilizzata per crittografare e firmare l'item.

- Se rileva che la versione corrispondente della CMP è nella sua cache e la CMP non ha superato il valore time-to-live (TTL), il provider più recente utilizza la CMP per generare materiali di decrittografia. Quindi restituisce i materiali di decrittografia al componente di crittografia dell'item. Questa operazione non richiede una chiamata dell'archivio provider o a qualsiasi altro CMP.
- Se la versione corrispondente della CMP non è nella cache o se quella memorizzata nella cache AWS KMS key ha superato il suo valore TTL, il provider più recente richiede una CMP dall'archivio

del provider. Nella richiesta invia il nome del materiale e il numero di versione del suo CMP di crittografia.

1. L'archivio provider ricerca nello storage persistente il CMP utilizzando il nome del provider più recente come chiave di partizione e il numero di versione come chiave di ordinamento.
   - Se il nome e il numero di versione non sono nello storage persistente, l'archivio provider rileva un'eccezione. Se l'archivio provider è stato utilizzato per generare il CMP, il CMP dovrebbe essere archiviato nel suo storage persistente, a meno che non sia stato volutamente eliminato.
   - Se il CMP con il numero di versione e il nome corrispondenti sono disponibili nello storage persistente dell'archivio provider, quest'ultimo restituisce il CMP specificato al provider più recente.

     Se l'archivio del provider è unMetaStore, ottiene la CMP crittografata dalla relativa tabella DynamoDB. Quindi utilizza i materiali crittografici dal suo CMP interno per decrittografare il CMP crittografato prima di restituire il CMP al provider più recente. Se il CMP interno è un provider KMS diretto, questa fase prevede una chiamata a AWS Key Management Service (AWS KMS).
2. Il provider più recente archivia il CMP nella cache.
3. Il provider più recente utilizza il CMP per generare i materiali di decrittografia. Quindi restituisce i materiali di decrittografia al componente di crittografia dell'item.

Aggiornamenti al provider più recente

Il simbolo del provider più recente viene modificato da `MostRecentProvider` a`CachingMostRecentProvider`.

> **Note**
>
> Il `MostRecentProvider` simbolo, che rappresenta il provider più recente, è obsoleto nella versione 1.15 del client di crittografia DynamoDB per Java e nella versione 1.3 del client di crittografia DynamoDB per Python e rimosso dalle versioni 2.0.0 del client di crittografia DynamoDB in entrambe le implementazioni linguistiche. Anziché, usa il`CachingMostRecentProvider`.

`CachingMostRecentProvider`Implementa le seguenti modifiche:

- Rimuove `CachingMostRecentProvider` periodicamente i materiali crittografici dalla memoria quando il loro tempo in memoria supera il valore configurato time-to-live (TTL).

  `MostRecentProvider`Potrebbero archiviare materiali crittografici in memoria per tutta la durata del processo. Di conseguenza, il fornitore più recente potrebbe non essere a conoscenza delle modifiche alle autorizzazioni. Potrebbe utilizzare chiavi di crittografia dopo la revoca delle autorizzazioni del chiamante per utilizzarle.

  Se non riesci ad eseguire l'aggiornamento a questa nuova versione, puoi ottenere un effetto simile chiamando periodicamente il `clear()` metodo nella cache. Questo metodo svuota manualmente il contenuto della cache e richiede al provider più recente di richiedere una nuova CMP e nuovi materiali crittografici.
- Include `CachingMostRecentProvider` anche un'impostazione della dimensione della cache che offre un maggiore controllo sulla cache.

Per eseguire l'aggiornamento al`CachingMostRecentProvider`, devi modificare il nome del simbolo nel tuo codice. Sotto tutti gli altri aspetti, `CachingMostRecentProvider` è completamente retrocompatibile con. `MostRecentProvider` Non è necessario crittografare nuovamente alcun elemento della tabella.

Tuttavia, `CachingMostRecentProvider` genera più chiamate all'infrastruttura chiave sottostante. Chiama l'archivio del provider almeno una volta in ogni intervallo time-to-live (TTL). È probabile che le applicazioni con numerose CMP attive (a causa della rotazione frequente) o le applicazioni con flotte di grandi dimensioni siano sensibili a questo cambiamento.

Prima di rilasciare il codice aggiornato, testalo attentamente per assicurarti che le chiamate più frequenti non compromettano l'applicazione o causino limitazioni da parte dei servizi da cui dipende il tuo provider, come AWS Key Management Service (AWS KMS) o Amazon DynamoDB. Per mitigare eventuali problemi di prestazioni, regola la dimensione e la time-to-live della cache in `CachingMostRecentProvider` base alle caratteristiche prestazionali osservate. Per le linee guida, consulta Impostazione di un time-to-live valore.

Provider di materiali statici

> **Note**
>
> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. L'argomento seguente fornisce informazioni sulle versioni 1. x —2. x del client di crittografia

DynamoDB per Java e versioni 1. x —3. x del client di crittografia DynamoDB per Python. Per ulteriori informazioni, consulta AWSDatabase Encryption SDK per il supporto delle versioni di DynamoDB.

Static Materials Provider (Static CMP) è un fornitore di materiali crittografici (CMP) molto semplice destinato a test, proof-of-concept dimostrazioni e compatibilità con versioni precedenti.

Per utilizzare il CMP di statici per crittografare un item della tabella, è necessario fornire una chiave di crittografia simmetrica AES e una chiave di firma o una coppia di chiavi. Devi fornire le stesse chiavi per decrittografare l'item crittografato. Il CMP di statici non esegue alcuna operazione di crittografia. Passa invece al componente di crittografia dell'item le chiavi di crittografia da te fornite senza modificarle. Il componente di crittografia dell'item crittografa direttamente gli item con la chiave di crittografia. Quindi, utilizza direttamente la chiave di firma per firmarli.

Poiché il CMP di statici non genera alcun materiale crittografico univoco, tutti gli item della tabella che hai elaborato sono crittografati con la stessa chiave di crittografia e firmati con la stessa chiave di firma. Quando utilizzi la stessa chiave per crittografare i valori degli attributi in numerosi item o utilizzi la stessa chiave o coppia di chiavi per firmare tutti gli item, rischi di superare i limiti crittografici delle chiavi.

> **Note**
>
> Il Provider di statici asimmetrico nella libreria Java non è un provider di statici. Fornisce soltanto costruttori alternativi per il CMP di sottoposti a wrapping. Può essere utilizzato per la produzione senza alcun rischio per la sicurezza, tuttavia dovresti utilizzare direttamente il CMP di sottoposti a wrapping ogni qualvolta sia possibile.

Static CMP è uno dei numerosi fornitori di materiali crittografici (CMP) supportati dal client di crittografia DynamoDB. Per informazioni sugli altri CMP, consulta Fornitore di materiali crittografici.

Per il codice di esempio, consulta:

- Java: SymmetricEncryptedItem

Argomenti

- Come utilizzarlo

- Come funziona

Come utilizzarlo

Per creare un provider di statici, fornisci una chiave di crittografia o una coppia di chiavi e una chiave di firma o una coppia di chiavi. Devi fornire materiali chiave per crittografare e decrittografare gli item della tabella.

Java

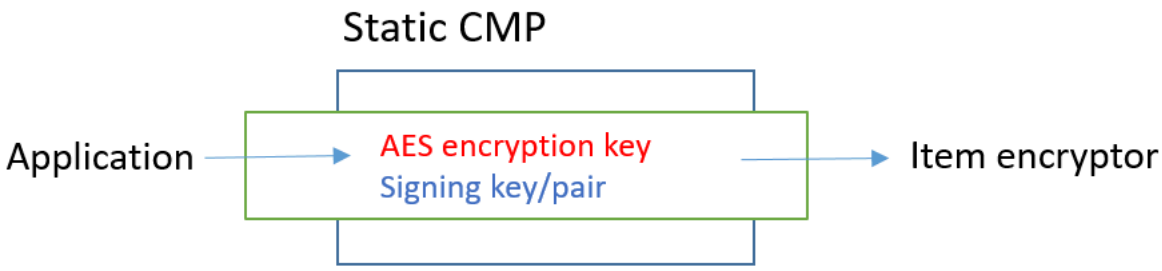
```
// To encrypt
SecretKey cek = ...;        // Encryption key
SecretKey macKey =  ...;    // Signing key
EncryptionMaterialsProvider provider = new SymmetricStaticProvider(cek, macKey);

// 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
)
```

Come funziona

Il provider di statici passa le chiavi di crittografia e firma che hai fornito al componente di crittografia dell'item, dove vengono direttamente utilizzate per crittografare e firmare gli item della tabella. Vengono utilizzate le stesse chiavi per tutti gli item, a meno che tu non fornisca chiavi diverse per ciascun item.



Ottenere materiali di crittografia

In questa sezione vengono descritti nei dettagli gli input, gli output e l'elaborazione eseguita dal provider di materiali statici (CMP di statici) al momento della ricezione di una richiesta di materiali di crittografia.

Input (dall'applicazione)

- Una chiave di crittografia: deve essere una chiave simmetrica, ad esempio una chiave Advanced Encryption Standard (AES).
- Una chiave di firma: può essere una chiave simmetrica o una coppia di chiavi asimmetrica.

Input (dal componente di crittografia dell'item)

- Contesto di crittografia DynamoDB

Output (sul componente di crittografia dell'item)

- La chiave di crittografia passata come input.
- La chiave di firma passata come input.
- Descrizione dei materiali effettivi: l'eventuale descrizione dei materiali richiesti invariata.

Ottenere materiali di decrittografia

In questa sezione vengono descritti nei dettagli gli input, gli output e l'elaborazione eseguita dal provider di materiali statici (CMP di statici) al momento della ricezione di una richiesta di materiali di decrittografia.

Sebbene i metodi per ottenere i materiali di crittografia e ottenere i materiali di decrittografia siano separati, il comportamento è lo stesso.

Input (dall'applicazione)

- Una chiave di crittografia: deve essere una chiave simmetrica, ad esempio una chiave Advanced Encryption Standard (AES).
- Una chiave di firma: può essere una chiave simmetrica o una coppia di chiavi asimmetrica.

Input (dal componente di crittografia dell'item)

- Contesto di crittografia DynamoDB (non utilizzato)

Output (sul componente di crittografia dell'item)

- La chiave di crittografia passata come input.
- La chiave di firma passata come input.

Linguaggi di programmazione disponibili per Amazon DynamoDB Encryption Client

> **Note**
>
> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. L'argomento seguente fornisce informazioni sulle versioni 1. x —2. x del client di crittografia DynamoDB per Java e versioni 1. x —3. x del client di crittografia DynamoDB per Python. Per ulteriori informazioni, consulta AWSDatabase Encryption SDK per il supporto delle versioni di DynamoDB.

Il client di crittografia Amazon DynamoDB è disponibile per i seguenti linguaggi di programmazione. Anche se ogni linguaggio ha delle librerie specifiche, le implementazioni risultanti sono interoperabili. Ad esempio, puoi crittografare (e firmare) un item con il client Java e decrittografarlo con il client Python.

Per maggiori informazioni, consulta l'argomento corrispondente.

Argomenti

- Client di crittografia Amazon DynamoDB per Java
- Client di crittografia DynamoDB per Python

Client di crittografia Amazon DynamoDB per Java

> **Note**
>
> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. L'argomento seguente fornisce informazioni sulle versioni 1. x —2. x del client di crittografia DynamoDB per Java e versioni 1. x —3. x del client di crittografia DynamoDB per Python. Per ulteriori informazioni, consulta AWSDatabase Encryption SDK per il supporto delle versioni di DynamoDB.

Questo argomento spiega come installare e utilizzare il client di crittografia Amazon DynamoDB per Java. Per dettagli sulla programmazione con il client di crittografia DynamoDB, consulta gli esempi Java, gli esempi nel aws-dynamodb-encryption-java repository attivo e il Javadoc per il GitHub client di crittografia DynamoDB.

> **Note**
>
> Versioni 1. x. X del DynamoDB Encryption Client for Java sono in nd-of-supportfase di entrata in vigore a luglio 2022. Effettua l'aggiornamento a una versione più recente il prima possibile.

Argomenti

- Prerequisiti
- Installazione

- Utilizzo del client di crittografia DynamoDB per Java
- Codice di esempio per il client di crittografia DynamoDB per Java

Prerequisiti

Prima di installare il client di crittografia Amazon DynamoDB per Java, assicurati di avere i seguenti prerequisiti.

Un ambiente di sviluppo Java

È necessario Java 8 o versioni successive. Nel sito Web di Oracle, accedi alla pagina Java SE Download, quindi scarica e installa Java SE Development Kit (JDK).

Se utilizzi Oracle JDK, devi scaricare e installare anche Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files.

AWS SDK for Java

Il client di crittografia DynamoDB richiede il modulo DynamoDB AWS SDK for Java anche se l'applicazione non interagisce con DynamoDB. Puoi installare l'intero SDK o solo questo modulo. Se utilizzi Maven, aggiungi `aws-java-sdk-dynamodb` al file `pom.xml`.

Per ulteriori informazioni sull'installazione e la configurazione di AWS SDK for Java, consulta AWS SDK for Java.

Installazione

Puoi installare il client di crittografia Amazon DynamoDB per Java nei seguenti modi.

Manualmente

Per installare il client di crittografia Amazon DynamoDB per Java, clona o scarica il repository. aws-dynamodb-encryption-javaGitHub

Utilizzo di Apache Maven

Il client di crittografia Amazon DynamoDB per Java è disponibile tramite Apache Maven con la seguente definizione di dipendenza.

```
<dependency>
```

```
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-dynamodb-encryption-java</artifactId>
  <version>version-number</version>
</dependency>
```

Dopo aver installato l'SDK, inizia guardando il codice di esempio in questa guida e il client di crittografia DynamoDB Javadoc attivo. GitHub

Utilizzo del client di crittografia DynamoDB per Java

> **Note**
>
> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. L'argomento seguente fornisce informazioni sulle versioni 1. x —2. x del client di crittografia DynamoDB per Java e versioni 1. x —3. x del client di crittografia DynamoDB per Python. Per ulteriori informazioni, consulta AWSDatabase Encryption SDK per il supporto delle versioni di DynamoDB.

Questo argomento spiega alcune delle funzionalità del client di crittografia DynamoDB in Java che potrebbero non essere presenti in altre implementazioni di linguaggi di programmazione.

Per dettagli sulla programmazione con il client di crittografia DynamoDB, consulta gli esempi Java, gli esempi in basso e Javadoc per il client di crittografia DynamoDB. `aws-dynamodb-encryption-java repository` GitHub

Argomenti

- Crittografi di articoli: AttributeEncryptor e DynamoDBEncryptor
- Configurazione del comportamento di salvataggio
- Operazioni di attributo in Java
- Sovrascrivere i nomi delle tabelle

Crittografi di articoli: AttributeEncryptor e DynamoDBEncryptor

Il client di crittografia DynamoDB in Java ha due crittografi: il DynamoDBEncryptor di livello inferiore e il. AttributeEncryptor

`AttributeEncryptor`È una classe di supporto che consente di utilizzare DynamoDBMapper in combinazione AWS SDK for Java con il client di crittografia DynamoDB. `DynamoDB Encryptor` Quando utilizzi `AttributeEncryptor` con `DynamoDBMapper`, crittografa e firma in modo trasparente gli item quando li salvi. Inoltre, verifica e decrittografa in modo trasparente gli item quando li carichi.

Configurazione del comportamento di salvataggio

È possibile utilizzare `AttributeEncryptor` and `DynamoDBMapper` per aggiungere o sostituire gli elementi della tabella con attributi solo firmati o crittografati e firmati. Per queste attività, ti consigliamo di configurare il servizio per utilizzare il comportamento di salvataggio PUT, come illustrato nell'esempio seguente. In caso contrario, potresti non riuscire a decrittografare i dati.

```
DynamoDBMapperConfig mapperConfig =
 DynamoDBMapperConfig.builder().withSaveBehavior(SaveBehavior.PUT).build();
DynamoDBMapper mapper = new DynamoDBMapper(ddb, mapperConfig, new
 AttributeEncryptor(encryptor));
```

Se si utilizza il comportamento di salvataggio predefinito, che aggiorna solo gli attributi modellati nell'elemento della tabella, gli attributi non modellati non vengono inclusi nella firma e non vengono modificati dalle scritture nella tabella. Di conseguenza, alla lettura successiva di tutti gli attributi, la firma non verrà convalidata, poiché non include attributi non modellati.

Puoi inoltre utilizzare il comportamento di salvataggio `CLOBBER`. Questo comportamento di salvataggio è identico al comportamento di salvataggio PUT, ma disabilita il blocco ottimistico e sovrascrive l'item nella tabella.

Per evitare errori di firma, il client di crittografia DynamoDB genera un'eccezione di runtime se `AttributeEncryptor` viene utilizzato un con un file `DynamoDBMapper` che non è configurato con un comportamento di salvataggio di o. `CLOBBER` PUT

Per vedere questo codice usato in un esempio, vedi Utilizzo di DynamoDBMapper l'esempio AwsKmsEncryptedObject.java nel `aws-dynamodb-encryption-java` repository in. GitHub

Operazioni di attributo in Java

Le operazioni di attributo determinano quali valori attributo sono crittografati e firmati, quali solo firmati e quali ignorati. Il metodo che utilizzi per specificare le operazioni di attributo varia a seconda se utilizzi il metodo `DynamoDBMapper` e il componente `AttributeEncryptor` o il componente DynamoDBEncryptor di livello inferiore.

> **Important**
>
> Dopo aver utilizzato le azioni degli attributi per crittografare gli elementi della tabella, l'aggiunta o la rimozione di attributi dal modello di dati potrebbe causare un errore di convalida della firma che impedisce di decrittografare i dati. Per una spiegazione dettagliata, consulta Modifica del modello di dati.

Operazioni di attributo per il componente DynamoDBMapper

Quando utilizzi `DynamoDBMapper` e `AttributeEncryptor`, devi utilizzare le annotazioni per specificare le operazioni di attributo. Il client di crittografia DynamoDB utilizza le annotazioni standard degli attributi DynamoDB che definiscono il tipo di attributo per determinare come proteggere un attributo. Per impostazione predefinita, tutti gli attributi sono crittografati e firmati, tranne le chiavi primarie, che sono firmate ma non crittografate.

> **Note**
>
> Non crittografate il valore degli attributi con l'VersionAttributeannotazione @DynamoDB, anche se potete (e dovreste) firmarli. In caso contrario, le condizioni che utilizzano questo valore potrebbero avere effetti imprevisti.

```
// Attributes are encrypted and signed
@DynamoDBAttribute(attributeName="Description")

// Partition keys are signed but not encrypted
@DynamoDBHashKey(attributeName="Title")

// Sort keys are signed but not encrypted
@DynamoDBRangeKey(attributeName="Author")
```

Per specificare le eccezioni, utilizza le annotazioni di crittografia definite nel client di crittografia DynamoDB per Java. Se le specifichi a livello di classe, diventano il valore predefinito per la classe.

```
// Sign only
@DoNotEncrypt

// Do nothing; not encrypted or signed
```

```
@DoNotTouch
```

Ad esempio, queste annotazioni firmano ma non crittografano l'attributo `PublicationYear` e non crittografano né firmano il valore attributo `ISBN`.

```
// Sign only (override the default)
@DoNotEncrypt
@DynamoDBAttribute(attributeName="PublicationYear")

// Do nothing (override the default)
@DoNotTouch
@DynamoDBAttribute(attributeName="ISBN")
```

Operazioni di attributo per il componente DynamoDBEncryptor

Per specificare le operazioni di attributo quando utilizzi direttamente il componente DynamoDBEncryptor, crea un oggetto `HashMap` in cui le coppie nome-valore rappresentano i nomi degli attributi e le operazioni specificate.

I valori sono validi per le operazioni di attributo definite nel tipo enumerato `EncryptionFlags`. Puoi utilizzare `ENCRYPT` e `SIGN` insieme o solo `SIGN` o ometterle entrambe. Tuttavia, se si utilizza `ENCRYPT` da solo, il client di crittografia DynamoDB genera un errore. Non puoi crittografare un attributo non firmato.

```
ENCRYPT
SIGN
```

> **Warning**
>
> Non crittografare gli attributi che vengono usati per la chiave primaria. Devono rimanere in testo normale in modo che DynamoDB possa trovare l'elemento senza eseguire una scansione completa della tabella.

Se si specifica una chiave primaria nel contesto di crittografia e quindi si specifica `ENCRYPT` nell'azione dell'attributo per uno degli attributi della chiave primaria, il client di crittografia DynamoDB genera un'eccezione.

Ad esempio, il seguente codice Java crea un file `actions` HashMap che crittografa e firma tutti gli attributi dell'`record`elemento. Le eccezioni sono la chiave di partizione e gli attributi della

chiave di ordinamento, che sono firmati ma non crittografati, e l'attributo `test`, che non è firmato o crittografato.

```
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;
  }
}
```

Successivamente, quando chiami il metodo encryptRecord del componente `DynamoDBEncryptor`, devi specificare la mappa come valore del parametro `attributeFlags`. Ad esempio, questa chiamata a `encryptRecord` utilizza la mappa `actions`.

```
// Encrypt the plaintext record
final Map<String, AttributeValue> encrypted_record = encryptor.encryptRecord(record,
 actions, encryptionContext);
```

Sovrascrivere i nomi delle tabelle

Nel client di crittografia DynamoDB, il nome della tabella DynamoDB è un elemento del contesto di crittografia DynamoDB che viene passato ai metodi di crittografia e decrittografia. Quando si crittografano o si firmano gli elementi della tabella, il contesto di crittografia DynamoDB, incluso il nome della tabella, viene associato crittograficamente al testo cifrato. Se il contesto di crittografia DynamoDB passato al metodo di decrittografia non corrisponde al contesto di crittografia DynamoDB passato al metodo di crittografia, l'operazione di decrittografia ha esito negativo.

Di tanto in tanto, il nome di una tabella cambia, ad esempio quando si esegue il backup di una tabella o si esegue un point-in-timeripristino. Quando decifri o verifichi la firma di questi elementi, devi passare nello stesso contesto di crittografia DynamoDB utilizzato per crittografare e firmare gli elementi, incluso il nome della tabella originale. Il nome della tabella corrente non è necessario.

Quando si utilizza il`DynamoDBEncryptor`, si assembla il contesto di crittografia DynamoDB manualmente. Tuttavia, se si utilizza il`DynamoDBMapper`, `AttributeEncryptor` viene creato automaticamente il contesto di crittografia DynamoDB, incluso il nome della tabella corrente. Per comunicare a `AttributeEncryptor` di creare un contesto di crittografia con un nome di tabella diverso, utilizza `EncryptionContextOverrideOperator`.

Ad esempio, il codice seguente crea istanze del provider di materiali crittografici (CMP) e di `DynamoDBEncryptor`. Quindi chiama il metodo `setEncryptionContextOverrideOperator` di `DynamoDBEncryptor`. Utilizza l'operatore `overrideEncryptionContextTableName`, che sovrascrive il nome di una tabella. Quando è configurato in questo modo, `AttributeEncryptor` crea un contesto di crittografia DynamoDB che include al posto `newTableName` di. `oldTableName` Per un esempio completo, vedere EncryptionContextOverridesWithDynamoDBMapper.java.

```
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, keyArn);
final DynamoDBEncryptor encryptor = DynamoDBEncryptor.getInstance(cmp);

encryptor.setEncryptionContextOverrideOperator(EncryptionContextOperators.overrideEncryptionCon
                oldTableName, newTableName));
```

Quando chiami il metodo di caricamento di `DynamoDBMapper`, che esegue la decrittografia e la verifica dell'item, devi specificare il nome della tabella originale.

```
mapper.load(itemClass, DynamoDBMapperConfig.builder()

 .withTableNameOverride(DynamoDBMapperConfig.TableNameOverride.withTableNameReplacement(oldTabl
                .build());
```

Puoi anche utilizzare l'operatore `overrideEncryptionContextTableNameUsingMap`, che sovrascrive più nomi di tabella.

Gli operatori che sovrascrivono i nomi di tabella vengono in genere utilizzati per la decrittografia dei dati e la verifica delle firme. Tuttavia, puoi utilizzarli per impostare il nome della tabella nel contesto di crittografia DynamoDB su un valore diverso durante la crittografia e la firma.

Non utilizzare operatori che sovrascrivono i nomi di tabella se utilizzi `DynamoDBEncryptor`. Crea invece un contesto di crittografia con il nome della tabella originale e invialo al metodo di decrittografia.

Codice di esempio per il client di crittografia DynamoDB per Java

> **Note**
>
> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. L'argomento seguente fornisce informazioni sulle versioni 1. x —2. x del client di crittografia DynamoDB per Java e versioni 1. x —3. x del client di crittografia DynamoDB per Python. Per ulteriori informazioni, consulta AWSDatabase Encryption SDK per il supporto delle versioni di DynamoDB.

Gli esempi seguenti mostrano come utilizzare il client di crittografia DynamoDB per Java per proteggere gli elementi della tabella DynamoDB nell'applicazione. Puoi trovare altri esempi (e contribuire con i tuoi) nella cartella degli esempi del aws-dynamodb-encryption-javarepository su. GitHub

Argomenti

- Utilizzo di DynamoDBEncryptor
- Utilizzo di DynamoDBMapper

Utilizzo di DynamoDBEncryptor

Questo esempio mostra come utilizzare il componente DynamoDBEncryptor di livello inferiore con il provider KMS diretto. Il Direct KMS Provider genera e protegge i propri materiali crittografici con un valore AWS KMS keyin AWS Key Management Service (AWS KMS) specificato dall'utente.

Puoi utilizzare qualsiasi fornitore di materiali crittografici (CMP) compatibile con`DynamoDBEncryptor`, e puoi utilizzare Direct KMS Provider con e. `DynamoDBMapper` AttributeEncryptor

Guarda l'esempio di codice completo: AwsKmsEncryptedItem.java

Fase 1: creazione del provider KMS diretto

Crea un'istanza del client AWS KMS con la regione specificata. Quindi, utilizza l'istanza client per creare un'istanza del Direct KMS Provider con la tua preferenzaAWS KMS key.

Questo esempio utilizza Amazon Resource Name (ARN) per identificareAWS KMS key, ma puoi utilizzare qualsiasi identificatore di chiave valido.

```
final String keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
final String region = 'us-west-2'

final AWSKMS kms = AWSKMSClientBuilder.standard().withRegion(region).build();
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, keyArn);
```

Fase 2: creazione di un item

Questo esempio definisce un elemento `record` HashMap che rappresenta un elemento della tabella di esempio.

```
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"));
```

Fase 3: creazione di un componente DynamoDBEncryptor

Crea un'istanza del componente `DynamoDBEncryptor` con il provider KMS diretto.

```
final DynamoDBEncryptor encryptor = DynamoDBEncryptor.getInstance(cmp);
```

Fase 4: Creare un contesto di crittografia DynamoDB

Il contesto di crittografia DynamoDB contiene informazioni sulla struttura della tabella e su come viene crittografata e firmata. Se utilizzi il componente `DynamoDBMapper`, il componente `AttributeEncryptor` crea il contesto di crittografia per tuo conto.

```
final String tableName = "testTable";

final EncryptionContext encryptionContext = new EncryptionContext.Builder()
    .withTableName(tableName)
    .withHashKeyName(partitionKeyName)
    .withRangeKeyName(sortKeyName)
    .build();
```

Fase 5: creazione dell'oggetto delle operazioni di attributo.

Le operazioni di attributo determinano quali attributi dell'item sono crittografati e firmati, quali solo firmati e quali non sono né crittografati né firmati.

In Java, per specificare le azioni degli attributi, si crea una serie HashMap di coppie di nome e `EncryptionFlags` valore dell'attributo.

Ad esempio, il codice Java seguente crea un file `actions` HashMap che crittografa e firma tutti gli attributi dell'`record`elemento, ad eccezione degli attributi della chiave di partizione e della chiave di ordinamento, che sono firmati ma non crittografati, e l'`test`attributo, che non è firmato o crittografato.

```
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;
  }
}
```

Fase 6: crittografia e firma dell'item

Per crittografare e firmare l'item della tabella, chiama il metodo `encryptRecord` nell'istanza del componente `DynamoDBEncryptor`. Specifica l'item della tabella (`record`), le operazioni di attributo (`actions`) e il contesto di crittografia (`encryptionContext`).

```
final Map<String, AttributeValue> encrypted_record = encryptor.encryptRecord(record,
 actions, encryptionContext);
```

Fase 7: Inserire l'elemento nella tabella DynamoDB

Infine, inserisci l'elemento crittografato e firmato nella tabella DynamoDB.

```
final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();
ddb.putItem(tableName, encrypted_record);
```

Utilizzo di DynamoDBMapper

L'esempio seguente mostra come utilizzare la classe helper di mappatura DynamoDB con il Direct KMS Provider. Il Direct KMS Provider genera e protegge i propri materiali crittografici con un valore AWS KMS keyin AWS Key Management Service (AWS KMS) specificato dall'utente.

Puoi usare qualunque provider di materiali crittografici (CMP) compatibile insieme al mappatore `DynamoDBMapper` e puoi utilizzare il provider KMS diretto con il componente `DynamoDBEncryptor` di livello inferiore.

Guarda l'esempio di codice completo: AwsKmsEncryptedObject.java

Fase 1: creazione del provider KMS diretto

Crea un'istanza del client AWS KMS con la regione specificata. Quindi, utilizza l'istanza client per creare un'istanza del Direct KMS Provider con la tua preferenzaAWS KMS key.

Questo esempio utilizza Amazon Resource Name (ARN) per identificareAWS KMS key, ma puoi utilizzare qualsiasi identificatore di chiave valido.

```
final String keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
final String region = 'us-west-2'

final AWSKMS kms = AWSKMSClientBuilder.standard().withRegion(region).build();
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, keyArn);
```

Fase 2: creazione del componente di crittografia DynamoDB e del mappatore DynamoDB

Usa il Direct KMS Provider che hai creato nel passaggio precedente per creare un'istanza di DynamoDB Encryptor. È necessario istanziare il DynamoDB Encryptor di livello inferiore per utilizzare DynamoDB Mapper.

Quindi, crea un'istanza del tuo database DynamoDB e una configurazione del mappatore e usali per creare un'istanza di DynamoDB Mapper.

> **Important**
>
> Quando utilizzi `DynamoDBMapper` per aggiungere o modificare item firmati (oppure crittografati e firmati), configuralo per utilizzare un comportamento di salvataggio, ad esempio PUT, che includa tutti gli attributi, come mostrato nel seguente esempio. In caso contrario, potresti non riuscire a decrittografare i dati.

```
final DynamoDBEncryptor encryptor = DynamoDBEncryptor.getInstance(cmp)
final AmazonDynamoDB ddb =
 AmazonDynamoDBClientBuilder.standard().withRegion(region).build();

DynamoDBMapperConfig mapperConfig =
 DynamoDBMapperConfig.builder().withSaveBehavior(SaveBehavior.PUT).build();
DynamoDBMapper mapper = new DynamoDBMapper(ddb, mapperConfig, new
 AttributeEncryptor(encryptor));
```

Fase 3: Definizione della tabella DynamoDB

Quindi, definisci la tua tabella DynamoDB. Per specificare le operazioni di attributo, utilizza le annotazioni. Questo esempio crea una tabella DynamoDB e una `DataPoJo` classe che rappresenta gli elementi della tabella. `ExampleTable`

In questo esempio, gli attributi della chiave primaria saranno firmati ma non crittografati. Ciò vale per l'attributo `partition_attribute`, che viene annotato con `@DynamoDBHashKey`, e per l'attributo `sort_attribute`, che viene annotato con `@DynamoDBRangeKey`.

Gli attributi annotati con `@DynamoDBAttribute`, ad esempio `some numbers`, saranno crittografati e firmati. Le eccezioni sono attributi che utilizzano le annotazioni di crittografia `@DoNotEncrypt` (solo firma) o `@DoNotTouch` (non crittografano né firmano) definite dal client di crittografia DynamoDB Encryption. Ad esempio, poiché l'attributo `leave me` ha un'annotazione `@DoNotTouch`, non sarà crittografato né firmato.

```
@DynamoDBTable(tableName = "ExampleTable")
public static final class DataPoJo {
  private String partitionAttribute;
  private int sortAttribute;
  private String example;
  private long someNumbers;
  private byte[] someBinary;
  private String leaveMe;

  @DynamoDBHashKey(attributeName = "partition_attribute")
  public String getPartitionAttribute() {
    return partitionAttribute;
  }

  public void setPartitionAttribute(String partitionAttribute) {
    this.partitionAttribute = partitionAttribute;
  }

  @DynamoDBRangeKey(attributeName = "sort_attribute")
  public int getSortAttribute() {
    return sortAttribute;
  }

  public void setSortAttribute(int sortAttribute) {
    this.sortAttribute = sortAttribute;
  }

  @DynamoDBAttribute(attributeName = "example")
  public String getExample() {
    return example;
  }
```

```
  public void setExample(String example) {
    this.example = example;
  }

  @DynamoDBAttribute(attributeName = "some numbers")
  public long getSomeNumbers() {
    return someNumbers;
  }

  public void setSomeNumbers(long someNumbers) {
    this.someNumbers = someNumbers;
  }

  @DynamoDBAttribute(attributeName = "and some binary")
  public byte[] getSomeBinary() {
    return someBinary;
  }

  public void setSomeBinary(byte[] someBinary) {
    this.someBinary = someBinary;
  }

  @DynamoDBAttribute(attributeName = "leave me")
  @DoNotTouch
  public String getLeaveMe() {
    return leaveMe;
  }

  public void setLeaveMe(String leaveMe) {
    this.leaveMe = leaveMe;
  }

  @Override
  public String toString() {
    return "DataPoJo [partitionAttribute=" + partitionAttribute + ", sortAttribute="
        + sortAttribute + ", example=" + example + ", someNumbers=" + someNumbers
        + ", someBinary=" + Arrays.toString(someBinary) + ", leaveMe=" + leaveMe +
 "]";
  }
}
```

Fase 4: crittografia e salvataggio di un item della tabella

Ora, quando si crea un elemento della tabella e si utilizza DynamoDB Mapper per salvarlo, l'elemento viene automaticamente crittografato e firmato prima di essere aggiunto alla tabella.

In questo esempio viene definito un item della tabella denominato `record`. Prima che venga salvato nella tabella, i suoi attributi vengono crittografati e firmati in base alle annotazioni nella classe `DataPoJo`. In questo caso, tutti gli attributi eccetto `PartitionAttribute`, `SortAttribute` e `LeaveMe` sono crittografati e firmati. `PartitionAttribute` e `SortAttributes` sono solo firmati. L'attributo `LeaveMe` non è crittografato né firmato.

Per crittografare e firmare l'item `record` e aggiungerlo alla tabella `ExampleTable`, chiama il metodo `save` della classe `DynamoDBMapper`. Poiché DynamoDB Mapper è configurato per utilizzare il comportamento PUT di salvataggio, l'elemento sostituisce qualsiasi elemento con le stesse chiavi primarie, anziché aggiornarlo. In questo modo le firme corrispondono ed è possibile decrittografare l'item quando si ottiene dalla tabella.

```
DataPoJo record = new DataPoJo();
record.setPartitionAttribute("is this");
record.setSortAttribute(55);
record.setExample("data");
record.setSomeNumbers(99);
record.setSomeBinary(new byte[]{0x00, 0x01, 0x02});
record.setLeaveMe("alone");

mapper.save(record);
```

Client di crittografia DynamoDB per Python

> **Note**
>
> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. L'argomento seguente fornisce informazioni sulle versioni 1. x —2. x del client di crittografia DynamoDB per Java e versioni 1. x —3. x del client di crittografia DynamoDB per Python. Per ulteriori informazioni, consulta AWSDatabase Encryption SDK per il supporto delle versioni di DynamoDB.

Questo argomento spiega come installare e utilizzare il client di crittografia DynamoDB per Python. Puoi trovare il codice nel aws-dynamodb-encryption-pythonrepository suGitHub, incluso un codice di esempio completo e testato per aiutarti a iniziare.

> **Note**
>
> Versioni 1. x. x e 2. x. X del DynamoDB Encryption Client per Python sono in nd-of-supportfase di entrata in vigore a luglio 2022. Effettua l'aggiornamento a una versione più recente il prima possibile.

Argomenti

Prerequisiti

Prima di installare il client di crittografia Amazon DynamoDB per Python, assicurati di avere i seguenti prerequisiti.

Una versione supportata di Python

Python 3.6 o versione successiva è richiesto dal client di crittografia Amazon DynamoDB per Python versioni 3.1.0 e successive. Per scaricare Python, consulta la pagina relativa ai download di Python.

Le versioni precedenti del client di crittografia Amazon DynamoDB per Python supportano Python 2.7 e Python 3.4 e versioni successive, ma ti consigliamo di utilizzare la versione più recente del client di crittografia DynamoDB.

Lo strumento di installazione pip per Python

Python 3.6 e versioni successive includono pip, anche se potresti volerlo aggiornare. Per ulteriori informazioni sull'aggiornamento o sull'installazione di pip, consulta la sezione relativa all'installazione nella documentazione su pip.

Installazione

Usa pip per installare il client di crittografia Amazon DynamoDB per Python, come illustrato negli esempi seguenti.

Per installare la versione più recente

```
pip install dynamodb-encryption-sdk
```

Per ulteriori dettagli sull'utilizzo di pip per installare e aggiornare pacchetti, consulta la sezione relativa all'installazione dei pacchetti.

Il client di crittografia DynamoDB richiede la libreria di crittografia su tutte le piattaforme. Tutte le versioni di pip installano e creano la libreria di crittografia su Windows. pip 8.1 e le versioni successive installano e creano la libreria di crittografia su Linux. Se utilizzi una versione precedente di pip e il tuo ambiente Linux non possiede gli strumenti necessari per creare la libreria di crittografia, devi installarli. Per ulteriori informazioni, consulta la sezione relativa alla creazione di una crittografia in Linux.

Puoi scaricare l'ultima versione di sviluppo del DynamoDB Encryption Client dal aws-dynamodb-encryption-pythonrepository in poi. GitHub

Dopo aver installato il client di crittografia DynamoDB, inizia guardando l'esempio di codice Python in questa guida.

Utilizzo del client di crittografia DynamoDB per Python

> **Note**
>
> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. L'argomento seguente fornisce informazioni sulle versioni 1. x —2. x del client di crittografia DynamoDB per Java e versioni 1. x —3. x del client di crittografia DynamoDB per Python. Per ulteriori informazioni, consulta AWSDatabase Encryption SDK per il supporto delle versioni di DynamoDB.

Questo argomento spiega alcune delle funzionalità del client di crittografia DynamoDB per Python che potrebbero non essere presenti in altre implementazioni del linguaggio di programmazione.

Queste funzionalità sono progettate per semplificare l'utilizzo del client di crittografia DynamoDB nel modo più sicuro. Ti consigliamo di utilizzarle a meno che il tuo caso d'uso non sia insolito.

Per dettagli sulla programmazione con il client di crittografia DynamoDB, consulta gli esempi di Python in questa guida, gli esempi nel aws-dynamodb-encryption-python repository su GitHub e la documentazione Python per il client di crittografia DynamoDB.

Argomenti

- Classi helper del client
- Classe TableInfo
- Operazioni di attributo in Python

Classi helper del client

Il client di crittografia DynamoDB per Python include diverse classi di supporto client che rispecchiano le classi Boto 3 per DynamoDB. Queste classi di supporto sono progettate per semplificare l'aggiunta di crittografia e firma all'applicazione DynamoDB esistente ed evitare i problemi più comuni, come segue:

- Impedisci di crittografare la chiave primaria del tuo elemento, aggiungendo un'azione di sostituzione per la chiave primaria all'AttributeActionsoggetto o generando un'eccezione se l'`AttributeActions`soggetto dice esplicitamente al client di crittografare la chiave primaria. Se l'azione predefinita nell'oggetto `AttributeActions` è `DO_NOTHING`, le classi helper del client utilizzano tale azione per la chiave primaria. Altrimenti, utilizzano `SIGN_ONLY`.
- Crea un TableInfooggetto e compila il contesto di crittografia DynamoDB in base a una chiamata a DynamoDB. Questo aiuta a garantire che il contesto di crittografia DynamoDB sia accurato e che il client possa identificare la chiave primaria.
- Metodi di supporto, come `put_item` e`get_item`, che crittografano e decrittografano in modo trasparente gli elementi della tabella quando si scrive o si legge da una tabella DynamoDB. L'unico metodo non supportato è `update_item`.

Puoi utilizzare le classi helper del client al posto dell'interazione diretta con il componente di crittografia dell'item di livello inferiore. Utilizza queste classi a meno che non sia necessario impostare opzioni avanzate nel componente di crittografia dell'item.

Le classi helper del client includono:

- EncryptedTableper le applicazioni che utilizzano la risorsa Tabella in DynamoDB per elaborare una tabella alla volta.
- EncryptedResourceper le applicazioni che utilizzano la classe Service Resource in DynamoDB per l'elaborazione in batch.
- EncryptedClientper le applicazioni che utilizzano il client di livello inferiore in DynamoDB.

Per utilizzare le classi di supporto del client, il chiamante deve disporre dell'autorizzazione per chiamare l'DescribeTableoperazione DynamoDB sulla tabella di destinazione.

Classe TableInfo

La TableInfoclasse è una classe di supporto che rappresenta una tabella DynamoDB, completa di campi per la chiave primaria e gli indici secondari. Ti consente di ottenere informazioni precise e in tempo reale sulla tabella.

Se utilizzi una classe helper del client, questa crea e utilizza un oggetto `TableInfo` per tuo conto. Altrimenti, puoi crearne uno esplicitamente. Per un esempio, consulta Utilizzo del componente di crittografia dell'item.

Quando si chiama il `refresh_indexed_attributes` metodo su un `TableInfo` oggetto, questo compila i valori delle proprietà dell'oggetto richiamando l'operazione DynamoDB DescribeTable. L'esecuzione di query sulla tabella è molto più affidabile rispetto all'impostazione come hardcoded dei nomi di indice. La `TableInfo` classe include anche una `encryption_context_values` proprietà che fornisce i valori richiesti per il contesto di crittografia DynamoDB.

Per utilizzare il `refresh_indexed_attributes` metodo, il chiamante deve disporre dell'autorizzazione per chiamare l'DescribeTableoperazione DynamoDB sulla tabella di destinazione.

Operazioni di attributo in Python

Le operazioni di attributo comunicano al componente di crittografia dell'item quali operazioni effettuare su ciascun attributo dell'item. Per specificare le operazioni di attributo in Python, creare un oggetto `AttributeActions` con un'operazione predefinita ed eventuali eccezioni per determinati attributi. I valori validi vengono definiti nel tipo enumerato `CryptoAction`.

> **Important**
>
> Dopo aver utilizzato le azioni degli attributi per crittografare gli elementi della tabella, l'aggiunta o la rimozione di attributi dal modello di dati potrebbe causare un errore di

convalida della firma che impedisce di decrittografare i dati. Per una spiegazione dettagliata, consulta Modifica del modello di dati.

```
DO_NOTHING = 0
SIGN_ONLY = 1
ENCRYPT_AND_SIGN = 2
```

Ad esempio, questo oggetto `AttributeActions` stabilisce l'operazione `ENCRYPT_AND_SIGN` come predefinita per tutti gli attributi e specifica le eccezioni per gli attributi `ISBN` e `PublicationYear`.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={
        'ISBN': CryptoAction.DO_NOTHING,
        'PublicationYear': CryptoAction.SIGN_ONLY
    }
)
```

Se utilizzi una classe helper del client, non è necessario specificare un'operazione di attributo per gli attributi della chiave primaria. Le classi helper del client impediscono la crittografia della chiave primaria.

Se non utilizzi una classe helper del client e l'operazione predefinita è `ENCRYPT_AND_SIGN`, devi specificare un'operazione per la chiave primaria. L'operazione consigliata per le chiavi primarie è `SIGN_ONLY`. Per semplificare la procedura, utilizza il metodo `set_index_keys`, che utilizza l'operazione SIGN_ONLY per le chiavi primarie o l'operazione DO_NOTHING quando questa è impostata come operazione predefinita.

Warning

Non crittografare gli attributi che vengono usati per la chiave primaria. Devono rimanere in testo normale in modo che DynamoDB possa trovare l'elemento senza eseguire una scansione completa della tabella.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
```

```
)
actions.set_index_keys(*table_info.protected_index_keys())
```

Codice di esempio per il client di crittografia DynamoDB per Python

> **Note**
>
> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. L'argomento seguente fornisce informazioni sulle versioni 1. x —2. x del client di crittografia DynamoDB per Java e versioni 1. x —3. x del client di crittografia DynamoDB per Python. Per ulteriori informazioni, consulta AWSDatabase Encryption SDK per il supporto delle versioni di DynamoDB.

Gli esempi seguenti mostrano come utilizzare il client di crittografia DynamoDB per Python per proteggere i dati DynamoDB nella tua applicazione. Puoi trovare altri esempi (e contribuire con i tuoi) nella cartella degli esempi del aws-dynamodb-encryption-pythonrepository su. GitHub

Argomenti

- Usa la classe EncryptedTable client helper
- Utilizzo del componente di crittografia dell'item

Usa la classe EncryptedTable client helper

L'esempio seguente mostra come utilizzare il provider KMS diretto con la `EncryptedTable` classe helper del client. Questo esempio utilizza lo stesso provider di materiali crittografici dell'esempio Utilizzo del componente di crittografia dell'item seguente. Tuttavia, utilizza la classe `EncryptedTable` invece di interagire direttamente con il componente di crittografia dell'item di livello inferiore.

Confrontando questi esempi, puoi visualizzare il lavoro che la classe helper del client esegue per tuo conto. Ciò include la creazione del contesto di crittografia DynamoDB e la verifica che gli attributi della chiave primaria siano sempre firmati, ma mai crittografati. Per creare il contesto di crittografia e scoprire la chiave primaria, le classi di supporto del client richiamano l'operazione DynamoDB DescribeTable. Per eseguire questo codice, devi disporre dell'autorizzazione per chiamare questa operazione.

Consulta l'esempio di codice completo: aws_kms_encrypted_table.py

Fase 1: creazione della tabella

Inizia creando un'istanza di una tabella DynamoDB standard con il nome della tabella.

```
table_name='test-table'
table = boto3.resource('dynamodb').Table(table_name)
```

Fase 2: creazione di un provider di materiali crittografici

Crea un'istanza del provider di materiali crittografici (cryptographic materials provider, CMP) selezionato.

Questo esempio utilizza il provider KMS diretto, ma puoi utilizzare qualunque CMP compatibile. Per creare un provider KMS diretto, specificare un AWS KMS key. Questo esempio utilizza l'Amazon Resource Name (ARN) diAWS KMS key, ma puoi utilizzare qualsiasi identificatore di chiave valido.

```
kms_key_id='arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key_id)
```

Fase 3: creazione dell'oggetto delle operazioni di attributo.

Le operazioni di attributo comunicano al componente di crittografia dell'item quali operazioni effettuare su ciascun attributo dell'item. L'oggetto `AttributeActions` in questo esempio crittografa e firma tutti gli item tranne l'attributo `test`, che viene ignorato.

Non devi specificare operazioni di attributo per gli attributi della chiave primaria quando utilizzi una classe helper del client. La classe `EncryptedTable` firma gli attributi della chiave primaria, ma non li crittografa mai.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={'test': CryptoAction.DO_NOTHING}
)
```

Fase 4: creazione della tabella crittografata

Crea la tabella criptata utilizzando la tabella standard, il provider KMS diretto e le operazioni di attributo. Questa fase completa la configurazione.

```
encrypted_table = EncryptedTable(
    table=table,
    materials_provider=kms_cmp,
    attribute_actions=actions
)
```

Fase 5: inserimento dell'item non crittografato nella tabella

Quando richiami il `put_item` metodo su`encrypted_table`, gli elementi della tabella vengono crittografati, firmati e aggiunti alla tabella DynamoDB in modo trasparente.

Come prima cosa, definisci l'item della tabella.

```
plaintext_item = {
    'partition_attribute': 'value1',
    'sort_attribute': 55
    'example': 'data',
    'numbers': 99,
    'binary': Binary(b'\x00\x01\x02'),
    'test': 'test-value'
}
```

Inseriscilo quindi nella tabella.

```
encrypted_table.put_item(Item=plaintext_item)
```

Per ottenere l'elemento dalla tabella DynamoDB nella sua forma crittografata, chiama il `get_item` metodo sull'`table`oggetto. Per ottenere l'item decrittografato, chiama il metodo `get_item` nell'oggetto `encrypted_table`.

Utilizzo del componente di crittografia dell'item

Questo esempio mostra come interagire direttamente con il criptatore degli elementi nel client di crittografia DynamoDB durante la crittografia degli elementi della tabella, anziché utilizzare le classi di supporto del client che interagiscono con il criptatore degli elementi per conto dell'utente.

Quando si utilizza questa tecnica, si crea manualmente il contesto di crittografia DynamoDB e l'oggetto di configurazione (`CryptoConfig`). Inoltre, è possibile crittografare gli elementi in una chiamata e inserirli nella tabella DynamoDB in una chiamata separata. Ciò consente di

personalizzare `put_item` le chiamate e utilizzare il client di crittografia DynamoDB per crittografare e firmare dati strutturati che non vengono mai inviati a DynamoDB.

Questo esempio utilizza il provider KMS diretto, ma puoi utilizzare qualunque CMP compatibile.

Consulta l'esempio di codice completo: aws_kms_encrypted_item.py

Fase 1: creazione della tabella

Inizia creando un'istanza di una risorsa di tabella DynamoDB standard con il nome della tabella.

```
table_name='test-table'
table = boto3.resource('dynamodb').Table(table_name)
```

Fase 2: creazione di un provider di materiali crittografici

Crea un'istanza del provider di materiali crittografici (cryptographic materials provider, CMP) selezionato.

Questo esempio utilizza il provider KMS diretto, ma puoi utilizzare qualunque CMP compatibile. Per creare un provider KMS diretto, specificare un AWS KMS key. Questo esempio utilizza l'Amazon Resource Name (ARN) diAWS KMS key, ma puoi utilizzare qualsiasi identificatore di chiave valido.

```
kms_key_id='arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key_id)
```

Fase 3: Usa la classe TableInfo helper

Per ottenere informazioni sulla tabella da DynamoDB, crea un'istanza della classe TableInfohelper. Quando lavori direttamente con il componente di crittografia dell'item, devi creare un'istanza `TableInfo` e chiamarne i metodi. Le classi helper del client eseguono questa operazione per tuo conto.

Il `refresh_indexed_attributes` metodo `TableInfo` utilizza l'operazione DescribeTableDynamoDB per ottenere informazioni accurate e in tempo reale sulla tabella. Queste comprendono la chiave primaria e gli indici secondari locali e globali. L'intermediario deve disporre dell'autorizzazione a chiamare `DescribeTable`.

```
table_info = TableInfo(name=table_name)
```

```
table_info.refresh_indexed_attributes(table.meta.client)
```

Fase 4: Creare il contesto di crittografia DynamoDB

Il contesto di crittografia DynamoDB contiene informazioni sulla struttura della tabella e su come viene crittografata e firmata. Questo esempio crea un contesto di crittografia DynamoDB in modo esplicito, poiché interagisce con il criptatore degli elementi. Le classi di supporto del client creano automaticamente il contesto di crittografia DynamoDB.

Per ottenere la chiave di partizione e la chiave di ordinamento, è possibile utilizzare le proprietà della classe TableInfohelper.

```
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)
)
```

Fase 5: creazione dell'oggetto delle operazioni di attributo.

Le operazioni di attributo comunicano al componente di crittografia dell'item quali operazioni effettuare su ciascun attributo dell'item. L'oggetto `AttributeActions` in questo esempio crittografa e firma tutti gli item tranne gli attributi della chiave primaria, che vengono firmati ma non crittografati, e l'attributo `test`, che viene ignorato.

Quando interagisci direttamente con il componente di crittografia dell'item e l'operazione predefinita è `ENCRYPT_AND_SIGN`, devi specificare un'operazione alternativa per la chiave primaria. Puoi utilizzare il metodo `set_index_keys`, che utilizza `SIGN_ONLY` per la chiave primaria o `DO_NOTHING` se questa è l'operazione predefinita.

Per specificare la chiave primaria, questo esempio utilizza le chiavi di indice nell'TableInfooggetto, che è popolato da una chiamata a DynamoDB. Questa tecnica è più sicura rispetto all'impostazione come hardcoded dei nomi della chiave primaria.

```
actions = AttributeActions(
```

```
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={'test': CryptoAction.DO_NOTHING}
)
actions.set_index_keys(*table_info.protected_index_keys())
```

Fase 6: creazione della configurazione dell'item

Per configurare il client di crittografia DynamoDB, utilizza gli oggetti appena creati in una CryptoConfigconfigurazione per l'elemento della tabella. Le classi client helper creano il file CryptoConfig per te.

```
crypto_config = CryptoConfig(
    materials_provider=kms_cmp,
    encryption_context=encryption_context,
    attribute_actions=actions
)
```

Fase 7: crittografia dell'item

Questo passaggio crittografa e firma l'elemento, ma non lo inserisce nella tabella DynamoDB.

Quando utilizzi una classe di supporto client, i tuoi elementi vengono crittografati e firmati in modo trasparente e quindi aggiunti alla tabella DynamoDB quando chiami il `put_item` metodo della classe helper. Quando utilizzi direttamente il componente di crittografia dell'item, le operazioni di crittografia e di inserimento sono indipendenti.

Crea prima un item non crittografato.

```
plaintext_item = {
    'partition_attribute': 'value1',
    'sort_key': 55,
    'example': 'data',
    'numbers': 99,
    'binary': Binary(b'\x00\x01\x02'),
    'test': 'test-value'
}
```

Poi, crittografalo e firmalo. Il metodo `encrypt_python_item` richiede l'oggetto di configurazione `CryptoConfig`.

```
encrypted_item = encrypt_python_item(plaintext_item, crypto_config)
```

Fase 8: inserimento dell'item nella tabella

Questo passaggio inserisce l'elemento crittografato e firmato nella tabella DynamoDB.

```
table.put_item(Item=encrypted_item)
```

Per visualizzare l'item crittografato, chiama il metodo `get_item` nell'oggetto `table` originale, invece che nell'oggetto `encrypted_table`. Il metodo ottiene l'item dalla tabella DynamoDB senza verificarlo né decrittografarlo.

```
encrypted_item = table.get_item(Key=partition_key)['Item']
```

L'immagine seguente mostra parte di un esempio di item della tabella crittografato e firmato.

I valori dell'attributo crittografato sono dati binari. I nomi e i valori degli attributi della chiave primaria (`partition_attribute` e `sort_attribute`) e l'attributo `test` restano non crittografati. L'output mostra anche l'attributo che contiene la firma (`*amzn-ddb-map-sig*`) e l'attributo di descrizione del materiale (`*amzn-ddb-map-desc*`).

```
{
    '*amzn-ddb-map-desc*': Binary(b'\x00\x00\x00\x00\x00\x00\x00\x10amzn-ddb-env-alg\
\x00\x00\x00\xe0AQEBAHhA84wnXjEJdBbBBylRUFcZZK2j7xwh6UyLoL28nQ
+0FAAAAH4wfAYJKoZIhvcNAQcGoG8wbQIBADBoBgkqhkiG9w0BBwEwHgYJYIZIAWUDBAEuMBEEDPeFBydmoJD
izYl0R0C4M7wAK6E1/N/bgTmHI=\x00\x00\x00\x17amzn-ddb-map-signingAlg\x00\x00\x00\nHmacS
\x00\x00\x00\x11/CBC/PKCS5Padding\x00\x00\x00\x10amzn-ddb-sig-alg\x00\x00\x00\x0eHmac
\x00\x00\x00\x0faws-kms-ec-attr\x00\x00\x00\x06*keys*'),
    '*amzn-ddb-map-sig*': Binary(b"\xd3\xc6\xc7\n\xb7#\x13\xd1Y\xea\xe4.|^\xbd\xdf\xe
    'binary': Binary(b'!"\xc5\x92\xd7\x13\x1d\xe8Bs\x9b\x7f\xa8\x8e\x9c\xcf\x10\x1e\x
    'example': Binary(b"'b\x933\x9a+s\xf1\xd6a\xc5\xd5\x1aZ\xed\xd6\xce\xe9X\xf0T\xcb
    'numbers': Binary(b'\xd5\xa0\\d\xcc\x85\xf5\x1e\xb9-f!\xb9\xb8\x8a\x1aT\xbaq\xf7\
    'partition_attribute': 'value1',
    'sort_attribute': 55,
    'test': 'test-value'
}
```

Modifica del modello di dati

> **Note**
>
> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. L'argomento seguente fornisce informazioni sulle versioni 1. x —2. x del client di crittografia

DynamoDB per Java e versioni 1. x —3. x del client di crittografia DynamoDB per Python. Per ulteriori informazioni, consulta AWSDatabase Encryption SDK per il supporto delle versioni di DynamoDB.

Ogni volta che crittografi o decrittografi un elemento, devi fornire azioni sugli attributi che indichino al client di crittografia DynamoDB quali attributi crittografare e firmare, quali attributi firmare (ma non crittografare) e quali ignorare. Le azioni relative agli attributi non vengono salvate nell'elemento crittografato e il client di crittografia DynamoDB non aggiorna automaticamente le azioni degli attributi.

Important

Il client di crittografia DynamoDB non supporta la crittografia dei dati delle tabelle DynamoDB esistenti e non crittografati.

Ogni volta che modifichi il modello di dati, ovvero quando aggiungi o rimuovi attributi dagli item della tabella, rischi di incorrere un errore. Se le operazioni di attributo specificate non valgono per tutti gli attributi nell'item, l'item potrebbe non essere crittografato e firmato secondo le tue intenzioni. E, cosa più importante, se le operazioni di attributo che fornisci quando decrittografi un item sono diverse da quelle che hai fornito quando lo hai crittografato, la verifica della firma potrebbe non andare a buon fine.

Ad esempio, se le operazioni di attributo usate per crittografare l'item gli comunicano di firmare l'attributo `test`, la firma nell'item comprenderà l'attributo `test`. Ma se le operazioni di attributo usate per decrittografare l'item non valgono per l'attributo `test`, la verifica non andrà a buon fine, perché il client proverà a verificare una firma che non comprende l'attributo `test`.

Questo è un problema particolare quando più applicazioni leggono e scrivono gli stessi elementi DynamoDB, perché il client di crittografia DynamoDB deve calcolare la stessa firma per gli elementi in tutte le applicazioni. È anche un problema per qualsiasi applicazione distribuita perché le modifiche nelle operazioni di attributo devono propagarsi a tutti gli host. Anche se alle tabelle DynamoDB accede un host in un unico processo, stabilire un processo basato sulle best practice aiuterà a prevenire errori nel caso in cui il progetto diventi più complesso.

Per evitare errori di convalida delle firme che impediscono la lettura degli item della tabella, utilizza le istruzioni riportate di seguito.

- Aggiungere un attributo: se il nuovo attributo modifica le azioni dell'attributo, implementa completamente la modifica dell'azione dell'attributo prima di includere il nuovo attributo in un elemento.
- Rimozione di un attributo: se smetti di usare un attributo nei tuoi articoli, non modificare le azioni degli attributi.
- Modifica dell'azione: dopo aver utilizzato una configurazione delle azioni degli attributi per crittografare gli elementi della tabella, non è possibile modificare in modo sicuro l'azione predefinita o l'azione per un attributo esistente senza ricrittografare tutti gli elementi della tabella.

Gli errori di convalida delle firme possono essere estremamente difficili da risolvere, quindi l'approccio migliore è prevenirli.

Argomenti

- Aggiunta di un attributo
- Rimozione di un attributo

Aggiunta di un attributo

Quando aggiungi un nuovo attributo agli item della tabella, potrebbe essere necessario modificare le operazioni di attributo. Per evitare errori di convalida delle firme, ti consigliamo di implementare questa modifica in un processo a due fasi. Verifica che la prima fase sia completata prima di iniziare la seconda fase.

1. Modifica le operazioni di attributo in tutte le applicazioni che leggono o scrivono nella tabella. Distribuisci queste modifiche e conferma che l'aggiornamento è stato propagato a tutti gli host di destinazione.
2. Scrivi i valori nel nuovo attributo negli item della tabella.

Questo approccio in due fasi garantisce che tutte le applicazioni e gli host abbiano le stesse operazioni di attributo e calcolerà la stessa firma prima che qualsiasi elemento incontri il nuovo attributo. Ciò è importante anche quando l'operazione di attributo è Non fare nulla (non crittografare o firmare), perché l'impostazione predefinita per alcuni sistemi di crittografia è crittografare e firmare.

Negli esempi seguenti viene illustrato il codice per la prima fase di questo processo. Viene aggiunto un nuovo attributo dell'item, `link`, che memorizza un collegamento a un altro item della tabella. Poiché questo collegamento deve rimanere in testo normale, l'esempio assegna l'operazione di sola

firma. Dopo aver distribuito completamente questa modifica e aver verificato che tutte le applicazioni e gli host abbiano le nuove operazioni di attributo, puoi iniziare a utilizzare l'attributo `link` negli item della tabella.

Java DynamoDB Mapper

Quando utilizzi `DynamoDB Mapper` e `AttributeEncryptor`, per impostazione predefinita, tutti gli attributi sono crittografati e firmati, tranne le chiavi primarie, che sono firmate ma non crittografate. Per specificare un'operazione di sola firma, utilizza l'annotazione `@DoNotEncrypt`.

In questo esempio viene utilizzata l'annotazione `@DoNotEncrypt` per il nuovo attributo `link`.

```
@DynamoDBTable(tableName = "ExampleTable")
public static final class DataPoJo {
  private String partitionAttribute;
  private int sortAttribute;
  private String link;

  @DynamoDBHashKey(attributeName = "partition_attribute")
  public String getPartitionAttribute() {
    return partitionAttribute;
  }

  public void setPartitionAttribute(String partitionAttribute) {
    this.partitionAttribute = partitionAttribute;
  }

  @DynamoDBRangeKey(attributeName = "sort_attribute")
  public int getSortAttribute() {
    return sortAttribute;
  }

  public void setSortAttribute(int sortAttribute) {
    this.sortAttribute = sortAttribute;
  }

  @DynamoDBAttribute(attributeName = "link")
  @DoNotEncrypt
  public String getLink() {
    return link;
  }

  public void setLink(String link) {
```

```
    this.link = link;
  }

  @Override
  public String toString() {
    return "DataPoJo [partitionAttribute=" + partitionAttribute + ",
        sortAttribute=" + sortAttribute + ",
        link=" + link + "]";
  }
}
```

Java DynamoDB encryptor

Nel criptatore DynamoDB di livello inferiore, è necessario impostare azioni per ogni attributo. In questo esempio viene utilizzata un'istruzione switch in cui l'impostazione predefinita è `encryptAndSign` e vengono specificate eccezioni per la chiave di partizione, la chiave di ordinamento e il nuovo attributo `link`. In questo esempio, se il codice attributo di collegamento non è stato distribuito completamente prima dell'utilizzo, l'attributo di collegamento verrà crittografato e firmato da alcune applicazioni, mentre verrà solo firmato da altre applicazioni.

```
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

Nel client di crittografia DynamoDB per Python, puoi specificare un'azione predefinita per tutti gli attributi e quindi specificare le eccezioni.

Se utilizzi una classe helper del client Python, non devi specificare un'operazione di attributo per gli attributi della chiave primaria. Le classi helper del client impediscono la crittografia della chiave primaria. Tuttavia, se utilizzi una classe helper del client, devi impostare l'operazione SIGN_ONLY sulla chiave di partizione e la chiave di ordinamento. Se esegui accidentalmente la crittografia della partizione o della chiave di ordinamento, non potrai recuperare i dati senza una scansione completa della tabella.

In questo esempio viene specificata un'eccezione per il nuovo attributo `link`, che ottiene l'operazione `SIGN_ONLY`.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={
      'example': CryptoAction.DO_NOTHING,
      'link': CryptoAction.SIGN_ONLY
    }
)
```

Rimozione di un attributo

Se non hai più bisogno di un attributo negli elementi che sono stati crittografati con il client di crittografia DynamoDB, puoi smettere di usare l'attributo. Tuttavia, non eliminare o modificare l'operazione per tale attributo. In tal caso, se viene riscontrato un item con tale attributo, la firma calcolata per l'item non corrisponderà alla firma originale e la convalida della firma avrà esito negativo.

Anche se potresti essere tentato di rimuovere tutte le tracce dell'attributo dal tuo codice, aggiungi un commento che indica che l'item non è più utilizzato invece di eliminarlo. Anche se esegui una scansione completa della tabella per eliminare tutte le istanze dell'attributo, un item crittografato con tale attributo potrebbe essere memorizzato nella cache o in fase di elaborazione in qualche punto della configurazione.

Risoluzione dei problemi nell'applicazione DynamoDB Encryption Client

> **Note**
>
> La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. L'argomento seguente fornisce informazioni sulle versioni 1. x —2. x del client di crittografia DynamoDB per Java e versioni 1. x —3. x del client di crittografia DynamoDB per Python. Per ulteriori informazioni, consulta AWSDatabase Encryption SDK per il supporto delle versioni di DynamoDB.

Questa sezione descrive i problemi che potresti riscontrare durante l'utilizzo del client di crittografia DynamoDB e offre suggerimenti per risolverli.

Per fornire feedback sul client di crittografia DynamoDB, invia un problema nel repository aws-dynamodb-encryption-javaor aws-dynamodb-encryption-pythonGitHub.

Per fornire feedback su questa documentazione, utilizzare il link di feedback in qualsiasi pagina. Puoi anche segnalare un problema o contribuire all'aws-dynamodb-encryption-docsarchivio open source su cui è contenuta questa documentazione. GitHub

Argomenti

Accesso negato

Problema: la tua applicazione non può accedere a una risorsa necessaria.

Suggerimento: scopri le autorizzazioni richieste e aggiungile al contesto di sicurezza in cui opera la tua applicazione.

Dettagli

Per eseguire un'applicazione che utilizza una libreria DynamoDB Encryption Client, il chiamante deve disporre dell'autorizzazione per utilizzarne i componenti. In caso contrario, le applicazioni non potranno accedere agli elementi richiesti.

- Il client di crittografia DynamoDB non richiede un account Amazon Web Services (AWS) né dipende da alcun AWS servizio. Tuttavia, se l'applicazione utilizzaAWS, è necessario disporre di un account Account AWS e di utenti autorizzati a utilizzare l'account.
- Il client di crittografia DynamoDB non richiede Amazon DynamoDB. Tuttavia, se l'applicazione che utilizza il client crea tabelle DynamoDB, inserisce elementi in una tabella o ottiene elementi da una tabella, il chiamante deve disporre dell'autorizzazione per utilizzare le operazioni DynamoDB richieste nella tua. Account AWS Per i dettagli, consulta gli argomenti relativi al controllo degli accessi nella Amazon DynamoDB Developer Guide.
- Se l'applicazione utilizza una classe di supporto client nel client di crittografia DynamoDB per Python, il chiamante deve disporre dell'autorizzazione per chiamare l'operazione DynamoDB. DescribeTable
- Il client di crittografia DynamoDB non richiede AWS Key Management Service ()AWS KMS. Tuttavia, se l'applicazione utilizza un Direct KMS Materials Provider o utilizza un Most Recent Provider con un provider store che utilizzaAWS KMS, il chiamante deve disporre dell'autorizzazione per utilizzare le operazioni AWS KMS GenerateDataKeye Decrypt.

La verifica della firma non va a buon fine

Problema: un item non può essere decrittografato perché la verifica della firma non va a buon fine. L'item potrebbe anche non essere crittografato e firmato secondo le tue intenzioni.

Suggerimento: assicurati che tutte le operazioni di attributo valgano per tutti gli attributi dell'item. Quando decrittografi un item, assicurati di fornire operazioni di attributo che corrispondano a quelle utilizzate per crittografare l'item.

Dettagli

Le azioni relative agli attributi fornite indicano al client di crittografia DynamoDB quali attributi crittografare e firmare, quali firmare (ma non crittografare) e quali ignorare.

Se le operazioni di attributo specificate non valgono per tutti gli attributi nell'item, l'item potrebbe non essere crittografato e firmato secondo le tue intenzioni. Se le operazioni di attributo che fornisci quando decrittografi un item sono diverse da quelle che hai fornito quando lo hai crittografato, la verifica della firma potrebbe non andare a buon fine. Questo è problema tipico delle applicazioni distribuite, in cui le nuove operazioni di attributo non sono state propagate a tutti gli host.

Gli errori di convalida delle firme sono difficili da risolvere. Per aiutare a prevenirli, adotta ulteriori precauzioni quando modifichi il modello di dati. Per informazioni dettagliate, consultare Modifica del modello di dati.

Problemi con le tabelle globali delle versioni precedenti

Problema: gli elementi in una versione precedente della tabella globale di Amazon DynamoDB non possono essere decrittografati perché la verifica della firma non riesce.

Suggerimento: imposta le azioni degli attributi in modo che i campi di replica riservati non siano crittografati o firmati.

Dettagli

È possibile utilizzare il client di crittografia DynamoDB con tabelle globali DynamoDB. Ti consigliamo di utilizzare tabelle globali con una chiave KMS multiregionale e di replicare la chiave KMS in tutti i punti in Regioni AWS cui viene replicata la tabella globale.

A partire dalla versione 2019.11.21 delle tabelle globali, puoi utilizzare le tabelle globali con il client di crittografia DynamoDB senza alcuna configurazione speciale. Tuttavia, se utilizzi la versione 2017.11.29 delle tabelle globali, devi assicurarti che i campi di replica riservati non siano crittografati o firmati.

Se si utilizza la versione 2017.11.29 delle tabelle globali, è necessario impostare le azioni degli attributi per i seguenti attributi `DO_NOTHING` in Java o in Python. `@DoNotTouch`

- `aws:rep:deleting`
- `aws:rep:updatetime`
- `aws:rep:updateregion`

Se si utilizza qualsiasi altra versione di tabelle globali, non è richiesta alcuna azione.

Scarse prestazioni del provider più recente

Problema: l'applicazione è meno reattiva, soprattutto dopo l'aggiornamento a una versione più recente del client di crittografia DynamoDB.

Suggerimento: modifica il time-to-live valore e la dimensione della cache.

Dettagli

Il provider più recente è progettato per migliorare le prestazioni delle applicazioni che utilizzano il client di crittografia DynamoDB, consentendo un riutilizzo limitato dei materiali crittografici. Quando configuri il provider più recente per la tua applicazione, devi bilanciare il miglioramento delle prestazioni con i problemi di sicurezza derivanti dalla memorizzazione nella cache e dal riutilizzo.

Nelle versioni più recenti del client di crittografia DynamoDB, il valore time-to-live (TTL) determina per quanto tempo possono essere utilizzati i fornitori di materiale crittografico (CMP) memorizzati nella cache. Il TTL determina anche la frequenza con cui il provider più recente verifica la presenza di una nuova versione della CMP.

Se il TTL è troppo lungo, l'applicazione potrebbe violare le regole aziendali o gli standard di sicurezza. Se il TTL è troppo breve, le frequenti chiamate all'archivio del provider possono far sì che l'archivio del provider limiti le richieste provenienti dall'applicazione e da altre applicazioni che condividono l'account del servizio. Per risolvere questo problema, regola il TTL e la dimensione della cache su un valore che soddisfi i tuoi obiettivi di latenza e disponibilità e sia conforme ai tuoi standard di sicurezza. Per informazioni dettagliate, consulta Impostazione di un time-to-live valore.

Rinomina del client di crittografia Amazon DynamoDB

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

Il 9 giugno 2023, la nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. L'SDK per la crittografia del AWS database è compatibile con Amazon DynamoDB. Può decrittografare e leggere gli elementi crittografati dal vecchio client di crittografia DynamoDB. Per ulteriori informazioni sulle versioni precedenti di DynamoDB Encryption Client, vedere. AWSSupporto per la versione di Database Encryption SDK per DynamoDB

Il AWS Database Encryption SDK fornisce la versione 3. x della libreria di crittografia lato client Java per DynamoDB, che è un'importante riscrittura del client di crittografia DynamoDB per Java. Include molti aggiornamenti, come un nuovo formato di dati strutturati, un supporto multitenancy migliorato, modifiche allo schema senza interruzioni e supporto per la crittografia ricercabile.

Per ulteriori informazioni sulle nuove funzionalità introdotte con AWS Database Encryption SDK, consulta i seguenti argomenti.

Crittografia ricercabile

È possibile progettare database in grado di cercare record crittografati senza decrittografare l'intero database. A seconda del modello di minaccia e dei requisiti delle query, puoi utilizzare la crittografia ricercabile per eseguire ricerche con corrispondenza esatta o query complesse più personalizzate sui tuoi record crittografati.

Portachiavi

Il AWS Database Encryption SDK utilizza i portachiavi per eseguire la crittografia delle buste. I portachiavi generano, crittografano e decrittografano le chiavi dati che proteggono i tuoi record. Il AWS Database Encryption SDK supporta AWS KMS portachiavi che utilizzano la crittografia simmetrica o RSA asimmetrica AWS KMS keysper proteggere le chiavi dei dati e i portachiavi AWS KMS gerarchici che consentono di proteggere i materiali crittografici con una chiave KMS con crittografia simmetrica senza chiamare ogni volta che si crittografa o decrittografa un record. AWS KMS Puoi anche specificare il tuo materiale chiave con i portachiavi Raw AES e i portachiavi Raw RSA.

Modifiche allo schema senza interruzioni

Quando configuri AWS Database Encryption SDK, fornisci azioni crittografiche che indicano al client quali campi crittografare e firmare, quali firmare (ma non crittografare) e quali ignorare. Dopo aver utilizzato AWS Database Encryption SDK per proteggere i tuoi record, puoi comunque apportare modifiche al tuo modello di dati. Puoi aggiornare le tue azioni crittografiche, come l'aggiunta o la rimozione di campi crittografati, in un'unica distribuzione.

Configurazione delle tabelle DynamoDB esistenti per la crittografia lato client

Le versioni precedenti del client di crittografia DynamoDB sono state progettate per essere implementate in nuove tabelle non popolate. Con il AWS Database Encryption SDK per DynamoDB, puoi migrare le tabelle Amazon DynamoDB esistenti alla versione 3. x della libreria di crittografia lato client Java per DynamoDB.

Riferimento

La nostra libreria di crittografia lato client è stata rinominata AWS Database Encryption SDK. Questa guida per gli sviluppatori fornisce comunque informazioni sul client di crittografia DynamoDB.

I seguenti argomenti forniscono dettagli tecnici per il AWS Database Encryption SDK.

Formato di descrizione del materiale

La descrizione del materiale funge da intestazione per un record crittografato. Quando crittografi e firmi i campi con AWS Database Encryption SDK, il crittografo registra la descrizione del materiale mentre assembla i materiali crittografici e archivia la descrizione del materiale in un nuovo campo (`aws_dbe_head`) che il crittografo aggiunge al tuo record. La descrizione del materiale è una struttura di dati formattata portatile che contiene la chiave di dati crittografata e informazioni su come il record è stato crittografato e firmato. La tabella seguente descrive i valori che costituiscono la descrizione del materiale. I byte vengono aggiunti nell'ordine mostrato.

Value (Valore)	Lunghezza in byte
Versione	1
Firme abilitate	1
ID record	32
Crittografa legenda	Variabile
Lunghezza del contesto di crittografia	2
Contesto di crittografia	Variabile
Conteggio chiave dati crittografati	1
Chiavi dati crittografate	Variabile
Impegno record	1

Versione

La versione del formato di questo `aws_dbe_head` campo.

Firme abilitate

Codifica se le firme sono abilitate per questo record.

Valore in byte	Significato
`0x01`	Firme abilitate (impostazione predefinita)
`0x00`	Firme disattivate

ID record

Un valore a 256 bit generato casualmente che identifica il record. L'ID del record:

- Identifica in modo univoco il record crittografato.
- Associa la descrizione del materiale al record crittografato.

Crittografa legenda

Una descrizione serializzata dei campi autenticati che sono stati crittografati. La legenda della crittografia viene utilizzata per determinare quali campi il metodo di decrittografia deve tentare di decrittografare.

Valore in byte	Significato
`0x65`	`ENCRYPT_AND_SIGN`
`0x73`	`SIGN_ONLY`

The Encrypt Legend è serializzato come segue:

1. Lessicograficamente con la sequenza di byte che rappresenta il loro percorso canonico.
2. Per ogni campo, in ordine, aggiungi uno dei valori di byte sopra specificati per indicare se quel campo deve essere crittografato.

Lunghezza del contesto di crittografia

La lunghezza del contesto di crittografia. È un valore a 2 byte interpretato come un numero intero senza segno a 16 bit. La lunghezza massima è di 65.535 byte.

Contesto di crittografia

Un set di coppie nome-valore che contengono dati autenticati aggiuntivi arbitrari e non segreti.

Quando le firme digitali sono abilitate, il contesto di crittografia contiene la coppia chiave-valore. `{"aws-crypto-footer-ecdsa-key": Qtxt}` `Qtxt`rappresenta il punto della curva ellittica `Q` compresso secondo la versione 2.0 di SEC 1 e quindi codificato in base64.

Conteggio chiave dati crittografati

Il numero di chiavi di dati crittografati. È un valore a 1 byte interpretato come un numero intero senza segno a 8 bit che specifica il numero di chiavi dati crittografate. Il numero massimo di chiavi dati crittografate in ogni record è 255.

Chiavi dati crittografate

Sequenza di chiavi di dati crittografati. La lunghezza della sequenza è determinata dal numero di chiavi di dati crittografati e dalla lunghezza di ciascuna. La sequenza contiene almeno una chiave di dati crittografati.

La tabella seguente descrive i campi che costituiscono ogni chiave di dati crittografati. I byte vengono aggiunti nell'ordine mostrato.

Struttura chiave dati crittografati

Campo	Lunghezza in byte
Lunghezza ID provider chiave	2
ID provider chiave	Variabile. Pari al valore specificato nei 2 byte precedenti (lunghezza ID provider chiave).
Lunghezza informazioni provider chiave	2
Informazioni provider chiave	Variabile. Pari al valore specificato nei 2 byte precedenti (lunghezza informazione provider chiave).
Lunghezza chiave dati crittografati	2

Campo	Lunghezza in byte
Chiave di dati crittografati	Variabile. Pari al valore specificato nei 2 byte precedenti (lunghezza chiave dati crittografati).

Lunghezza ID provider chiave

Lunghezza dell'identificatore del provider della chiave. Si tratta di un valore di 2 byte interpretato come un numero intero senza segno a 16 bit che specifica il numero di byte che contengono l'ID del provider della chiave.

ID provider chiave

Identificatore del provider della chiave. Viene utilizzato per indicare il provider della chiave dei dati crittografati ed è destinato a essere ampliabile.

Lunghezza informazioni provider chiave

Lunghezza delle informazioni del provider della chiave. Si tratta di un valore di 2 byte interpretato come un numero intero senza segno a 16 bit che specifica il numero di byte che contengono le informazioni del provider della chiave.

Informazioni provider chiave

Informazioni provider chiave. Dipende dal provider di chiavi.

Quando utilizzi un AWS KMS portachiavi, questo valore contiene l'Amazon Resource Name (ARN) di. AWS KMS key

Lunghezza chiave dati crittografati

La lunghezza della chiave di dati crittografati. Si tratta di un valore di 2 byte interpretato come un numero intero senza segno a 16 bit che specifica il numero di byte che contengono la chiave di dati crittografati.

Chiave di dati crittografati

Chiave di dati crittografati. È la chiave dati crittografata dal fornitore della chiave.

Impegno record

Un hash distinto HMAC (Hash-Based Message Authentication Code) a 256 bit calcolato su tutti i byte di descrizione del materiale precedenti utilizzando la chiave di impegno.

AWS KMSDettagli tecnici del portachiavi gerarchico

Il portachiavi AWS KMS gerarchico utilizza una chiave dati univoca per crittografare ogni campo e crittografa ogni chiave dati con una chiave di avvolgimento univoca derivata da una chiave di ramo attiva. Utilizza una derivazione chiave in modalità counter con una funzione pseudocasuale con HMAC SHA-256 per derivare la chiave di wrapping a 32 byte con i seguenti input.

- Un sale casuale da 16 byte
- La chiave di filiale attiva
- Il valore codificato UTF-8 per l'identificatore del fornitore di chiavi "» aws-kms-hierarchy

Il portachiavi gerarchico utilizza la chiave di avvolgimento derivata per crittografare una copia della chiave dati in testo normale utilizzando AES-GCM-256 con un tag di autenticazione a 16 byte e i seguenti input.

- La chiave di wrapping derivata viene utilizzata come chiave di cifratura AES-GCM
- La chiave dati viene utilizzata come messaggio AES-GCM
- Un vettore di inizializzazione casuale a 12 byte (IV) viene utilizzato come AES-GCM IV
- Dati autenticati aggiuntivi (AAD) contenenti i seguenti valori serializzati.

Value (Valore)	Lunghezza in byte	Interpretato come
"aws-kms-hierarchy"	17	codifica UTF-8
L'identificatore della chiave di filiale	Variabile	codifica UTF-8
La versione con chiave di ramo	16	codifica UTF-8
Contesto di crittografia	Variabile	Coppie di valori chiave codificate UTF-8

Cronologia dei documenti per la AWS Database Encryption SDK Developer Guide

La tabella seguente descrive le modifiche significative apportate a questa documentazione. Oltre a queste modifiche maggiori, aggiorniamo la documentazione di frequente per migliorare le descrizioni e gli esempi e per dar spazio al feedback inviatoci. Per ricevere una notifica sulle modifiche rilevanti, iscriversi al feed RSS.

Modifica	Descrizione	Data
Versione General Availability (GA)	Documentazione aggiornat a per la versione GA della versione 3. x della libreria di crittografia lato client Java per DynamoDB. **Warning** Le chiavi di diramazio ne create durante la versione di anteprima per sviluppatori non sono più supportate.	24 luglio 2023
Nuovo marchio del client di crittografia DynamoDB	La libreria di crittografia lato client viene rinominata AWS Database Encryption SDK.	9 giugno 2023
Versione di anteprima	Documentazione aggiunta e aggiornata per la versione 3. x della libreria di crittografia lato client Java per DynamoDB, che include un nuovo formato di dati strutturati, supporto multitenancy migliorato, modifiche allo schema senza	9 giugno 2023

	interruzioni e supporto per la crittografia ricercabile.	
Modifica della documentazione	Sostituisci il AWS Key Management Service termine chiave master del cliente (CMK) con una AWS KMS keychiave KMS.	30 agosto 2021
Nuova caratteristica	È stato aggiunto il supporto per AWS Key Managemen t Service (AWS KMS) chiavi multiregionali. Le chiavi multiregionali sono AWS KMS chiavi diverse Regioni AWS che possono essere utilizzat e in modo intercambiabile perché hanno lo stesso ID chiave e lo stesso materiale chiave.	8 giugno 2021
Nuovo esempio	Aggiunto esempio di utilizzo di DynamoDBMapper in Java.	6 settembre 2018
Supporto per Python	Aggiunto il supporto per Python, oltre a Java.	2 maggio 2018
Versione iniziale	Versione iniziale di questa documentazione.	2 maggio 2018

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.