aws

Panduan Developer

AWS SDK Enkripsi Basis Data

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS SDK Enkripsi Basis Data: Panduan Developer

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Merek dagang dan tampilan dagang Amazon tidak boleh digunakan sehubungan dengan produk atau layanan apa pun yang bukan milik Amazon, dengan cara apa pun yang dapat menyebabkan kebingungan antara para pelanggan, atau dengan cara apa pun yang menghina atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon merupakan hak milik masing-masing pemiliknya, yang mungkin atau tidak terafiliasi, terkait dengan, atau disponsori oleh Amazon.

Table of Contents

Apa itu SDK Enkripsi AWS Database? 1
Dikembangkan dalam repositori sumber terbuka 3
Support dan pemeliharaan 3
Mengirim umpan balik 3
Konsep 4
Enkripsi amplop 5
Kunci data 6
Kunci pembungkus 7
Gantungan kunci 8
Tindakan kriptografi 8
Deskripsi materi 9
Konteks enkripsi 10
Manajer materi kriptografi 11
Enkripsi simetris dan asimetris 11
Komitmen utama 12
Tanda tangan digital 12
Cara kerjanya 14
Enkripsi dan tandatangani 15
Dekripsi dan verifikasi 16
Suite algoritma yang didukung 17
Rangkaian algoritme default 20
AES-GCM tanpa tanda tangan digital ECDSA 20
Berinteraksi dengan AWS KMS 22
Mengkonfigurasi SDK 24
Memilih bahasa pemrograman 24
Memilih tombol pembungkus 24
Membuat filter penemuan 26
Bekerja dengan basis data multitenant 27
Membuat suar yang ditandatangani 28
Toko-toko utama 36
Terminologi dan konsep toko kunci 36
Menerapkan izin yang paling tidak diistimewakan 37
Buat toko kunci 38
Konfigurasikan tindakan penyimpanan kunci 39

Konfigurasikan tindakan penyimpanan kunci Anda 40
Buatlah sebuah cabang 43
Putar kunci cabang aktif Anda 47
Gantungan kunci 49
Cara kerja gantungan kunci 50
AWS KMS gantungan kunci 51
Izin yang diperlukan untuk keyrings AWS KMS 52
Mengidentifikasi AWS KMS keys dalam AWS KMS keyring 52
Membuat AWS KMS keyring 54
Menggunakan Multi-region AWS KMS keys 57
Menggunakan AWS KMS keyring penemuan 59
Menggunakan AWS KMS keyring penemuan regional 61
AWS KMS Gantungan kunci hierarkis 64
Cara kerjanya 66
Prasyarat 67
Izin yang diperlukan 68
Pilih cache 68
Buat keyring Hierarkis 78
Menggunakan keyring Hierarkis untuk enkripsi yang dapat dicari 84
AWS KMS Gantungan kunci ECDH 88
Izin yang diperlukan untuk gantungan kunci AWS KMS ECDH 89
Membuat keyring AWS KMS ECDH 90
Membuat keyring AWS KMS penemuan ECDH 93
Gantungan kunci AES mentah 96
Gantungan kunci RSA mentah 98
Gantungan kunci ECDH mentah 102
Membuat keyring ECDH mentah 103
Multi-gantungan kunci 112
Enkripsi yang dapat dicari 117
Apakah beacon tepat untuk dataset saya? 118
Skenario enkripsi yang dapat dicari 121
Beacon 122
Suar standar 123
Suar majemuk 125
Merencanakan suar 126
Pertimbangan untuk database multitenant 127

Memilih jenis suar 128
Memilih panjang suar 134
Memilih nama suar 141
Mengkonfigurasi beacon 141
Mengkonfigurasi beacon standar 142
Mengkonfigurasi suar majemuk 152
Contoh konfigurasi 162
Menggunakan beacon 167
Meminta suar 170
Enkripsi yang dapat dicari untuk database multitenant 171
Menanyakan beacon dalam database multitenant 174
Amazon DynamoDB 176
Enkripsi di sisi klien dan sisi server 177
Bidang mana yang dienkripsi dan ditandatangani? 179
Enkripsi nilai atribut 180
Penandatanganan item 181
Enkripsi yang dapat dicari di DynamoDB 181
Mengkonfigurasi indeks sekunder dengan beacon 181
Menguji output suar 183
Memperbarui model data Anda 189
Tambahkan SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT atribut baru ENCRYPT_AND_SIGNSIGN_ONLY, dan 191
Hapus atribut yang ada 191
Ubah ENCRYPT_AND_SIGN atribut yang ada ke SIGN_ONLY atau SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT 192
Ubah SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT atribut SIGN_ONLY atau yang sudah ada ke ENCRYPT_AND_SIGN 193
Tambahkan DO_NOTHING atribut baru 193
Ubah SIGN_ONLY atribut yang ada menjadi SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT 195
Ubah SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT atribut yang ada menjadi SIGN_ONLY 195
Bahasa pemrograman 196
Java 196
.NET 232
Rust 248

Warisan 254
AWS SDK Enkripsi Database untuk dukungan versi DynamoDB 255
Cara kerjanya 256
Konsep 259
Penyedia bahan kriptografi 264
Bahasa pemrograman 295
Mengubah model data Anda 323
Pemecahan Masalah 328
Ganti nama Klien Enkripsi DynamoDB 332
Referensi 334
Format deskripsi bahan 334
AWS KMS Rincian teknis keyring hierarkis 338
Riwayat dokumen 339
..... cccxli

Apa itu SDK Enkripsi AWS Database?

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

AWS Database Encryption SDK adalah sekumpulan pustaka perangkat lunak yang memungkinkan Anda menyertakan enkripsi sisi klien dalam desain database Anda. AWS Database Encryption SDK menyediakan solusi enkripsi tingkat rekaman. Anda menentukan bidang mana yang dienkripsi dan bidang mana yang disertakan dalam tanda tangan yang memastikan keaslian data Anda. Mengenkripsi data sensitif Anda saat transit dan diam membantu memastikan bahwa data teks biasa Anda tidak tersedia untuk pihak ketiga mana pun, termasuk. AWS AWS Database Encryption SDK disediakan secara gratis di bawah lisensi Apache 2.0.

Panduan pengembang ini memberikan gambaran konseptual SDK Enkripsi AWS Database, termasuk pengenalan arsitekturnya, detail tentang cara melindungi data Anda, perbedaannya dari enkripsi sisi server, dan panduan memilih komponen penting untuk aplikasi Anda untuk membantu Anda memulai.

SDK Enkripsi AWS Database mendukung Amazon DynamoDB dengan enkripsi tingkat atribut.

SDK Enkripsi AWS Database memiliki manfaat sebagai berikut:

Dirancang khusus untuk aplikasi database

- Anda tidak perlu menjadi ahli kriptografi untuk menggunakan AWS Database Encryption SDK. Implementasi termasuk metode pembantu yang dirancang untuk bekerja dengan aplikasi yang ada.
- Setelah Anda membuat dan mengonfigurasi komponen yang diperlukan, klien enkripsi secara transparan mengenkripsi dan menandatangani catatan Anda saat Anda menambahkannya ke database, dan memverifikasi serta mendekripsi saat Anda mengambilnya.

Termasuk aman dan penandatanganan yang aman

- SDK Enkripsi AWS Database mencakup implementasi aman yang mengenkripsi nilai bidang di setiap catatan menggunakan kunci enkripsi data unik, lalu menandatangani catatan untuk melindunginya dari perubahan yang tidak sah, seperti menambahkan atau menghapus bidang, atau menukar nilai terenkripsi.

Menggunakan materi kriptografis dari sumber mana pun

AWS Database Encryption SDK menggunakan keyrings untuk menghasilkan, mengenkripsi, dan mendekripsi kunci enkripsi data unik yang melindungi catatan Anda. Keyrings menentukan kunci pembungkus yang mengenkripsi kunci data tersebut.

Anda dapat menggunakan kunci pembungkus dari sumber apa pun, termasuk layanan kriptografi, seperti AWS Key Management Service(AWS KMS) atau. AWS CloudHSM SDK Enkripsi AWS Database tidak memerlukan layanan Akun AWS atau AWS layanan apa pun.

Support untuk caching materi kriptografi

AWS KMS Hierarchical keyring adalah solusi caching materi kriptografi yang mengurangi jumlah AWS KMS panggilan dengan menggunakan kunci cabang yang AWS KMS dilindungi yang disimpan dalam tabel Amazon DynamoDB, dan kemudian secara lokal menyimpan materi kunci cabang yang digunakan dalam operasi enkripsi dan dekripsi. Ini memungkinkan Anda untuk melindungi materi kriptografi Anda di bawah kunci KMS enkripsi simetris tanpa menelepon AWS KMS setiap kali Anda mengenkripsi atau mendekripsi catatan. AWS KMS Hierarchical keyring adalah pilihan yang baik untuk aplikasi yang perlu meminimalkan panggilan ke. AWS KMS

Enkripsi yang dapat dicari

Anda dapat merancang database yang dapat mencari catatan terenkripsi tanpa mendekripsi seluruh database. Bergantung pada model ancaman dan persyaratan kueri, Anda dapat menggunakan enkripsi yang dapat dicari untuk melakukan penelusuran yang sama persis atau kueri kompleks yang lebih disesuaikan pada basis data terenkripsi Anda.

Support untuk skema database multitenant

SDK Enkripsi AWS Database memungkinkan Anda untuk melindungi data yang disimpan dalam database dengan skema bersama dengan mengisolasi setiap penyewa dengan bahan enkripsi yang berbeda. Jika Anda memiliki beberapa pengguna yang melakukan operasi enkripsi dalam database Anda, gunakan salah satu AWS KMS keyrings untuk menyediakan setiap pengguna dengan kunci yang berbeda untuk digunakan dalam operasi kriptografi mereka. Untuk informasi selengkapnya, lihat Bekerja dengan basis data multitenant.

Support untuk pembaruan skema yang mulus

Saat Anda mengonfigurasi SDK Enkripsi AWS Database, Anda memberikan tindakan kriptografi yang memberi tahu klien bidang mana yang akan dienkripsi dan ditandatangani, bidang mana yang akan ditandatangani (tetapi tidak dienkripsi), dan mana yang harus diabaikan. Setelah Anda

menggunakan AWS Database Encryption SDK untuk melindungi catatan Anda, Anda masih dapat membuat perubahan pada model data Anda. Anda dapat memperbarui tindakan kriptografi Anda, seperti menambahkan atau menghapus bidang terenkripsi, dalam satu penerapan.

Dikembangkan dalam repositori sumber terbuka

AWS Database Encryption SDK dikembangkan dalam repositori open-source pada. GitHub Anda dapat menggunakan repositori ini untuk melihat kode, membaca dan mengirimkan masalah, dan menemukan informasi yang spesifik untuk implementasi Anda.

SDK Enkripsi AWS Database untuk DynamoDB

- Repositori aws-database-encryption-sdk-dynamodb aktif GitHub mendukung versi terbaru SDK AWS Enkripsi Database untuk DynamoDB di Java, .NET, dan Rust.

  SDK Enkripsi AWS Database untuk DynamoDB adalah produk dari Dafny, bahasa yang sadar verifikasi di mana Anda menulis spesifikasi, kode untuk mengimplementasikannya, dan bukti untuk mengujinya. Hasilnya adalah pustaka yang mengimplementasikan fitur SDK Enkripsi AWS Database untuk DynamoDB dalam kerangka kerja yang menjamin kebenaran fungsional.

Support dan pemeliharaan

SDK Enkripsi AWS Database menggunakan kebijakan pemeliharaan yang sama dengan yang digunakan AWS SDK dan Tools, termasuk fase pembuatan versi dan siklus hidupnya. Sebagai praktik terbaik, kami menyarankan Anda menggunakan SDK Enkripsi AWS Database versi terbaru yang tersedia untuk implementasi database Anda, dan memutakhirkan saat versi baru dirilis.

Untuk informasi selengkapnya, lihat kebijakan pemeliharaan AWS SDKs dan Alat di Panduan Referensi Alat AWS SDKs dan.

Mengirim umpan balik

Kami menyambut umpan balik Anda! Jika Anda memiliki pertanyaan atau komentar, atau masalah yang perlu dilaporkan, silakan gunakan sumber daya berikut.

Jika Anda menemukan potensi kerentanan keamanan di SDK Enkripsi AWS Database, harap beri tahu AWS keamanan. Jangan membuat GitHub masalah publik.

Untuk memberikan umpan balik tentang dokumentasi ini, gunakan tautan umpan balik pada halaman mana pun.

AWS Konsep SDK Enkripsi Database

> Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Topik ini menjelaskan konsep dan terminologi yang digunakan dalam AWS Database Encryption SDK.

Untuk mempelajari bagaimana komponen SDK Enkripsi AWS Database berinteraksi, lihatCara kerja SDK Enkripsi AWS Database.

Untuk mempelajari lebih lanjut tentang SDK Enkripsi AWS Database, lihat topik berikut.

- Pelajari cara SDK Enkripsi AWS Database menggunakan enkripsi amplop untuk melindungi data Anda.
- Pelajari tentang elemen enkripsi amplop: kunci data yang melindungi catatan Anda dan kunci pembungkus yang melindungi kunci data Anda.
- Pelajari tentang gantungan kunci yang menentukan kunci pembungkus yang Anda gunakan.
- Pelajari tentang konteks enkripsi yang menambahkan integritas pada proses enkripsi Anda.
- Pelajari tentang deskripsi materi yang ditambahkan metode enkripsi ke catatan Anda.
- Pelajari tentang tindakan kriptografi yang memberi tahu SDK Enkripsi AWS Database bidang apa yang harus dienkripsi dan ditandatangani.

Topik

- Enkripsi amplop
- Kunci data
- Kunci pembungkus
- Gantungan kunci
- Tindakan kriptografi
- Deskripsi materi
- Konteks enkripsi

- Manajer materi kriptografi
- Enkripsi simetris dan asimetris
- Komitmen utama
- Tanda tangan digital

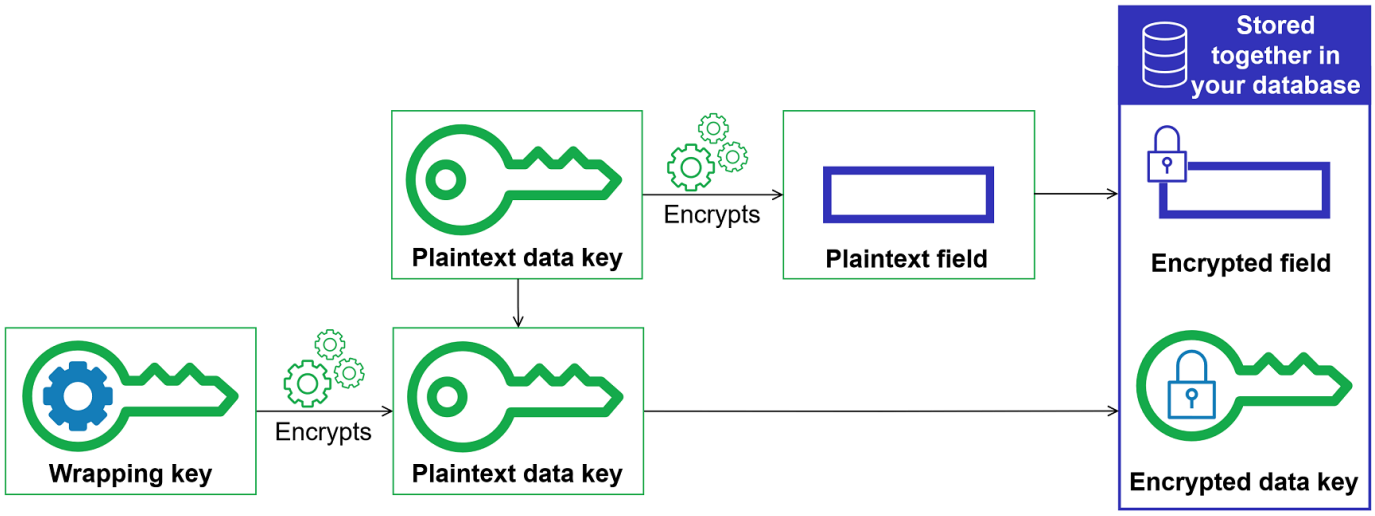
Enkripsi amplop

Keamanan data terenkripsi Anda sebagian bergantung pada perlindungan kunci data yang dapat mendekripsi itu. Salah satu praktik terbaik yang diterima untuk melindungi kunci data adalah mengenkripsinya. Untuk melakukan ini, Anda memerlukan kunci enkripsi lain, yang dikenal sebagai kunci enkripsi kunci atau kunci pembungkus. Praktek menggunakan kunci pembungkus untuk mengenkripsi kunci data dikenal sebagai enkripsi amplop.

Melindungi kunci data

AWS Database Encryption SDK mengenkripsi setiap bidang dengan kunci data yang unik. Kemudian mengenkripsi setiap kunci data di bawah kunci pembungkus yang Anda tentukan. Ini menyimpan kunci data terenkripsi dalam deskripsi materi.

Untuk menentukan kunci pembungkus Anda, Anda menggunakan keyring.



Mengenkripsi data yang sama di bawah beberapa kunci pembungkus

Anda dapat mengenkripsi kunci data dengan beberapa tombol pembungkus. Anda mungkin ingin memberikan kunci pembungkus yang berbeda untuk pengguna yang berbeda, atau kunci pembungkus dari jenis yang berbeda, atau di lokasi yang berbeda. Setiap kunci pembungkus

mengenkripsi kunci data yang sama. AWS Database Encryption SDK menyimpan semua kunci data terenkripsi bersama bidang terenkripsi dalam deskripsi material.

Untuk mendekripsi data, Anda harus menyediakan setidaknya satu kunci pembungkus yang dapat mendekripsi kunci data terenkripsi.

Menggabungkan kekuatan dari beberapa algoritme

Untuk mengenkripsi data Anda, secara default, AWS Database Encryption SDK menggunakan rangkaian algoritme dengan enkripsi simetris AES-GCM, fungsi derivasi kunci berbasis HMAC (HKDF), dan penandatanganan ECDSA. Untuk mengenkripsi kunci data, Anda dapat menentukan algoritma enkripsi simetris atau asimetris yang sesuai dengan kunci pembungkus Anda.

Secara umum, algoritma enkripsi kunci simetris lebih cepat dan menghasilkan ciphertext yang lebih kecil daripada enkripsi kunci asimetris atau publik. Tetapi algoritma kunci publik memberikan pemisahan peran yang melekat. Untuk menggabungkan kekuatan masing-masing, Anda dapat mengenkripsi kunci data dengan enkripsi kunci publik.

Kami merekomendasikan menggunakan salah satu AWS KMS gantungan kunci bila memungkinkan. Saat Anda menggunakan AWS KMS keyring, Anda dapat memilih untuk menggabungkan kekuatan beberapa algoritma dengan menentukan RSA AWS KMS key asimetris sebagai kunci pembungkus Anda. Anda juga dapat menggunakan kunci KMS enkripsi simetris.

Kunci data

Kunci data adalah kunci enkripsi yang digunakan SDK Enkripsi AWS Database untuk mengenkripsi bidang dalam catatan yang ditandai `ENCRYPT_AND_SIGN` dalam tindakan kriptografi. Setiap kunci data adalah array byte yang sesuai dengan persyaratan untuk kunci kriptografi. AWS Database Encryption SDK menggunakan kunci data unik untuk mengenkripsi setiap atribut.

Anda tidak perlu menentukan, menghasilkan, mengimplementasikan, memperluas, melindungi, atau menggunakan kunci data. SDK Enkripsi AWS Database berfungsi untuk Anda saat Anda memanggil operasi enkripsi dan dekripsi.

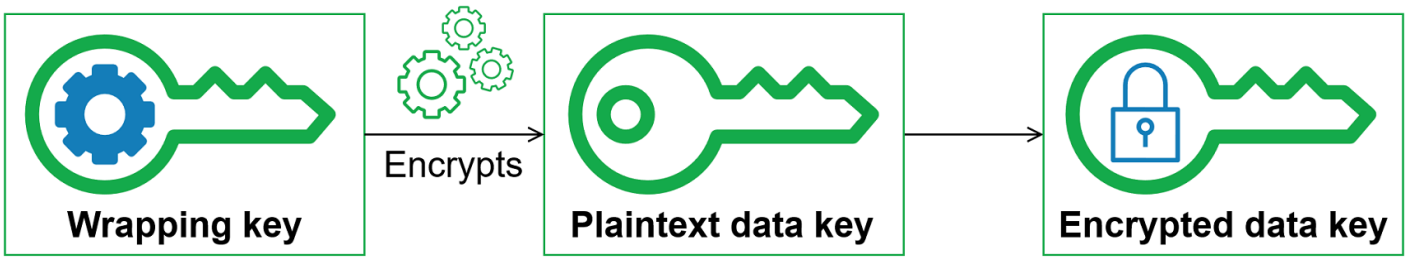
Untuk melindungi kunci data Anda, AWS Database Encryption SDK mengenkripsi mereka di bawah satu atau beberapa kunci enkripsi kunci yang dikenal sebagai kunci pembungkus. Setelah SDK Enkripsi AWS Database menggunakan kunci data teks biasa Anda untuk mengenkripsi data Anda, itu akan menghapusnya dari memori sesegera mungkin. Kemudian menyimpan kunci data terenkripsi dalam deskripsi materi. Untuk detailnya, lihat Cara kerja SDK Enkripsi AWS Database.

> **Tip**
>
> Dalam SDK Enkripsi AWS Database, kami membedakan kunci data dari kunci enkripsi data. Sebagai praktik terbaik, semua suite algoritme yang didukung menggunakan fungsi derivasi kunci. Fungsi derivasi kunci mengambil kunci data sebagai input dan mengembalikan kunci enkripsi data yang sebenarnya digunakan untuk mengenkripsi catatan Anda. Untuk alasan ini, kita sering mengatakan bahwa data dienkripsi "di bawah" kunci data daripada "oleh" kunci data.

Setiap kunci data terenkripsi mencakup metadata, termasuk pengidentifikasi kunci pembungkus yang mengenkripsi itu. Metadata ini memungkinkan SDK Enkripsi AWS Database mengidentifikasi kunci pembungkus yang valid saat mendekripsi.

Kunci pembungkus

Kunci pembungkus adalah kunci enkripsi kunci yang digunakan SDK Enkripsi AWS Database untuk mengenkripsi kunci data yang mengenkripsi catatan Anda. Setiap kunci data dapat dienkripsi di bawah satu atau lebih kunci pembungkus. Anda menentukan kunci pembungkus mana yang digunakan untuk melindungi data Anda saat Anda mengonfigurasi keyring.



AWS Database Encryption SDK mendukung beberapa kunci pembungkus yang umum digunakan, seperti AWS Key Management Service(AWS KMS) kunci KMS enkripsi simetris (termasuk kunci Multi-region) dan kunci KMS RSA asimetris, AWS KMS kunci mentah AES-GCM (Advanced Encryption Standard/Galois Counter Mode), dan kunci RSA mentah. Kami merekomendasikan menggunakan tombol KMS bila memungkinkan. Untuk memutuskan kunci pembungkus mana yang harus Anda gunakan, lihat Memilih kunci pembungkus.

Saat Anda menggunakan enkripsi amplop, Anda perlu melindungi kunci pembungkus Anda dari akses yang tidak sah. Anda dapat melakukan ini dengan salah satu cara berikut:

- Gunakan layanan yang dirancang untuk tujuan ini, seperti AWS Key Management Service (AWS KMS).
- Gunakan modul keamanan perangkat keras (HSM) seperti yang ditawarkan oleh AWS CloudHSM.
- Gunakan alat dan layanan manajemen kunci lainnya.

Jika Anda tidak memiliki sistem manajemen kunci, kami sarankan AWS KMS. SDK Enkripsi AWS Database terintegrasi AWS KMS untuk membantu Anda melindungi dan menggunakan kunci pembungkus Anda.

Gantungan kunci

Untuk menentukan kunci pembungkus yang Anda gunakan untuk enkripsi dan dekripsi, Anda menggunakan keyring. Anda dapat menggunakan keyrings yang disediakan oleh AWS Database Encryption SDK atau mendesain implementasi Anda sendiri.

Sebuah keyring menghasilkan, mengenkripsi, dan mendekripsi kunci data. Ini juga menghasilkan kunci MAC yang digunakan untuk menghitung Kode Otentikasi Pesan Berbasis Hash (HMACs) dalam tanda tangan. Saat Anda menentukan keyring, Anda dapat menentukan kunci pembungkus yang mengenkripsi kunci data Anda. Kebanyakan keyrings menentukan setidaknya satu kunci pembungkus atau layanan yang menyediakan dan melindungi kunci pembungkus. Saat mengenkripsi, AWS Database Encryption SDK menggunakan semua kunci pembungkus yang ditentukan dalam keyring untuk mengenkripsi kunci data. Untuk bantuan dalam memilih dan menggunakan keyrings yang didefinisikan oleh AWS Database Encryption SDK, lihat Menggunakan keyrings.

Tindakan kriptografi

Tindakan kriptografi memberi tahu enkripsi tindakan mana yang harus dilakukan pada setiap bidang dalam catatan.

Nilai tindakan kriptografi dapat berupa salah satu dari yang berikut:

- Enkripsi dan tandatangani — Enkripsi bidang. Sertakan bidang terenkripsi dalam tanda tangan.
- Hanya tanda tangan — Sertakan bidang di tanda tangan.
- Masuk dan sertakan dalam konteks enkripsi — Sertakan bidang dalam konteks tanda tangan dan enkripsi.

Secara default, kunci partisi dan sortir adalah satu-satunya atribut yang disertakan dalam konteks enkripsi. Anda dapat mempertimbangkan untuk mendefinisikan bidang tambahan `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` agar pemasok ID kunci cabang untuk keyring AWS KMS Hierarkis Anda dapat mengidentifikasi kunci cabang mana yang diperlukan untuk dekripsi dari konteks enkripsi. Untuk informasi selengkapnya, lihat pemasok ID kunci cabang.

> **Note**
>
> Untuk menggunakan tindakan `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` kriptografi, Anda harus menggunakan SDK Enkripsi AWS Database versi 3.3 atau yang lebih baru. Terapkan versi baru ke semua pembaca sebelum memperbarui model data Anda untuk disertakan`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

- Jangan melakukan apa-apa — Jangan mengenkripsi atau menyertakan bidang dalam tanda tangan.

Untuk bidang apa pun yang dapat menyimpan data sensitif, gunakan Enkripsi dan tandatangani. Untuk nilai kunci primer (misalnya, kunci partisi dan kunci sortir dalam tabel DynamoDB), gunakan Sign only atau Sign dan sertakan dalam konteks enkripsi. Jika Anda menentukan Tanda dan menyertakan atribut konteks enkripsi, maka atribut partisi dan sortir juga harus Tanda dan sertakan dalam konteks enkripsi. Anda tidak perlu menentukan tindakan kriptografi untuk deskripsi materi. SDK Enkripsi AWS Database secara otomatis menandatangani bidang tempat deskripsi materi disimpan.

Pilih tindakan kriptografi Anda dengan hati-hati. Bila ragu, gunakan Enkripsi dan tanda tangan. Setelah Anda menggunakan SDK Enkripsi AWS Database untuk melindungi catatan Anda, Anda tidak dapat mengubah `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` bidang yang ada `ENCRYPT_AND_SIGN`SIGN_ONLY, atau menjadi`DO_NOTHING`, atau mengubah tindakan kriptografi yang ditetapkan ke bidang yang ada`DO_NOTHING`. Namun, Anda masih dapat membuat perubahan lain pada model data Anda. Misalnya, Anda dapat menambahkan atau menghapus bidang terenkripsi, dalam satu penerapan.

Deskripsi materi

Deskripsi materi berfungsi sebagai header untuk catatan terenkripsi. Saat Anda mengenkripsi dan menandatangani bidang dengan SDK Enkripsi AWS Database, enkripsi mencatat deskripsi materi

saat merakit materi kriptografi dan menyimpan deskripsi materi di bidang baru (`aws_dbe_head`) yang ditambahkan enkripsi ke catatan Anda.

Deskripsi materi adalah struktur data berformat portabel yang berisi salinan terenkripsi dari kunci data dan informasi lainnya, seperti algoritma enkripsi, konteks enkripsi, dan instruksi enkripsi dan penandatanganan. Enkripsi mencatat deskripsi materi saat merakit bahan kriptografi untuk enkripsi dan penandatanganan. Kemudian, ketika perlu merakit bahan kriptografi untuk memverifikasi dan mendekripsi suatu bidang, ia menggunakan deskripsi materi sebagai panduannya.

Menyimpan kunci data terenkripsi di samping bidang terenkripsi merampingkan operasi dekripsi dan membebaskan Anda dari keharusan menyimpan dan mengelola kunci data terenkripsi secara independen dari data yang mereka enkripsi.

Untuk informasi teknis tentang deskripsi materi, lihatFormat deskripsi bahan.

Konteks enkripsi

Untuk meningkatkan keamanan operasi kriptografi Anda, AWS Database Encryption SDK menyertakan konteks enkripsi dalam semua permintaan untuk mengenkripsi dan menandatangani catatan.

Konteks enkripsi adalah sekumpulan pasangan nama-nilai yang berisi data otentikasi tambahan non-rahasia yang sewenang-wenang. AWS Database Encryption SDK menyertakan nama logis untuk database Anda dan nilai kunci primer (misalnya, kunci partisi dan kunci sortir dalam tabel DynamoDB) dalam konteks enkripsi. Saat Anda mengenkripsi dan menandatangani bidang, konteks enkripsi terikat secara kriptografis ke catatan terenkripsi sehingga konteks enkripsi yang sama diperlukan untuk mendekripsi bidang tersebut.

Jika Anda menggunakan AWS KMS keyring, SDK Enkripsi AWS Database juga menggunakan konteks enkripsi untuk menyediakan data terautentikasi tambahan (AAD) dalam panggilan yang dilakukan oleh keyring. AWS KMS

Setiap kali Anda menggunakan rangkaian algoritme default, manajer bahan kriptografi (CMM) menambahkan pasangan nama-nilai ke konteks enkripsi yang terdiri dari nama yang dicadangkan`aws-crypto-public-key`, dan nilai yang mewakili kunci verifikasi publik. Kunci verifikasi publik disimpan dalam deskripsi materi.

Manajer materi kriptografi

Manajer bahan kriptografi (CMM) merakit materi kriptografi yang digunakan untuk mengenkripsi, mendekripsi, dan menandatangani data Anda. Setiap kali Anda menggunakan rangkaian algoritme default, materi kriptografi mencakup teks biasa dan kunci data terenkripsi, kunci penandatanganan simetris, dan kunci penandatanganan asimetris. Anda tidak pernah berinteraksi dengan CMM secara langsung. Metode enkripsi dan dekripsi menanganinya untuk Anda.

Karena CMM bertindak sebagai penghubung antara SDK Enkripsi AWS Database dan keyring, ini adalah titik ideal untuk penyesuaian dan ekstensi, seperti dukungan untuk penegakan kebijakan. Anda dapat secara eksplisit menentukan CMM, tetapi itu tidak diperlukan. Saat Anda menentukan keyring, AWS Database Encryption SDK akan membuat CMM default untuk Anda. CMM default mendapatkan materi enkripsi atau dekripsi dari keyring yang Anda tentukan. Ini mungkin melibatkan panggilan ke layanan kriptografi, seperti AWS Key Management Service(AWS KMS).

Enkripsi simetris dan asimetris

Enkripsi simetris menggunakan kunci yang sama untuk mengenkripsi dan mendekripsi data.

Enkripsi asimetris menggunakan data key pair yang terkait secara matematis. Satu kunci dalam pasangan mengenkripsi data; hanya kunci lain dalam pasangan yang dapat mendekripsi data.

SDK Enkripsi AWS Database menggunakan enkripsi amplop. Ini mengenkripsi data Anda dengan kunci data simetris. Ini mengenkripsi kunci data simetris dengan satu atau lebih tombol pembungkus simetris atau asimetris. Ini menambahkan deskripsi material ke catatan yang mencakup setidaknya satu salinan kunci data terenkripsi.

Mengenkripsi data Anda (enkripsi simetris)

> Untuk mengenkripsi data Anda, AWS Database Encryption SDK menggunakan kunci data simetris dan rangkaian algoritma yang menyertakan algoritma enkripsi simetris. Untuk mendekripsi data, AWS Database Encryption SDK menggunakan kunci data yang sama dan rangkaian algoritma yang sama.

Mengenkripsi kunci data Anda (enkripsi simetris atau asimetris)

> Keyring yang Anda berikan ke operasi enkripsi dan dekripsi menentukan bagaimana kunci data simetris dienkripsi dan didekripsi. Anda dapat memilih keyring yang menggunakan enkripsi simetris, seperti AWS KMS keyring dengan kunci KMS enkripsi simetris, atau yang menggunakan enkripsi asimetris, seperti AWS KMS keyring dengan kunci KMS RSA asimetris.

Komitmen utama

AWS Database Encryption SDK mendukung komitmen utama (kadang-kadang dikenal sebagai ketahanan), properti keamanan yang memastikan bahwa setiap ciphertext dapat didekripsi hanya untuk satu plaintext. Untuk melakukan ini, komitmen utama memastikan bahwa hanya kunci data yang mengenkripsi catatan Anda yang akan digunakan untuk mendekripsi itu. SDK Enkripsi AWS Database mencakup komitmen utama untuk semua operasi enkripsi dan dekripsi.

Sebagian besar cipher simetris modern (termasuk AES) mengenkripsi plaintext di bawah satu kunci rahasia, seperti kunci data unik yang digunakan SDK Enkripsi AWS Database untuk mengenkripsi setiap bidang teks biasa yang ditandai dalam catatan. `ENCRYPT_AND_SIGN` Mendekripsi catatan ini dengan kunci data yang sama mengembalikan plaintext yang identik dengan aslinya. Mendekripsi dengan kunci yang berbeda biasanya akan gagal. Meskipun sulit, secara teknis dimungkinkan untuk mendekripsi ciphertext di bawah dua kunci yang berbeda. Dalam kasus yang jarang terjadi, adalah layak untuk menemukan kunci yang sebagian dapat mendekripsi ciphertext menjadi teks biasa yang berbeda, tetapi masih dapat dipahami.

AWS Database Encryption SDK selalu mengenkripsi setiap atribut di bawah satu kunci data unik. Mungkin mengenkripsi kunci data itu di bawah beberapa kunci pembungkus, tetapi kunci pembungkus selalu mengenkripsi kunci data yang sama. Meskipun demikian, catatan terenkripsi yang canggih dan dibuat secara manual mungkin sebenarnya berisi kunci data yang berbeda, masing-masing dienkripsi oleh kunci pembungkus yang berbeda. Misalnya, jika satu pengguna mendekripsi catatan terenkripsi, ia mengembalikan 0x0 (false) sementara pengguna lain yang mendekripsi catatan terenkripsi yang sama mendapat 0x1 (true).

Untuk mencegah skenario ini, AWS Database Encryption SDK menyertakan komitmen utama saat mengenkripsi dan mendekripsi. Metode enkripsi secara kriptografis mengikat kunci data unik yang menghasilkan ciphertext ke komitmen kunci, Hash Based Message Authentication Code (HMAC) dihitung atas deskripsi material menggunakan derivasi dari kunci data. Kemudian menyimpan komitmen utama dalam deskripsi materi. Saat mendekripsi catatan dengan komitmen utama, SDK Enkripsi AWS Database memverifikasi bahwa kunci data adalah satu-satunya kunci untuk catatan terenkripsi tersebut. Jika verifikasi kunci data gagal, operasi dekripsi gagal.

Tanda tangan digital

SDK Enkripsi AWS Database mengenkripsi data Anda menggunakan algoritma enkripsi yang diautentikasi, AES-GCM, dan proses dekripsi memverifikasi integritas dan keaslian pesan terenkripsi tanpa menggunakan tanda tangan digital. Tetapi karena AES-GCM menggunakan kunci simetris, siapa pun yang dapat mendekripsi kunci data yang digunakan untuk mendekripsi ciphertext juga

dapat secara manual membuat ciphertext terenkripsi baru, yang menyebabkan masalah keamanan potensial. Misalnya, jika Anda menggunakan AWS KMS key sebagai kunci pembungkus, pengguna dengan `kms:Decrypt` izin dapat membuat ciphertext terenkripsi tanpa menelepon. `kms:Encrypt`

Untuk menghindari masalah ini, rangkaian algoritme default menambahkan tanda tangan Elliptic Curve Digital Signature Algorithm (ECDSA) ke catatan terenkripsi. Rangkaian algoritme default mengenkripsi bidang dalam catatan Anda yang ditandai `ENCRYPT_AND_SIGN` menggunakan algoritma enkripsi yang diautentikasi, AES-GCM. Kemudian, ia menghitung Kode Otentikasi Pesan Berbasis Hash (HMACs) dan tanda tangan ECDSA asimetris atas bidang dalam catatan Anda yang ditandai,, dan. `ENCRYPT_AND_SIGN` `SIGN_ONLY` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Proses dekripsi menggunakan tanda tangan untuk memverifikasi bahwa pengguna yang berwenang mengenkripsi catatan.

Ketika rangkaian algoritme default digunakan, SDK Enkripsi AWS Database menghasilkan kunci pribadi sementara dan public key pair untuk setiap record terenkripsi. SDK Enkripsi AWS Database menyimpan kunci publik dalam deskripsi materi dan membuang kunci pribadi. Ini memastikan bahwa tidak ada yang dapat membuat tanda tangan lain yang memverifikasi dengan kunci publik. Algoritma mengikat kunci publik ke kunci data terenkripsi sebagai data otentikasi tambahan dalam deskripsi materi, mencegah pengguna yang hanya dapat mendekripsi bidang dari mengubah kunci publik atau memengaruhi verifikasi tanda tangan.

SDK Enkripsi AWS Database selalu menyertakan verifikasi HMAC. Tanda tangan digital ECDSA diaktifkan secara default, tetapi tidak diperlukan. Jika pengguna yang mengenkripsi data dan pengguna yang mendekripsi data sama-sama dipercaya, Anda dapat mempertimbangkan untuk menggunakan rangkaian algoritme yang tidak menyertakan tanda tangan digital untuk meningkatkan kinerja Anda. Untuk informasi selengkapnya tentang memilih rangkaian algoritme alternatif, lihat Memilih rangkaian algoritma.

> **Note**
>
> Jika keyring tidak menggambarkan antara enkripsi dan dekripsi, tanda tangan digital tidak memberikan nilai kriptografi.

AWS KMS keyrings, termasuk AWS KMS keyring RSA asimetris, dapat menggambarkan antara enkripsi dan dekripsi berdasarkan kebijakan utama dan kebijakan IAM. AWS KMS

Karena sifat kriptografinya, gantungan kunci berikut tidak dapat menggambarkan antara enkripsi dan dekripsi:

- AWS KMS Gantungan kunci hierarkis
- AWS KMS Gantungan kunci ECDH
- Gantungan kunci AES mentah
- Gantungan kunci RSA mentah
- Gantungan kunci ECDH mentah

Cara kerja SDK Enkripsi AWS Database

> Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

AWS Database Encryption SDK menyediakan pustaka enkripsi sisi klien yang dirancang khusus untuk melindungi data yang Anda simpan dalam database. Pustakanya meliputi implementasi aman yang dapat Anda perluas atau gunakan tanpa perubahan. Untuk informasi selengkapnya tentang mendefinisikan dan menggunakan komponen kustom, lihat GitHub repositori untuk implementasi database Anda.

Alur kerja di bagian ini menjelaskan cara SDK Enkripsi AWS Database mengenkripsi dan menandatangani serta mendekripsi serta memverifikasi data dalam database Anda. Alur kerja ini menjelaskan proses dasar menggunakan elemen abstrak dan fitur default. Untuk detail tentang cara kerja SDK Enkripsi AWS Database dengan implementasi database Anda, lihat topik Apa yang dienkripsi untuk database Anda.

AWS Database Encryption SDK menggunakan enkripsi amplop untuk melindungi data Anda. Setiap catatan dienkripsi di bawah kunci data unik. Kunci data digunakan untuk mendapatkan kunci enkripsi data unik untuk setiap bidang yang ditandai `ENCRYPT_AND_SIGN` dalam tindakan kriptografi Anda. Kemudian, salinan kunci data dienkripsi oleh kunci pembungkus yang Anda tentukan. Untuk mendekripsi catatan terenkripsi, AWS Database Encryption SDK menggunakan kunci pembungkus yang Anda tentukan untuk mendekripsi setidaknya satu kunci data terenkripsi. Kemudian dapat mendekripsi ciphertext dan mengembalikan entri plaintext.

Untuk informasi selengkapnya tentang istilah yang digunakan dalam SDK Enkripsi AWS Database, lihatAWS Konsep SDK Enkripsi Database.

Enkripsi dan tandatangani

Pada intinya, AWS Database Encryption SDK adalah enkripsi rekaman yang mengenkripsi, menandatangani, memverifikasi, dan mendekripsi catatan dalam database Anda. Dibutuhkan informasi tentang catatan dan instruksi Anda tentang bidang mana yang akan dienkripsi dan ditandatangani. Ini mendapatkan materi enkripsi, dan instruksi tentang cara menggunakannya, dari manajer bahan kriptografi yang dikonfigurasi dari kunci pembungkus yang Anda tentukan.

Panduan berikut menjelaskan cara SDK Enkripsi AWS Database mengenkripsi dan menandatangani entri data Anda.

1. Manajer bahan kriptografi menyediakan SDK Enkripsi AWS Database dengan kunci enkripsi data unik: satu kunci data teks biasa, salinan kunci data yang dienkripsi oleh kunci pembungkus yang ditentukan, dan kunci MAC.

   > **Note**
   >
   > Anda dapat mengenkripsi kunci data di bawah beberapa kunci pembungkus. Masing-masing kunci pembungkus mengenkripsi salinan terpisah dari kunci data. AWS Database Encryption SDK menyimpan semua kunci data terenkripsi dalam deskripsi material. AWS Database Encryption SDK menambahkan field baru (`aws_dbe_head`) ke record yang menyimpan deskripsi material.
   >
   > Kunci MAC diturunkan untuk setiap salinan kunci data yang dienkripsi. Tombol MAC tidak disimpan dalam deskripsi materi. Sebagai gantinya, metode dekripsi menggunakan kunci pembungkus untuk mendapatkan kunci MAC lagi.

2. Metode enkripsi mengenkripsi setiap bidang yang ditandai seperti `ENCRYPT_AND_SIGN` dalam tindakan kriptografi yang Anda tentukan.
3. Metode enkripsi berasal `commitKey` dari kunci data dan menggunakannya untuk menghasilkan nilai komitmen kunci, dan kemudian membuang kunci data.
4. Metode enkripsi menambahkan deskripsi material ke catatan. Deskripsi materi berisi kunci data terenkripsi dan informasi lain tentang catatan terenkripsi. Untuk daftar lengkap informasi yang disertakan dalam deskripsi materi, lihat Format deskripsi materi.
5. Metode enkripsi menggunakan kunci MAC yang dikembalikan pada Langkah 1 untuk menghitung nilai Kode Otentikasi Pesan Berbasis Hash (HMAC) di atas kanonikalisasi deskripsi material, konteks enkripsi, dan setiap bidang yang ditandai `ENCRYPT_AND_SIGNSIGN_ONLY`,

atau dalam tindakan kriptografi. `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Nilai HMAC disimpan dalam bidang baru (`aws_dbe_foot`) yang ditambahkan metode enkripsi ke catatan.

6. Metode enkripsi menghitung tanda tangan ECDSA atas kanonikalisasi deskripsi material, konteks enkripsi, dan setiap bidang yang ditandai `ENCRYPT_AND_SIGNSIGN_ONLY`, atau `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` dan menyimpan tanda tangan ECDSA di lapangan. `aws_dbe_foot`

   > **Note**
   >
   > Tanda tangan ECDSA diaktifkan secara default, tetapi tidak diperlukan.

7. Metode enkripsi menyimpan catatan terenkripsi dan ditandatangani dalam database Anda

Dekripsi dan verifikasi

1. Manajer bahan kriptografi (CMM) menyediakan metode dekripsi dengan bahan dekripsi yang disimpan dalam deskripsi materi, termasuk kunci data teks biasa dan kunci MAC terkait.

   - CMM mendekripsi kunci data terenkripsi dengan kunci pembungkus di keyring yang ditentukan dan mengembalikan kunci data plaintext.

2. Metode dekripsi membandingkan dan memverifikasi nilai komitmen utama dalam deskripsi material.
3. Metode dekripsi memverifikasi tanda tangan di bidang tanda tangan.

   Ini mengidentifikasi bidang mana yang ditandai`ENCRYPT_AND_SIGN`,`SIGN_ONLY`, atau `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` dari daftar bidang tidak diautentikasi yang diizinkan yang Anda tetapkan. Metode dekripsi menggunakan kunci MAC yang dikembalikan pada Langkah 1 untuk menghitung ulang dan membandingkan nilai HMAC untuk bidang yang ditandai,, atau. `ENCRYPT_AND_SIGN` `SIGN_ONLY` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Kemudian, memverifikasi tanda tangan ECDSA menggunakan kunci publik yang disimpan dalam konteks enkripsi.

4. Metode dekripsi menggunakan kunci data plaintext untuk mendekripsi setiap nilai yang ditandai. `ENCRYPT_AND_SIGN` AWS Database Encryption SDK kemudian membuang kunci data plaintext.
5. Metode dekripsi mengembalikan catatan plaintext.

Rangkaian algoritme yang didukung di SDK Enkripsi AWS Database

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Sebuah algoritma suite adalah kumpulan algoritma kriptografi dan nilai-nilai terkait. Sistem kriptografi menggunakan implementasi algoritma untuk menghasilkan ciphertext.

SDK Enkripsi AWS Database menggunakan rangkaian algoritme untuk mengenkripsi dan menandatangani bidang di database Anda. Semua suite algoritma yang didukung menggunakan algoritma Advanced Encryption Standard (AES) dengan Galois/Counter Mode (GCM), yang dikenal sebagai AES-GCM, untuk mengenkripsi data mentah. AWS Database Encryption SDK mendukung kunci enkripsi 256-bit. Panjang tag otentikasi selalu 16 byte.

AWS Enkripsi Database SDK Algoritma Suites

Algoritme	Enkripsi algoritme	Panjang kunci data (dalam bit)	Algoritma derivasi kunci	Algoritma tanda tangan simetris	Algoritma tanda tangan asimetris	Komitmen utama
Default	AES-GCM	256	HKDF dengan SHA-512	HMAC-SHA-384	ECDSA dengan P-384 dan SHA-384	HKDF dengan SHA-512
AES-GCM tanpa tanda tangan digital ECDSA	AES-GCM	256	HKDF dengan SHA-512	HMAC-SHA-384	Tidak ada	HKDF dengan SHA-512

Enkripsi algoritme

Nama dan mode algoritma enkripsi yang digunakan. Rangkaian algoritma dalam SDK Enkripsi AWS Database menggunakan algoritma Advanced Encryption Standard (AES) dengan Galois/Counter Mode (GCM).

Panjang kunci data

Panjang kunci data dalam bit. AWS Database Encryption SDK mendukung kunci data 256-bit. Kunci data digunakan sebagai masukan ke fungsi derivasi extract-and-expand kunci berbasis HMAC (HKDF). Output dari HKDF digunakan sebagai kunci enkripsi data dalam algoritma enkripsi.

Algoritma derivasi kunci

Fungsi derivasi extract-and-expand kunci berbasis HMAC (HKDF) digunakan untuk menurunkan kunci enkripsi data. AWS Database Encryption SDK menggunakan HKDF yang didefinisikan dalam RFC 5869.

- Fungsi hash yang digunakan adalah SHA-512
- Untuk langkah ekstrak:
  - Tidak ada garam yang digunakan. Per RFC, garam diatur ke string nol.
  - Bahan kunci input adalah kunci data dari keyring.
- Untuk langkah perluasan:
  - Kunci pseudorandom input adalah output dari langkah ekstrak.
  - Label kuncinya adalah byte yang dikodekan UTF-8 dari `DERIVEKEY` string dalam urutan byte endian besar.
  - Info input adalah gabungan dari ID algoritma dan label kunci (dalam urutan itu).
  - Panjang bahan kunci keluaran adalah panjang kunci Data. Output ini digunakan sebagai kunci enkripsi data dalam algoritma enkripsi.

Algoritma tanda tangan simetris

Algoritma Hash Based Message Authentication Code (HMAC) digunakan untuk menghasilkan tanda tangan simetris. Semua suite algoritma yang didukung termasuk verifikasi HMAC.

AWS Database Encryption SDK membuat serialisasi deskripsi material dan semua bidang yang ditandai`ENCRYPT_AND_SIGN`,`SIGN_ONLY`, atau. `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Kemudian, ia menggunakan HMAC dengan algoritma fungsi hash kriptografi (SHA-384) untuk menandatangani kanonikalisasi.

Tanda tangan HMAC simetris disimpan di bidang baru (`aws_dbe_foot`) yang ditambahkan SDK Enkripsi AWS Database ke catatan.

Algoritma tanda tangan asimetris

Algoritma tanda tangan digunakan untuk menghasilkan tanda tangan digital asimetris.

AWS Database Encryption SDK membuat serialisasi deskripsi material dan semua bidang yang ditandai`ENCRYPT_AND_SIGN`,`SIGN_ONLY`, atau. `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Kemudian, ia menggunakan Elliptic Curve Digital Signature Algorithm (ECDSA) dengan spesifikasi berikut untuk menandatangani kanonikalisasi:

- Kurva elips yang digunakan adalah P-384, sebagaimana didefinisikan dalam Digital Signature Standard (DSS) (FIPS PUB 186-4).
- Fungsi hash yang digunakan adalah SHA-384.

Tanda tangan ECDSA asimetris disimpan dengan tanda tangan HMAC simetris di lapangan. `aws_dbe_foot`

Tanda tangan digital ECDSA disertakan secara default, tetapi tidak diperlukan.

Komitmen utama

Fungsi derivasi extract-and-expand kunci berbasis HMAC (HKDF) digunakan untuk menurunkan kunci komit.

- Fungsi hash yang digunakan adalah SHA-512
- Untuk langkah ekstrak:
  - Tidak ada garam yang digunakan. Per RFC, garam diatur ke string nol.
  - Bahan kunci input adalah kunci data dari keyring.
- Untuk langkah perluasan:
  - Kunci pseudorandom input adalah output dari langkah ekstrak.
  - Info masukan adalah byte yang dikodekan UTF-8 dari `COMMITKEY` string dalam urutan byte endian besar.
  - Panjang bahan kunci keluaran adalah 256 bit. Output ini digunakan sebagai kunci komit.

Kunci komit menghitung komitmen rekaman, hash 256-bit Hash Based Message Authentication Code (HMAC) yang berbeda, di atas deskripsi materi. Untuk penjelasan teknis tentang menambahkan komitmen utama ke rangkaian algoritme, lihat Key Committing AEADs in Cryptology ePrint Archive.

Rangkaian algoritme default

Secara default, AWS Database Encryption SDK menggunakan rangkaian algoritma dengan AES-GCM, fungsi derivasi extract-and-expand kunci berbasis HMAC (HKDF), verifikasi HMAC, tanda tangan digital ECDSA, komitmen kunci, dan kunci enkripsi 256-bit.

Rangkaian algoritme default mencakup verifikasi HMAC (tanda tangan simetris) dan tanda tangan digital ECDSA (tanda tangan asimetris). Tanda tangan ini disimpan di bidang baru (`aws_dbe_foot`) yang ditambahkan SDK Enkripsi AWS Database ke catatan. Tanda tangan digital ECDSA sangat berguna ketika kebijakan otorisasi memungkinkan satu set pengguna untuk mengenkripsi data dan satu set pengguna yang berbeda untuk mendekripsi data.

Rangkaian algoritme default juga memperoleh komitmen utama — hash HMAC yang mengikat kunci data ke catatan. Nilai komitmen utama adalah HMAC yang dihitung dari deskripsi material dan kunci komit. Nilai komitmen utama kemudian disimpan dalam deskripsi materi. Komitmen utama memastikan bahwa setiap ciphertext hanya mendekripsi menjadi satu teks biasa. Mereka melakukan ini dengan memvalidasi kunci data yang digunakan sebagai input ke algoritma enkripsi. Saat mengenkripsi, rangkaian algoritma memperoleh komitmen utama HMAC. Sebelum mendekripsi, mereka memvalidasi bahwa kunci data menghasilkan komitmen kunci yang sama HMAC. Jika tidak, panggilan dekripsi gagal.

AES-GCM tanpa tanda tangan digital ECDSA

Meskipun rangkaian algoritme default kemungkinan cocok untuk sebagian besar aplikasi, Anda dapat memilih rangkaian algoritme alternatif. Misalnya, beberapa model kepercayaan akan dipenuhi oleh rangkaian algoritma tanpa tanda tangan digital ECDSA. Gunakan suite ini hanya ketika pengguna yang mengenkripsi data dan pengguna yang mendekripsi data sama-sama dipercaya.

Semua rangkaian algoritma SDK Enkripsi AWS Database menyertakan verifikasi HMAC (tanda tangan simetris). Satu-satunya perbedaan, adalah bahwa rangkaian algoritma AES-GCM tanpa tanda tangan digital ECDSA tidak memiliki tanda tangan asimetris yang memberikan lapisan keaslian dan non-penolakan tambahan.

Misalnya, jika Anda memiliki beberapa kunci pembungkus di keyring,,, dan `wrappingKeyA` `wrappingKeyBwrappingKeyC`, dan Anda mendekripsi rekaman menggunakan`wrappingKeyA`, tanda tangan simetris HMAC memverifikasi bahwa catatan dienkripsi oleh pengguna dengan akses ke. `wrappingKeyA` Jika Anda menggunakan rangkaian algoritme default, HMACs berikan verifikasi yang sama`wrappingKeyA`, dan juga menggunakan tanda tangan digital ECDSA untuk memastikan catatan dienkripsi oleh pengguna dengan izin enkripsi untuk. `wrappingKeyA`

Untuk memilih rangkaian algoritma AES-GCM tanpa tanda tangan digital, sertakan cuplikan berikut dalam konfigurasi enkripsi Anda.

Java

Cuplikan berikut menentukan rangkaian algoritma AES-GCM tanpa tanda tangan digital ECDSA. Untuk informasi selengkapnya, lihat the section called “Konfigurasi enkripsi”.

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

C# / .NET

Cuplikan berikut menentukan rangkaian algoritma AES-GCM tanpa tanda tangan digital ECDSA. Untuk informasi selengkapnya, lihat the section called “Konfigurasi enkripsi”.

```
AlgorithmSuiteId =
 DBEAlgorithmSuiteId.ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY_SYMSIG_HMAC_SHA384
```

Rust

Cuplikan berikut menentukan rangkaian algoritma AES-GCM tanpa tanda tangan digital ECDSA. Untuk informasi selengkapnya, lihat the section called “Konfigurasi enkripsi”.

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

Menggunakan SDK Enkripsi AWS Database dengan AWS KMS

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Untuk menggunakan AWS Database Encryption SDK, Anda perlu mengkonfigurasi keyring dan menentukan satu atau beberapa kunci pembungkus. Jika Anda tidak memiliki infrastruktur utama, sebaiknya gunakan AWS Key Management Service (AWS KMS).

AWS Database Encryption SDK mendukung dua jenis AWS KMS keyrings. AWS KMS Keyring tradisional digunakan AWS KMS keysuntuk menghasilkan, mengenkripsi, dan mendekripsi kunci data. Anda dapat menggunakan kunci simetris enkripsi (`SYMMETRIC_DEFAULT`) atau asimetris RSA KMS. Karena SDK Enkripsi AWS Database mengenkripsi dan menandatangani setiap catatan dengan kunci data unik, AWS KMS keyring harus memanggil AWS KMS setiap operasi enkripsi dan dekripsi. Untuk aplikasi yang perlu meminimalkan jumlah panggilan ke AWS KMS, AWS Database Encryption SDK juga mendukung keyring AWS KMS Hierarchical. Hierarchical keyring adalah solusi caching materi kriptografi yang mengurangi jumlah AWS KMS panggilan dengan menggunakan kunci cabang yang AWS KMS dilindungi yang disimpan dalam tabel Amazon DynamoDB, dan kemudian secara lokal menyimpan materi kunci cabang yang digunakan dalam operasi enkripsi dan dekripsi. Kami merekomendasikan menggunakan AWS KMS gantungan kunci bila memungkinkan.

Untuk berinteraksi dengan AWS KMS, AWS Database Encryption SDK memerlukan AWS KMS modul. AWS SDK untuk Java

Untuk mempersiapkan untuk menggunakan AWS Database Encryption SDK dengan AWS KMS

1. Buat sebuah Akun AWS. Untuk mempelajari caranya, lihat Bagaimana cara membuat dan mengaktifkan akun Amazon Web Services baru? di pusat AWS pengetahuan.
2. Buat enkripsi AWS KMS key simetris. Untuk bantuan, lihat Membuat Kunci di Panduan AWS Key Management Service Pengembang.

   > **Tip**
   >
   > Untuk menggunakan AWS KMS key pemrograman, Anda memerlukan Nama Sumber Daya Amazon (ARN) dari file. AWS KMS keyUntuk bantuan menemukan

ARN dari sebuah AWS KMS key, lihat Menemukan ID Kunci dan ARN di Panduan Pengembang.AWS Key Management Service

3. Hasilkan ID kunci akses dan kunci akses keamanan. Anda dapat menggunakan ID kunci akses dan kunci akses rahasia untuk pengguna IAM atau Anda dapat menggunakan AWS Security Token Service untuk membuat sesi baru dengan kredenal keamanan sementara yang mencakup ID kunci akses, kunci akses rahasia, dan token sesi. Sebagai praktik keamanan terbaik, kami menyarankan Anda menggunakan kredensil sementara alih-alih kredensil jangka panjang yang terkait dengan pengguna IAM atau akun pengguna AWS (root) Anda.

   Untuk membuat pengguna IAM dengan kunci akses, lihat Membuat Pengguna IAM di Panduan Pengguna IAM.

   Untuk menghasilkan kredensil keamanan sementara, lihat Meminta kredensil keamanan sementara di Panduan Pengguna IAM.

4. Tetapkan AWS kredensil Anda menggunakan instruksi di AWS SDK untuk Javadan ID kunci akses dan kunci akses rahasia yang Anda buat di langkah 3. Jika Anda membuat kredensil sementara, Anda juga perlu menentukan token sesi.

   Prosedur ini memungkinkan AWS SDKs untuk menandatangani permintaan AWS untuk Anda. Contoh kode dalam SDK Enkripsi AWS Database yang berinteraksi dengan AWS KMS asumsi bahwa Anda telah menyelesaikan langkah ini.

Mengkonfigurasi SDK Enkripsi AWS Database

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

AWS Database Encryption SDK dirancang agar mudah digunakan. Meskipun SDK Enkripsi AWS Database memiliki beberapa opsi konfigurasi, nilai default dipilih dengan cermat agar praktis dan aman untuk sebagian besar aplikasi. Namun, Anda mungkin perlu menyesuaikan konfigurasi untuk meningkatkan kinerja atau menyertakan fitur khusus dalam desain Anda.

Topik

- Memilih bahasa pemrograman
- Memilih tombol pembungkus
- Membuat filter penemuan
- Bekerja dengan basis data multitenant
- Membuat suar yang ditandatangani

Memilih bahasa pemrograman

AWS Database Encryption SDK untuk DynamoDB tersedia dalam beberapa bahasa pemrograman. Implementasi bahasa dirancang untuk sepenuhnya dapat dioperasikan dan menawarkan fitur yang sama, meskipun mereka mungkin diimplementasikan dengan cara yang berbeda. Biasanya, Anda menggunakan perpustakaan yang kompatibel dengan aplikasi Anda.

Memilih tombol pembungkus

AWS Database Encryption SDK menghasilkan kunci data simetris unik untuk mengenkripsi setiap bidang. Anda tidak perlu mengkonfigurasi, mengelola, atau menggunakan kunci data. AWS Database Encryption SDK melakukannya untuk Anda.

Namun, Anda harus memilih satu atau lebih kunci pembungkus untuk mengenkripsi setiap kunci data. AWS Database Encryption SDK mendukung AWS Key Management Service(AWS KMS) kunci KMS enkripsi simetris dan kunci KMS RSA asimetris. Ini juga mendukung tombol simetris AES dan tombol

asimetris RSA yang Anda berikan dalam berbagai ukuran. Anda bertanggung jawab atas keamanan dan daya tahan kunci pembungkus Anda, jadi kami sarankan Anda menggunakan kunci enkripsi dalam modul keamanan perangkat keras atau layanan infrastruktur utama, seperti AWS KMS.

Untuk menentukan kunci pembungkus Anda untuk enkripsi dan dekripsi, Anda menggunakan keyring. Bergantung pada jenis keyring yang Anda gunakan, Anda dapat menentukan satu kunci pembungkus atau beberapa kunci pembungkus dari jenis yang sama atau berbeda. Jika Anda menggunakan beberapa kunci pembungkus untuk membungkus kunci data, setiap kunci pembungkus akan mengenkripsi salinan kunci data yang sama. Kunci data terenkripsi (satu per kunci pembungkus) disimpan dalam deskripsi material yang disimpan di samping bidang terenkripsi. Untuk mendekripsi data, SDK Enkripsi AWS Database harus terlebih dahulu menggunakan salah satu kunci pembungkus Anda untuk mendekripsi kunci data terenkripsi.

Kami merekomendasikan agar menggunakan salah satu AWS KMS gantungan kunci jika memungkinkan. AWS Database Encryption SDK menyediakan AWS KMS keyring dan keyring AWS KMS Hierarkis, yang mengurangi jumlah panggilan yang dilakukan. AWS KMS Untuk menentukan AWS KMS key dalam keyring, gunakan pengenal AWS KMS kunci yang didukung. Jika Anda menggunakan keyring AWS KMS Hierarkis, Anda harus menentukan kunci ARN. Untuk detail tentang pengidentifikasi kunci untuk AWS KMS kunci, lihat Pengidentifikasi Kunci di Panduan AWS Key Management Service Pengembang.

- Saat Anda mengenkripsi dengan AWS KMS keyring, Anda dapat menentukan pengenal kunci yang valid (ARN kunci, nama alias, alias ARN, atau ID kunci) untuk kunci KMS enkripsi simetris. Jika Anda menggunakan kunci KMS RSA asimetris, Anda harus menentukan kunci ARN.

  Jika Anda menentukan nama alias atau alias ARN untuk kunci KMS saat mengenkripsi, SDK Enkripsi Database menyimpan ARN kunci yang saat ini terkait dengan alias tersebut; itu tidak menyimpan alias. AWS Perubahan pada alias tidak memengaruhi kunci KMS yang digunakan untuk mendekripsi kunci data Anda.
- Secara default, AWS KMS keyring mendekripsi catatan dalam mode ketat (di mana Anda menentukan kunci KMS tertentu). Anda harus menggunakan ARN kunci AWS KMS keys untuk mengidentifikasi dekripsi.

  Saat Anda mengenkripsi dengan AWS KMS keyring, SDK Enkripsi AWS Database menyimpan ARN kunci AWS KMS key dalam deskripsi materi dengan kunci data terenkripsi. Saat mendekripsi dalam mode ketat, SDK Enkripsi AWS Database memverifikasi bahwa ARN kunci yang sama muncul di keyring sebelum mencoba menggunakan kunci pembungkus untuk mendekripsi kunci data terenkripsi. Jika Anda menggunakan pengenal kunci yang berbeda, SDK Enkripsi AWS

Database tidak akan mengenali atau menggunakan AWS KMS key, bahkan jika pengidentifikasi merujuk ke kunci yang sama.

- Saat mendekripsi dalam mode penemuan, Anda tidak menentukan kunci pembungkus apa pun. Pertama, SDK Enkripsi AWS Database mencoba mendekripsi catatan dengan ARN kunci yang disimpan dalam deskripsi materi. Jika itu tidak berhasil, SDK Enkripsi AWS Database meminta AWS KMS untuk mendekripsi catatan menggunakan kunci KMS yang mengenkripsi itu, terlepas dari siapa yang memiliki atau memiliki akses ke kunci KMS itu.

Untuk menentukan kunci AES mentah atau key pair RSA mentah sebagai kunci pembungkus dalam keyring, Anda harus menentukan namespace dan nama. Saat mendekripsi, Anda harus menggunakan namespace dan nama yang sama persis untuk setiap kunci pembungkus mentah seperti yang Anda gunakan saat mengenkripsi. Jika Anda menggunakan namespace atau nama yang berbeda, SDK Enkripsi AWS Database tidak akan mengenali atau menggunakan kunci pembungkus, meskipun materi kuncinya sama.

Membuat filter penemuan

Saat mendekripsi data yang dienkripsi dengan kunci KMS, ini adalah praktik terbaik untuk mendekripsi dalam mode ketat, yaitu membatasi kunci pembungkus yang digunakan hanya untuk yang Anda tentukan. Namun, jika perlu, Anda juga dapat mendekripsi dalam mode penemuan, di mana Anda tidak menentukan kunci pembungkus apa pun. Dalam mode ini, AWS KMS dapat mendekripsi kunci data terenkripsi menggunakan kunci KMS yang mengenkripsi itu, terlepas dari siapa yang memiliki atau memiliki akses ke kunci KMS itu.

Jika Anda harus mendekripsi dalam mode penemuan, kami sarankan Anda selalu menggunakan filter penemuan, yang membatasi kunci KMS yang dapat digunakan untuk yang ada di partisi dan yang ditentukan Akun AWS . Filter penemuan bersifat opsional, tetapi ini adalah praktik terbaik.

Gunakan tabel berikut untuk menentukan nilai partisi untuk filter penemuan Anda.

Wilayah	Partition
Wilayah AWS	`aws`
Wilayah Tiongkok	`aws-cn`
AWS GovCloud (US) Regions	`aws-us-gov`

Contoh berikut menunjukkan cara membuat filter penemuan. Sebelum menggunakan kode, ganti nilai contoh dengan nilai yang valid untuk Anda Akun AWS dan partisi.

Java

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

C# / .NET

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

Rust

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

Bekerja dengan basis data multitenant

Dengan AWS Database Encryption SDK, Anda dapat mengonfigurasi enkripsi sisi klien untuk database dengan skema bersama dengan mengisolasi setiap penyewa dengan bahan enkripsi yang berbeda. Saat mempertimbangkan database multitenant, luangkan waktu untuk meninjau persyaratan keamanan Anda dan bagaimana multitenansi dapat memengaruhi mereka. Misalnya, menggunakan database multitenant dapat memengaruhi kemampuan Anda untuk menggabungkan SDK Enkripsi AWS Database dengan solusi enkripsi sisi server lainnya.

Jika Anda memiliki beberapa pengguna yang melakukan operasi enkripsi dalam database Anda, Anda dapat menggunakan salah satu AWS KMS keyring untuk menyediakan setiap pengguna dengan kunci yang berbeda untuk digunakan dalam operasi kriptografi mereka. Mengelola kunci

data untuk solusi enkripsi sisi klien multitenant bisa rumit. Kami merekomendasikan untuk mengatur data Anda dengan penyewa bila memungkinkan. Jika penyewa diidentifikasi oleh nilai kunci primer (misalnya, kunci partisi dalam tabel Amazon DynamoDB), maka mengelola kunci Anda lebih mudah.

Anda dapat menggunakan AWS KMS keyring untuk mengisolasi setiap penyewa dengan keyring yang berbeda AWS KMS dan. AWS KMS keys Berdasarkan volume AWS KMS panggilan yang dilakukan per penyewa, Anda mungkin ingin menggunakan keyring AWS KMS Hierarkis untuk meminimalkan panggilan Anda. AWS KMSAWS KMS Hierarchical keyring adalah solusi caching materi kriptografi yang mengurangi jumlah AWS KMS panggilan dengan menggunakan kunci cabang yang AWS KMS dilindungi yang disimpan dalam tabel Amazon DynamoDB, dan kemudian secara lokal menyimpan materi kunci cabang yang digunakan dalam operasi enkripsi dan dekripsi. Anda harus menggunakan keyring AWS KMS Hierarkis untuk mengimplementasikan enkripsi yang dapat dicari di database Anda.

Membuat suar yang ditandatangani

SDK Enkripsi AWS Database menggunakan beacon standar dan suar majemuk untuk menyediakan solusi enkripsi yang dapat dicari yang memungkinkan Anda mencari catatan terenkripsi tanpa mendekripsi seluruh database yang ditanyakan. Namun, SDK Enkripsi AWS Database juga mendukung suar bertanda tangan yang dapat dikonfigurasi sepenuhnya dari bidang bertanda teks biasa. Signed beacon adalah jenis suar majemuk yang mengindeks dan melakukan kueri kompleks pada dan bidang. `SIGN_ONLY` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`

Misalnya, jika Anda memiliki database multitenant, Anda mungkin ingin membuat suar bertanda tangan yang memungkinkan Anda untuk menanyakan database Anda untuk catatan yang dienkripsi oleh kunci penyewa tertentu. Untuk informasi selengkapnya, lihat Menanyakan beacon dalam database multitenant.

Anda harus menggunakan keyring AWS KMS Hierarkis untuk membuat suar bertanda tangan.

Untuk mengonfigurasi suar yang ditandatangani, berikan nilai berikut.

Java

Konfigurasi suar majemuk

Contoh berikut mendefinisikan daftar bagian yang ditandatangani secara lokal dalam konfigurasi suar yang ditandatangani.

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

Definisi versi suar

Contoh berikut mendefinisikan daftar bagian yang ditandatangani secara global dalam versi beacon. Untuk informasi selengkapnya tentang mendefinisikan versi beacon, lihat Menggunakan beacon.

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

C# / .NET

Lihat contoh kode lengkapnya: BeaconConfig.cs

Konfigurasi suar yang ditandatangani

Contoh berikut mendefinisikan daftar bagian yang ditandatangani secara lokal dalam konfigurasi suar yang ditandatangani.

```
var compoundBeaconList = new List<CompoundBeacon>();
```

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

Definisi versi suar

Contoh berikut mendefinisikan daftar bagian yang ditandatangani secara global dalam versi beacon. Untuk informasi selengkapnya tentang mendefinisikan versi beacon, lihat Menggunakan beacon.

```
var beaconVersions = new List<BeaconVersion>
{
    new BeaconVersion
    {
        StandardBeacons = standardBeaconList,
        CompoundBeacons = compoundBeaconList,
        SignedParts = signedPartsList,
        Version = 1, // MUST be 1
        KeyStore = keyStore,
        KeySource = new BeaconKeySource
        {
            Single = new SingleKeyStore
            {
                KeyId = branchKeyId,
                CacheTTL = 6000
            }
        }
    }
};
```

Anda dapat menentukan bagian yang ditandatangani dalam daftar yang ditentukan secara lokal atau global. Kami merekomendasikan untuk menentukan bagian yang Anda tandatangani dalam daftar global dalam versi suar bila memungkinkan. Dengan mendefinisikan bagian yang ditandatangani secara global, Anda dapat menentukan setiap bagian sekali dan kemudian menggunakan kembali bagian-bagian tersebut dalam beberapa konfigurasi suar majemuk. Jika Anda hanya ingin menggunakan bagian yang ditandatangani sekali, Anda dapat mendefinisikannya dalam daftar lokal

dalam konfigurasi suar yang ditandatangani. Anda dapat mereferensikan bagian lokal dan global dalam daftar konstruktor Anda.

Jika Anda menentukan daftar bagian yang ditandatangani secara global, Anda harus memberikan daftar bagian konstruktor yang mengidentifikasi semua kemungkinan cara suar yang ditandatangani dapat merakit bidang dalam konfigurasi suar Anda.

> **Note**
>
> Untuk menentukan daftar bagian yang ditandatangani secara global, Anda harus menggunakan versi 3.2 atau yang lebih baru dari SDK AWS Encryption SDK. Men-deploy versi baru untuk semua pembaca sebelum mendefinisikan bagian baru secara global. Anda tidak dapat memperbarui konfigurasi suar yang ada untuk menentukan daftar bagian yang ditandatangani secara global.

Nama suar

Nama yang Anda gunakan saat menanyakan suar.

Nama suar yang ditandatangani tidak bisa menjadi nama yang sama dengan bidang yang tidak terenkripsi. Tidak ada dua suar yang bisa memiliki nama suar yang sama.

Split character

Karakter yang digunakan untuk memisahkan bagian-bagian yang membentuk suar yang ditandatangani Anda.

Karakter split tidak dapat muncul dalam nilai plaintext dari salah satu bidang tempat suar yang ditandatangani dibuat.

Daftar bagian yang ditandatangani

Mengidentifikasi bidang yang ditandatangani termasuk dalam suar yang ditandatangani.

Setiap bagian harus menyertakan nama, sumber, dan awalan. Sumbernya adalah `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` bidang `SIGN_ONLY` atau yang diidentifikasi oleh bagian tersebut. Sumber harus berupa nama bidang atau indeks yang mengacu pada nilai bidang bersarang. Jika nama bagian Anda mengidentifikasi sumber, Anda dapat menghilangkan sumber dan SDK Enkripsi AWS Database akan secara otomatis menggunakan nama sebagai sumbernya. Sebaiknya tentukan sumber sebagai nama bagian bila memungkinkan. Awalan dapat berupa string apa saja, tetapi harus unik. Tidak ada dua bagian yang ditandatangani dalam suar

bertanda tangan yang dapat memiliki awalan yang sama. Sebaiknya gunakan nilai pendek yang membedakan bagian dari bagian lain yang dilayani oleh suar majemuk.

Kami merekomendasikan untuk menentukan suku cadang Anda yang ditandatangani secara global bila memungkinkan. Anda dapat mempertimbangkan untuk mendefinisikan bagian yang ditandatangani secara lokal jika Anda hanya bermaksud menggunakannya dalam satu suar majemuk. Bagian yang didefinisikan secara lokal tidak dapat memiliki awalan atau nama yang sama dengan bagian yang didefinisikan secara global.

Java

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

C# / .NET

```
var signedPartsList = new List<SignedPart>
{
    new SignedPart { Name = "signedFieldName1", Prefix = "S-" },
    new SignedPart { Name = "signedFieldName2", Prefix = "SF-" }
};
```

Daftar konstruktor (Opsional)

Mengidentifikasi konstruktor yang menentukan berbagai cara agar bagian yang ditandatangani dapat dirakit oleh suar yang ditandatangani.

Jika Anda tidak menentukan daftar konstruktor, AWS Database Encryption SDK akan merakit beacon yang ditandatangani dengan konstruktor default berikut.

- Semua bagian yang ditandatangani sesuai urutan ditambahkan ke daftar bagian yang ditandatangani
- Semua bagian diperlukan

Konstruktor

Setiap konstruktor adalah daftar terurut dari bagian-bagian konstruktor yang mendefinisikan satu cara bahwa suar yang ditandatangani dapat dirakit. Bagian konstruktor digabungkan

bersama dalam urutan mereka ditambahkan ke daftar, dengan setiap bagian dipisahkan oleh karakter split yang ditentukan.

Setiap bagian konstruktor memberi nama bagian yang ditandatangani, dan menentukan apakah bagian itu diperlukan atau opsional dalam konstruktor. Misalnya, jika Anda ingin menanyakan suar yang ditandatangani pada`Field1`,, dan `Field1.Field2``Field1.Field2.Field3`, tandai dan `Field3` sebagai opsional `Field2` dan buat satu konstruktor.

Setiap konstruktor harus memiliki setidaknya satu bagian yang dibutuhkan. Sebaiknya buat bagian pertama di setiap konstruktor yang diperlukan sehingga Anda dapat menggunakan `BEGINS_WITH` operator dalam kueri Anda.

Konstruktor berhasil jika semua bagian yang diperlukan ada dalam catatan. Saat Anda menulis catatan baru, suar yang ditandatangani menggunakan daftar konstruktor untuk menentukan apakah suar dapat dirakit dari nilai yang diberikan. Ini mencoba untuk merakit suar dalam urutan bahwa konstruktor ditambahkan ke daftar konstruktor, dan menggunakan konstruktor pertama yang berhasil. Jika tidak ada konstruktor yang berhasil, suar tidak ditulis ke catatan.

Semua pembaca dan penulis harus menentukan urutan konstruktor yang sama untuk memastikan bahwa hasil kueri mereka benar.

Gunakan prosedur berikut untuk menentukan daftar konstruktor Anda sendiri.

1. Buat bagian konstruktor untuk setiap bagian yang ditandatangani untuk menentukan apakah bagian itu diperlukan atau tidak.

   Nama bagian konstruktor harus nama bidang yang ditandatangani.

   Contoh berikut menunjukkan cara membuat bagian konstruktor untuk satu bidang yang ditandatangani.

   Java

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

C# / .NET

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

2. Buat konstruktor untuk setiap cara yang mungkin agar suar yang ditandatangani dapat dirakit menggunakan bagian konstruktor yang Anda buat di Langkah 1.

   Misalnya, jika Anda ingin menanyakan `Field1.Field2.Field3` dan`Field4.Field2.Field3`, Anda harus membuat dua konstruktor. `Field1`dan keduanya `Field4` dapat diperlukan karena mereka didefinisikan dalam dua konstruktor terpisah.

   Java

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

   C# / .NET

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

```
    Parts = new List<ConstructorPart> { field4ConstructorPart,
 field2ConstructorPart, field1ConstructorPart }
};
```

3. Buat daftar konstruktor yang mencakup semua konstruktor yang Anda buat di Langkah 2.

   Java

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

   C# / .NET

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

4. Tentukan `constructorList` kapan Anda membuat suar yang ditandatangani.

Penyimpanan kunci dalam SDK Enkripsi AWS Database

Dalam SDK Enkripsi AWS Database, penyimpanan kunci adalah tabel Amazon DynamoDB yang mempertahankan data hierarkis yang digunakan oleh keyring Hierarkis.AWS KMS Toko kunci membantu mengurangi jumlah panggilan yang perlu Anda lakukan AWS KMS untuk melakukan operasi kriptografi dengan keyring Hierarkis.

Penyimpanan kunci tetap ada dan mengelola kunci cabang yang digunakan keyring Hierarkis untuk melakukan enkripsi amplop dan melindungi kunci enkripsi data. Key store menyimpan kunci cabang aktif dan semua versi sebelumnya dari kunci cabang. Kunci cabang aktif adalah versi kunci cabang terbaru. Keyring Hierarkis menggunakan kunci enkripsi data unik untuk setiap permintaan enkripsi dan mengenkripsi setiap kunci enkripsi data dengan kunci pembungkus unik yang berasal dari kunci cabang aktif. Keyring Hierarkis bergantung pada hierarki yang ditetapkan antara kunci cabang aktif dan kunci pembungkus turunannya.

Terminologi dan konsep toko kunci

Toko kunci

Tabel DynamoDB yang mempertahankan data hierarkis, seperti kunci cabang dan kunci suar.

Kunci root

Kunci KMS enkripsi simetris yang menghasilkan dan melindungi kunci cabang dan kunci suar di toko kunci Anda.

Kunci cabang

Kunci data yang digunakan kembali untuk mendapatkan kunci pembungkus unik untuk enkripsi amplop. Anda dapat membuat beberapa kunci cabang dalam satu penyimpanan kunci, tetapi setiap kunci cabang hanya dapat memiliki satu versi kunci cabang aktif pada satu waktu. Kunci cabang aktif adalah versi kunci cabang terbaru.

Kunci cabang berasal dari AWS KMS keys menggunakan kms: GenerateDataKeyWithoutPlaintext operasi.

Kunci pembungkus

Kunci data unik yang digunakan untuk mengenkripsi kunci enkripsi data yang digunakan dalam operasi enkripsi.

Kunci pembungkus berasal dari kunci cabang. Untuk informasi selengkapnya tentang proses derivasi kunci, lihat Detail teknis keyring AWS KMS hierarkis.

Kunci enkripsi data

Kunci data yang digunakan dalam operasi enkripsi. Keyring Hierarkis menggunakan kunci enkripsi data unik untuk setiap permintaan enkripsi.

Kunci suar

Kunci data yang digunakan untuk menghasilkan beacon untuk enkripsi yang dapat dicari. Untuk informasi selengkapnya, lihat Enkripsi yang dapat dicari.

Menerapkan izin yang paling tidak diistimewakan

Saat menggunakan penyimpanan kunci dan gantungan kunci AWS KMS Hierarkis, kami sarankan Anda mengikuti prinsip hak istimewa paling sedikit dengan mendefinisikan peran berikut:

Administrator toko kunci

Administrator toko utama bertanggung jawab untuk membuat dan mengelola toko kunci dan kunci cabang yang bertahan dan dilindungi. Administrator toko utama harus menjadi satu-satunya pengguna dengan izin menulis ke tabel Amazon DynamoDB yang berfungsi sebagai toko kunci Anda. Mereka harus menjadi satu-satunya pengguna dengan akses ke operasi administrator istimewa, seperti `CreateKey`dan `VersionKey`. Anda hanya dapat melakukan operasi ini ketika Anda mengonfigurasi tindakan penyimpanan kunci secara statis.

`CreateKey`adalah operasi istimewa yang dapat menambahkan ARN kunci KMS baru ke daftar izin toko kunci Anda. Kunci KMS ini dapat membuat kunci cabang aktif baru. Kami menyarankan untuk membatasi akses ke operasi ini karena setelah kunci KMS ditambahkan ke toko kunci cabang, itu tidak dapat dihapus.

Pengguna toko kunci

Dalam kebanyakan kasus penggunaan, pengguna key store hanya berinteraksi dengan key store melalui keyring Hierarchical saat mereka mengenkripsi, mendekripsi, menandatangani, dan memverifikasi data. Akibatnya, mereka hanya perlu izin baca ke tabel Amazon DynamoDB yang berfungsi sebagai toko kunci Anda. Pengguna toko kunci hanya perlu akses ke operasi penggunaan yang memungkinkan operasi kriptografi, seperti`GetActiveBranchKey`,`GetBranchKeyVersion`, dan`GetBeaconKey`. Mereka tidak memerlukan izin untuk membuat atau mengelola kunci cabang yang mereka gunakan.

Anda dapat melakukan operasi penggunaan ketika tindakan penyimpanan kunci Anda dikonfigurasi secara statis, atau ketika mereka dikonfigurasi untuk penemuan. Anda tidak dapat melakukan operasi administrator (`CreateKeydanVersionKey`) ketika tindakan penyimpanan kunci Anda dikonfigurasi untuk penemuan.

Jika administrator toko kunci cabang Anda mengizinkan daftar beberapa kunci KMS di toko kunci cabang Anda, kami menyarankan agar pengguna toko kunci Anda mengonfigurasi tindakan penyimpanan kunci mereka untuk ditemukan sehingga keyring Hierarkis mereka dapat menggunakan beberapa kunci KMS.

Buat toko kunci

Sebelum Anda dapat membuat kunci cabang atau menggunakan keyring AWS KMS Hierarkis, Anda harus membuat toko kunci Anda, tabel Amazon DynamoDB yang mengelola dan melindungi kunci cabang Anda.

> **Important**
>
> Jangan hapus tabel DynamoDB yang mempertahankan kunci cabang Anda. Jika Anda menghapus tabel ini, Anda tidak akan dapat mendekripsi data apa pun yang dienkripsi menggunakan keyring Hierarkis.

Ikuti prosedur Buat tabel di Panduan Pengembang Amazon DynamoDB, menggunakan nilai string yang diperlukan berikut untuk kunci partisi dan kunci sortir.

	Kunci partisi	Sortir kunci
Tabel dasar	`branch-key-id`	`type`

Nama toko kunci logis

Saat menamai tabel DynamoDB yang berfungsi sebagai penyimpanan kunci Anda, penting untuk mempertimbangkan dengan cermat nama toko kunci logis yang akan Anda tentukan saat mengonfigurasi tindakan penyimpanan kunci Anda. Nama penyimpanan kunci logis bertindak sebagai pengidentifikasi untuk toko kunci Anda dan tidak dapat diubah setelah awalnya ditentukan

oleh pengguna pertama. Anda harus selalu menentukan nama penyimpanan kunci logis yang sama dalam tindakan penyimpanan kunci Anda.

Harus ada one-to-one pemetaan antara nama tabel DynamoDB dan nama toko kunci logis. Nama penyimpanan kunci logis terikat secara kriptografis ke semua data yang disimpan dalam tabel untuk menyederhanakan operasi pemulihan DynamoDB. Meskipun nama toko kunci logis dapat berbeda dari nama tabel DynamoDB Anda, kami sangat menyarankan untuk menentukan nama tabel DynamoDB Anda sebagai nama toko kunci logis. Jika nama tabel Anda berubah setelah memulihkan tabel DynamoDB Anda dari cadangan, nama penyimpanan kunci logis dapat dipetakan ke nama tabel DynamoDB baru untuk memastikan bahwa keyring Hierarkis masih dapat mengakses penyimpanan kunci Anda.

Jangan sertakan informasi rahasia atau sensitif dalam nama toko kunci logis Anda. Nama penyimpanan kunci logis ditampilkan dalam teks biasa dalam AWS KMS CloudTrail peristiwa sebagai. `tablename`

Langkah selanjutnya

1. the section called “Konfigurasikan tindakan penyimpanan kunci”
2. the section called “Buatlah sebuah cabang”
3. Buat keyring AWS KMS Hierarkis

Konfigurasikan tindakan penyimpanan kunci

Tindakan penyimpanan kunci menentukan operasi apa yang dapat dilakukan pengguna Anda dan bagaimana keyring AWS KMS Hierarkis mereka menggunakan kunci KMS yang diizinkan terdaftar di toko kunci Anda. AWS Database Encryption SDK mendukung konfigurasi tindakan penyimpanan kunci berikut.

Statis

Saat Anda mengonfigurasi penyimpanan kunci secara statis, toko kunci hanya dapat menggunakan kunci KMS yang terkait dengan ARN kunci KMS yang Anda berikan `kmsConfiguration` saat Anda mengonfigurasi tindakan penyimpanan kunci Anda. Pengecualian dilemparkan jika ARN kunci KMS yang berbeda ditemukan saat membuat, membuat versi, atau mendapatkan kunci cabang.

Anda dapat menentukan kunci KMS Multi-wilayah di Anda`kmsConfiguration`, tetapi seluruh ARN kunci, termasuk wilayah, tetap ada di kunci cabang yang berasal dari kunci KMS. Anda tidak

dapat menentukan kunci di wilayah yang berbeda, Anda harus memberikan kunci multi-wilayah yang sama persis agar nilainya cocok.

Saat Anda mengonfigurasi tindakan penyimpanan kunci secara statis, Anda dapat melakukan operasi penggunaan (`GetActiveBranchKey`,`GetBranchKeyVersion`,`GetBeaconKey`) dan operasi administratif (`CreateKey`dan`VersionKey`). `CreateKey`adalah operasi istimewa yang dapat menambahkan ARN kunci KMS baru ke daftar izin toko kunci Anda. Kunci KMS ini dapat membuat kunci cabang aktif baru. Kami menyarankan untuk membatasi akses ke operasi ini karena setelah kunci KMS ditambahkan ke toko kunci, itu tidak dapat dihapus.

Penemuan

Saat Anda mengonfigurasi tindakan penyimpanan kunci Anda untuk penemuan, toko kunci dapat menggunakan AWS KMS key ARN apa pun yang diizinkan terdaftar di toko kunci Anda. Namun, pengecualian dilemparkan ketika kunci KMS Multi-wilayah ditemui dan wilayah di ARN kunci tidak cocok dengan wilayah klien yang digunakan. AWS KMS

Ketika Anda mengonfigurasi penyimpanan kunci untuk penemuan, Anda tidak dapat melakukan operasi administratif, seperti `CreateKey` dan`VersionKey`. Anda hanya dapat melakukan operasi penggunaan yang mengaktifkan enkripsi, mendekripsi, menandatangani, dan memverifikasi operasi. Untuk informasi selengkapnya, lihat the section called “Menerapkan izin yang paling tidak diistimewakan”.

Konfigurasikan tindakan penyimpanan kunci Anda

Sebelum Anda mengonfigurasi tindakan penyimpanan kunci Anda, pastikan prasyarat berikut terpenuhi.

- Tentukan operasi apa yang perlu Anda lakukan. Untuk informasi selengkapnya, lihat the section called “Menerapkan izin yang paling tidak diistimewakan”.
- Pilih nama toko kunci logis

  Harus ada one-to-one pemetaan antara nama tabel DynamoDB dan nama toko kunci logis. Nama penyimpanan kunci logis terikat secara kriptografis ke semua data yang disimpan dalam tabel untuk menyederhanakan operasi pemulihan DynamoDB, tidak dapat diubah setelah awalnya ditentukan oleh pengguna pertama. Anda harus selalu menentukan nama penyimpanan kunci logis yang sama dalam tindakan penyimpanan kunci Anda. Untuk informasi selengkapnya, lihat logical key store name.

Konfigurasi statis

Contoh berikut secara statis mengkonfigurasi tindakan penyimpanan kunci. Anda harus menentukan nama tabel DynamoDB yang berfungsi sebagai penyimpanan kunci Anda, nama logis untuk penyimpanan kunci, dan ARN kunci KMS yang mengidentifikasi kunci KMS enkripsi simetris.

> **Note**
>
> Hati-hati mempertimbangkan ARN kunci KMS yang Anda tentukan saat mengonfigurasi layanan penyimpanan kunci Anda secara statis. `CreateKey`Operasi menambahkan ARN kunci KMS ke daftar izin toko kunci cabang Anda. Setelah kunci KMS ditambahkan ke toko kunci cabang, itu tidak dapat dihapus.

Java

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

C# / .NET

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

Rust

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

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

Konfigurasi penemuan

Contoh berikut mengonfigurasi tindakan penyimpanan kunci untuk penemuan. Anda harus menentukan nama tabel DynamoDB yang berfungsi sebagai toko kunci Anda dan nama toko kunci logis.

Java

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

C# / .NET

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

```
    LogicalKeyStoreName = logicalKeyStoreName
};
var keystore = new KeyStore(keystoreConfig);
```

Rust

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

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

Buat kunci cabang aktif

Kunci cabang adalah kunci data yang berasal dari AWS KMS key yang digunakan oleh keyring AWS KMS Hierarkis untuk mengurangi jumlah panggilan yang dilakukan. AWS KMS Kunci cabang aktif adalah versi kunci cabang terbaru. Keyring Hierarkis menghasilkan kunci data unik untuk setiap permintaan enkripsi dan mengenkripsi setiap kunci data dengan kunci pembungkus unik yang berasal dari kunci cabang aktif.

Untuk membuat kunci cabang aktif baru, Anda harus mengonfigurasi tindakan penyimpanan kunci secara statis. `CreateKey`adalah operasi istimewa yang menambahkan ARN kunci KMS yang ditentukan dalam konfigurasi tindakan penyimpanan kunci Anda ke daftar izin toko kunci Anda. Kemudian, kunci KMS digunakan untuk menghasilkan kunci cabang aktif baru. Kami menyarankan untuk membatasi akses ke operasi ini karena setelah kunci KMS ditambahkan ke toko kunci, itu tidak dapat dihapus.

Sebaiknya gunakan `CreateKey` operasi melalui antarmuka KeyStore Admin di bidang kontrol aplikasi Anda. Pendekatan ini sejalan dengan praktik terbaik untuk manajemen kunci.

Jangan membuat kunci cabang di bidang data. Praktik ini dapat menghasilkan:

- Panggilan yang tidak perlu ke AWS KMS
- Beberapa panggilan bersamaan ke AWS KMS dalam lingkungan konkurensi tinggi
- Beberapa TransactWriteItems panggilan ke tabel DynamoDB backing.

`CreateKey`Operasi ini mencakup pemeriksaan kondisi dalam `TransactWriteItems` panggilan untuk mencegah penimpaan kunci cabang yang ada. Namun, membuat kunci di bidang data masih dapat menyebabkan penggunaan sumber daya yang tidak efisien dan potensi masalah kinerja.

Anda dapat mengizinkan daftar satu kunci KMS di toko kunci Anda, atau Anda dapat mengizinkan beberapa kunci KMS dengan memperbarui ARN kunci KMS yang Anda tentukan dalam konfigurasi tindakan penyimpanan kunci Anda dan menelepon lagi. `CreateKey` Jika Anda mengizinkan beberapa kunci KMS, pengguna toko kunci Anda harus mengonfigurasi tindakan penyimpanan kunci mereka untuk penemuan sehingga mereka dapat menggunakan salah satu kunci yang diizinkan di toko kunci yang dapat mereka akses. Untuk informasi selengkapnya, lihat the section called “Konfigurasikan tindakan penyimpanan kunci”.

Izin yang diperlukan

Untuk membuat kunci cabang, Anda memerlukan ReEncrypt izin kms: GenerateDataKeyWithoutPlaintext dan kms: pada kunci KMS yang ditentukan dalam tindakan penyimpanan kunci Anda.

Buatlah sebuah cabang

Operasi berikut membuat kunci cabang aktif baru menggunakan kunci KMS yang Anda tentukan dalam konfigurasi tindakan penyimpanan kunci Anda, dan menambahkan kunci cabang aktif ke tabel DynamoDB yang berfungsi sebagai penyimpanan kunci Anda.

Saat Anda meneleponCreateKey, Anda dapat memilih untuk menentukan nilai opsional berikut.

- `branchKeyIdentifier`: mendefinisikan kustom`branch-key-id`.

  Untuk membuat kustom`branch-key-id`, Anda juga harus menyertakan konteks enkripsi tambahan dengan `encryptionContext` parameter.
- `encryptionContext`: mendefinisikan kumpulan opsional pasangan kunci-nilai non-rahasia yang menyediakan data terautentikasi tambahan (AAD) dalam konteks enkripsi yang disertakan dalam panggilan kms:. GenerateDataKeyWithoutPlaintext

  Konteks enkripsi tambahan ini ditampilkan dengan `aws-crypto-ec:` awalan.

Java

```
final Map<String, String> additionalEncryptionContext =
 Collections.singletonMap("Additional Encryption Context for",
```

```
            "custom branch key id");

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

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

C# / .NET

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

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

Rust

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

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

Pertama, `CreateKey` operasi menghasilkan nilai-nilai berikut.

- Versi 4 Universally Unique Identifier (UUID) untuk `branch-key-id` (kecuali Anda menentukan kustom). `branch-key-id`
- UUID versi 4 untuk versi kunci cabang

- A `timestamp` dalam format tanggal dan waktu ISO 8601 dalam Waktu Universal Terkoordinasi (UTC).

Kemudian, `CreateKey` operasi memanggil kms: GenerateDataKeyWithoutPlaintext menggunakan permintaan berikut.

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

> **Note**
>
> `CreateKey`Operasi membuat kunci cabang aktif dan kunci suar, bahkan jika Anda belum mengonfigurasi database Anda untuk enkripsi yang dapat dicari. Kedua kunci disimpan di toko kunci Anda. Untuk informasi selengkapnya, lihat Menggunakan keyring hierarkis untuk enkripsi yang dapat dicari.

Selanjutnya, `CreateKey` operasi memanggil kms: ReEncrypt untuk membuat catatan aktif untuk kunci cabang dengan memperbarui konteks enkripsi.

Terakhir, `CreateKey` operasi memanggil ddb: TransactWriteItems untuk menulis item baru yang akan mempertahankan kunci cabang dalam tabel yang Anda buat di Langkah 2. Item memiliki atribut berikut.

```
{
      "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"
}
```

Putar kunci cabang aktif Anda

Hanya ada satu versi aktif untuk setiap kunci cabang pada satu waktu. Biasanya, setiap versi kunci cabang aktif digunakan untuk memenuhi beberapa permintaan. Tetapi Anda mengontrol sejauh mana kunci cabang aktif digunakan kembali dan menentukan seberapa sering kunci cabang aktif diputar.

Kunci cabang tidak digunakan untuk mengenkripsi kunci data teks biasa. Mereka digunakan untuk mendapatkan kunci pembungkus unik yang mengenkripsi kunci data teks biasa. Proses derivasi kunci pembungkus menghasilkan kunci pembungkus 32 byte yang unik dengan 28 byte keacakan. Ini berarti bahwa kunci cabang dapat memperoleh lebih dari 79 oktillion, atau 2^{96}, kunci pembungkus unik sebelum keausan kriptografi terjadi. Meskipun risiko kelelahan yang sangat rendah ini, Anda mungkin diminta untuk memutar kunci cabang aktif Anda karena aturan bisnis atau kontrak atau peraturan pemerintah.

Versi aktif dari kunci cabang tetap aktif sampai Anda memutarnya. Versi sebelumnya dari kunci cabang aktif tidak akan digunakan untuk melakukan operasi enkripsi dan tidak dapat digunakan untuk mendapatkan kunci pembungkus baru, tetapi mereka masih dapat ditanyakan dan menyediakan kunci pembungkus untuk mendekripsi kunci data yang mereka enkripsi saat aktif.

> **Warning**
>
> Menghapus kunci cabang di lingkungan pengujian tidak dapat diubah. Anda tidak dapat memulihkan kunci cabang yang dihapus. Saat Anda menghapus dan membuat ulang kunci cabang dengan ID yang sama di lingkungan pengujian, masalah berikut dapat terjadi:
>
> - Materi dari pengujian sebelumnya mungkin tetap ada di cache
> - Beberapa host pengujian atau utas mungkin mengenkripsi data menggunakan kunci cabang yang dihapus
> - Data yang dienkripsi dengan cabang yang dihapus tidak dapat didekripsi
>
> Untuk mencegah kegagalan enkripsi dalam tes integrasi:

- Setel ulang referensi keyring hierarkis sebelum membuat kunci cabang baru ATAU
- Gunakan kunci cabang unik IDs untuk setiap pengujian

Izin yang diperlukan

Untuk memutar kunci cabang, Anda memerlukan ReEncrypt izin kms: GenerateDataKeyWithoutPlaintext dan kms: pada kunci KMS yang ditentukan dalam tindakan penyimpanan kunci Anda.

Putar tombol cabang aktif

Gunakan `VersionKey` operasi untuk memutar kunci cabang aktif Anda. Saat Anda memutar kunci cabang aktif, kunci cabang baru dibuat untuk menggantikan versi sebelumnya. `branch-key-id`Tidak berubah saat Anda memutar kunci cabang aktif. Anda harus menentukan `branch-key-id` yang mengidentifikasi kunci cabang aktif saat ini ketika Anda meneleponVersionKey.

Java

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

C# / .NET

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

Rust

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

Gantungan kunci

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

AWS Database Encryption SDK menggunakan keyrings untuk melakukan enkripsi amplop. Keyrings menghasilkan, mengenkripsi, dan mendekripsi kunci data. Keyrings menentukan sumber kunci data unik yang melindungi setiap catatan terenkripsi, dan kunci pembungkus yang mengenkripsi kunci data tersebut. Anda menentukan keyring saat mengenkripsi dan keyring yang sama atau berbeda saat mendekripsi.

Anda dapat menggunakan setiap keyring satu per satu atau menggabungkan keyrings menjadi multi-keyring. Meskipun sebagian besar keyrings dapat menghasilkan, mengenkripsi, dan mendekripsi kunci data, Anda dapat membuat keyring yang hanya melakukan satu operasi tertentu, seperti keyring yang hanya menghasilkan kunci data, dan menggunakan keyring tersebut dalam kombinasi dengan yang lain.

Kami menyarankan Anda menggunakan keyring yang melindungi kunci pembungkus Anda dan melakukan operasi kriptografi dalam batas aman, seperti AWS KMS keyring, yang menggunakan AWS KMS keys yang tidak pernah meninggalkan () tidak terenkripsi. AWS Key Management ServiceAWS KMS Anda juga dapat menulis keyring yang menggunakan kunci pembungkus yang disimpan dalam modul keamanan perangkat keras Anda (HSMs) atau dilindungi oleh layanan kunci utama lainnya.

Keyring Anda menentukan kunci pembungkus yang melindungi kunci data Anda, dan akhirnya, data Anda. Gunakan kunci pembungkus paling aman yang praktis untuk tugas Anda. Bila memungkinkan gunakan kunci pembungkus yang dilindungi oleh modul keamanan perangkat keras (HSM) atau infrastruktur manajemen kunci, seperti kunci KMS in AWS Key Management Service(AWS KMS) atau kunci enkripsi di. AWS CloudHSM

AWS Database Encryption SDK menyediakan beberapa konfigurasi keyrings dan keyring, dan Anda dapat membuat keyring kustom Anda sendiri. Anda juga dapat membuat multi-keyring yang menyertakan satu atau lebih gantungan kunci dari jenis yang sama atau berbeda.

Topik

- Cara kerja gantungan kunci

- AWS KMS gantungan kunci
- AWS KMS Gantungan kunci hierarkis
- AWS KMS Gantungan kunci ECDH
- Gantungan kunci AES mentah
- Gantungan kunci RSA mentah
- Gantungan kunci ECDH mentah
- Multi-gantungan kunci

Cara kerja gantungan kunci

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Saat Anda mengenkripsi dan menandatangani bidang di database Anda, SDK Enkripsi AWS Database meminta keyring untuk materi enkripsi. Keyring mengembalikan kunci data plaintext, salinan kunci data yang dienkripsi oleh masing-masing kunci pembungkus di keyring, dan kunci MAC yang terkait dengan kunci data. AWS Database Encryption SDK menggunakan kunci plaintext untuk mengenkripsi data, dan kemudian menghapus kunci data plaintext dari memori sesegera mungkin. Kemudian, AWS Database Encryption SDK menambahkan deskripsi material yang mencakup kunci data terenkripsi dan informasi lainnya, seperti enkripsi dan instruksi penandatanganan. SDK Enkripsi AWS Database menggunakan kunci MAC untuk menghitung Kode Otentikasi Pesan Berbasis Hash (HMACs) melalui kanonikalisasi deskripsi materi dan semua bidang yang ditandai atau. `ENCRYPT_AND_SIGN` `SIGN_ONLY`

Saat mendekripsi data, Anda dapat menggunakan keyring yang sama dengan yang Anda gunakan untuk mengenkripsi data, atau yang lain. Untuk mendekripsi data, keyring dekripsi harus memiliki akses ke setidaknya satu kunci pembungkus di keyring enkripsi.

AWS Database Encryption SDK meneruskan kunci data terenkripsi dari deskripsi material ke keyring, dan meminta keyring untuk mendekripsi salah satu dari mereka. Keyring menggunakan kunci pembungkusnya untuk mendekripsi salah satu kunci data terenkripsi dan mengembalikan kunci data plaintext. AWS Database Encryption SDK menggunakan kunci data plaintext untuk mendekripsi data. Jika tidak ada kunci pembungkus di keyring yang dapat mendekripsi salah satu kunci data terenkripsi, operasi dekripsi gagal.

Anda dapat menggunakan keyring tunggal atau juga menggabungkan keyrings dari jenis yang sama atau jenis yang berbeda ke dalam multi-keyring. Saat Anda mengenkripsi data, multi-keyring mengembalikan salinan kunci data yang dienkripsi oleh semua kunci pembungkus di semua keyring yang terdiri dari multi-keyring dan kunci MAC yang terkait dengan kunci data. Anda dapat mendekripsi data menggunakan keyring dengan salah satu tombol pembungkus di multi-keyring.

AWS KMS gantungan kunci

> Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

AWS KMS Keyring menggunakan enkripsi simetris atau RSA asimetris AWS KMS keysuntuk menghasilkan, mengenkripsi, dan mendekripsi kunci data. AWS Key Management Service (AWS KMS) melindungi kunci KMS Anda dan melakukan operasi kriptografi dalam batas FIPS. Kami menyarankan Anda menggunakan AWS KMS keyring, atau keyring dengan properti keamanan serupa, bila memungkinkan.

Anda juga dapat menggunakan kunci KMS Multi-wilayah simetris dalam keyring. AWS KMS Untuk detail dan contoh selengkapnya menggunakan Multi-region AWS KMS keys, lihatMenggunakan Multi-region AWS KMS keys. Untuk informasi tentang kunci Multi-region, lihat Menggunakan kunci Multi-region di Panduan AWS Key Management Service Pengembang.

AWS KMS gantungan kunci dapat mencakup dua jenis kunci pembungkus:

- Kunci generator: Menghasilkan kunci data teks biasa dan mengenkripsinya. Sebuah keyring yang mengenkripsi data harus memiliki satu kunci generator.
- Kunci tambahan: Mengenkripsi kunci data teks biasa yang dihasilkan oleh kunci generator. AWS KMS keyrings dapat memiliki nol atau lebih tombol tambahan.

Anda harus memiliki kunci generator untuk mengenkripsi catatan. Ketika AWS KMS keyring hanya memiliki satu AWS KMS kunci, kunci itu digunakan untuk menghasilkan dan mengenkripsi kunci data.

Seperti semua gantungan kunci, AWS KMS gantungan kunci dapat digunakan secara independen atau dalam multi-keyring dengan gantungan kunci lain dari jenis yang sama atau berbeda.

Topik

- Izin yang diperlukan untuk keyrings AWS KMS
- Mengidentifikasi AWS KMS keys dalam AWS KMS keyring
- Membuat AWS KMS keyring
- Menggunakan Multi-region AWS KMS keys
- Menggunakan AWS KMS keyring penemuan
- Menggunakan AWS KMS keyring penemuan regional

Izin yang diperlukan untuk keyrings AWS KMS

SDK Enkripsi AWS Database tidak memerlukan Akun AWS dan tidak bergantung pada apa pun Layanan AWS. Namun, untuk menggunakan AWS KMS keyring, Anda memerlukan izin minimum Akun AWS dan berikut pada keyring Anda. AWS KMS keys

- Untuk mengenkripsi dengan AWS KMS keyring, Anda memerlukan GenerateDataKey izin kms: pada kunci generator. Anda memerlukan izin KMS: Encrypt pada semua kunci tambahan di keyring. AWS KMS
- Untuk mendekripsi dengan AWS KMS keyring, Anda memerlukan izin KMS: Decrypt pada setidaknya satu kunci di keyring. AWS KMS
- Untuk mengenkripsi dengan multi-keyring yang terdiri dari AWS KMS keyrings, Anda memerlukan GenerateDataKey izin kms: pada kunci generator di keyring generator. Anda memerlukan izin KMS: Encrypt pada semua kunci lain di semua keyrings lainnya. AWS KMS
- Untuk mengenkripsi dengan AWS KMS keyring RSA asimetris, Anda tidak perlu kms: GenerateDataKey atau KMS:Encrypt karena Anda harus menentukan materi kunci publik yang ingin Anda gunakan untuk enkripsi saat Anda membuat keyring. Tidak ada AWS KMS panggilan yang dilakukan saat mengenkripsi dengan keyring ini. Untuk mendekripsi dengan AWS KMS keyring RSA asimetris, Anda memerlukan izin KMS: Dekripsi.

Untuk informasi selengkapnya tentang izin AWS KMS keys, lihat Otentikasi dan kontrol akses di Panduan AWS Key Management Service Pengembang.

Mengidentifikasi AWS KMS keys dalam AWS KMS keyring

AWS KMS Keyring dapat mencakup satu atau lebih AWS KMS keys. Untuk menentukan AWS KMS key dalam AWS KMS keyring, gunakan pengenal AWS KMS kunci yang didukung. Pengidentifikasi kunci yang dapat Anda gunakan untuk mengidentifikasi AWS KMS key dalam keyring bervariasi

dengan operasi dan implementasi bahasa. Untuk detail tentang pengidentifikasi kunci AWS KMS key, lihat Pengidentifikasi Kunci di Panduan AWS Key Management Service Pengembang.

Sebagai praktik terbaik, gunakan pengenal kunci paling spesifik yang praktis untuk tugas Anda.

- Untuk mengenkripsi dengan AWS KMS keyring, Anda dapat menggunakan ID kunci, ARN kunci, nama alias, atau alias ARN untuk mengenkripsi data.

  > **Note**
  >
  > Jika Anda menentukan nama alias atau alias ARN untuk kunci KMS dalam keyring enkripsi, operasi enkripsi menyimpan ARN kunci yang saat ini terkait dengan alias dalam metadata kunci data terenkripsi. Itu tidak menyimpan alias. Perubahan pada alias tidak memengaruhi kunci KMS yang digunakan untuk mendekripsi kunci data terenkripsi Anda.

- Untuk mendekripsi dengan AWS KMS keyring, Anda harus menggunakan ARN kunci untuk mengidentifikasi. AWS KMS keys Untuk detailnya, lihat Memilih tombol pembungkus.
- Dalam keyring yang digunakan untuk enkripsi dan dekripsi, Anda harus menggunakan ARN kunci untuk mengidentifikasi. AWS KMS keys

Saat mendekripsi, AWS Database Encryption SDK mencari AWS KMS keyring untuk kunci AWS KMS key yang dapat mendekripsi salah satu kunci data terenkripsi. Secara khusus, SDK Enkripsi AWS Database menggunakan pola berikut untuk setiap kunci data terenkripsi dalam deskripsi material.

- AWS Database Encryption SDK mendapatkan ARN kunci AWS KMS key yang mengenkripsi kunci data dari metadata deskripsi material.
- AWS Database Encryption SDK mencari keyring dekripsi untuk ARN dengan kunci AWS KMS key yang cocok.
- Jika menemukan ARN AWS KMS key dengan kunci yang cocok di keyring, SDK Enkripsi AWS Database meminta AWS KMS untuk menggunakan kunci KMS untuk mendekripsi kunci data terenkripsi.
- Jika tidak, ia melompat ke kunci data terenkripsi berikutnya, jika ada.

Membuat AWS KMS keyring

Anda dapat mengonfigurasi setiap AWS KMS keyring dengan satu AWS KMS key atau beberapa AWS KMS keys yang sama atau berbeda Akun AWS dan Wilayah AWS. AWS KMS key Harus berupa kunci enkripsi simetris (`SYMMETRIC_DEFAULT`) atau kunci KMS RSA asimetris. Anda juga dapat menggunakan enkripsi simetris Multi-region KMS key. Anda dapat menggunakan satu atau lebih AWS KMS keyring dalam multi-keyring.

Anda dapat membuat AWS KMS keyring yang mengenkripsi dan mendekripsi data, atau Anda dapat membuat AWS KMS gantungan kunci khusus untuk mengenkripsi atau mendekripsi. Saat Anda membuat AWS KMS keyring untuk mengenkripsi data, Anda harus menentukan kunci generator, AWS KMS key yang digunakan untuk menghasilkan kunci data plaintext dan mengenkripsinya. Kunci data secara matematis tidak terkait dengan kunci KMS. Kemudian, jika Anda memilih, Anda dapat menentukan tambahan AWS KMS keys yang mengenkripsi kunci data plaintext yang sama. Untuk mendekripsi bidang terenkripsi yang dilindungi oleh keyring ini, keyring dekripsi yang Anda gunakan harus menyertakan setidaknya satu dari yang ditentukan dalam keyring, atau tidak. AWS KMS keys AWS KMS keys(AWS KMS Gantungan kunci tanpa AWS KMS keys dikenal sebagai gantungan kunci AWS KMS penemuan.)

Semua kunci pembungkus dalam keyring enkripsi atau multi-keyring harus dapat mengenkripsi kunci data. Jika ada kunci pembungkus gagal untuk mengenkripsi, metode enkripsi gagal. Akibatnya, penelepon harus memiliki izin yang diperlukan untuk semua kunci di keyring. Jika Anda menggunakan keyring penemuan untuk mengenkripsi data, sendiri atau dalam multi-keyring, operasi enkripsi gagal.

Contoh berikut menggunakan `CreateAwsKmsMrkMultiKeyring` metode untuk membuat AWS KMS keyring dengan kunci KMS enkripsi simetris. `CreateAwsKmsMrkMultiKeyring`Metode ini secara otomatis membuat AWS KMS klien dan memastikan bahwa keyring akan menangani kunci Single-region dan Multi-region dengan benar. Contoh-contoh ini menggunakan kunci ARNs untuk mengidentifikasi kunci KMS. Untuk detailnya, lihat Mengidentifikasi AWS KMS keys dalam AWS KMS keyring

Java

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

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

C# / .NET

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

Rust

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

Contoh berikut menggunakan `CreateAwsKmsRsaKeyring` metode untuk membuat AWS KMS keyring dengan kunci KMS RSA asimetris. Untuk membuat AWS KMS keyring RSA asimetris, berikan nilai berikut.

- `kmsClient`: buat AWS KMS klien baru
- `kmsKeyID`: kunci ARN yang mengidentifikasi kunci KMS RSA asimetris Anda
- `publicKey`: file PEM yang dikodekan UTF-8 yang mewakili kunci publik dari kunci yang Anda kirimkan ByteBuffer `kmsKeyID`
- `encryptionAlgorithm`: algoritma enkripsi harus `RSAES_OAEP_SHA_256` atau `RSAES_OAEP_SHA_1`

Java

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

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

C# / .NET

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

Rust

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

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

Menggunakan Multi-region AWS KMS keys

Anda dapat menggunakan Multi-region AWS KMS keys sebagai kunci pembungkus di SDK Enkripsi AWS Database. Jika Anda mengenkripsi dengan kunci Multi-wilayah dalam satu Wilayah AWS, Anda dapat mendekripsi menggunakan kunci Multi-wilayah terkait di yang berbeda. Wilayah AWS

Kunci KMS Multi-Region adalah satu AWS KMS keys set berbeda Wilayah AWS yang memiliki bahan kunci dan ID kunci yang sama. Anda dapat menggunakan kunci terkait ini seolah-olah mereka adalah kunci yang sama di Wilayah yang berbeda. Kunci Multi-Region mendukung pemulihan bencana umum dan skenario pencadangan yang memerlukan enkripsi di satu Wilayah dan mendekripsi di Wilayah yang berbeda tanpa melakukan panggilan lintas wilayah. AWS KMS Untuk informasi tentang kunci Multi-region, lihat Menggunakan kunci Multi-region di Panduan AWS Key Management Service Pengembang.

Untuk mendukung kunci Multi-region, AWS Database Encryption SDK menyertakan AWS KMS multi-Region-aware keyrings. `CreateAwsKmsMrkMultiKeyring`Metode ini mendukung kunci Single-region dan Multi-region.

- Untuk kunci wilayah Tunggal, multi-Region-aware simbol berperilaku seperti keyring wilayah Tunggal AWS KMS . Ini mencoba untuk mendekripsi ciphertext hanya dengan kunci Single-region yang mengenkripsi data. Untuk menyederhanakan pengalaman AWS KMS keyring Anda, sebaiknya gunakan `CreateAwsKmsMrkMultiKeyring` metode ini setiap kali Anda menggunakan kunci KMS enkripsi simetris.
- Untuk kunci Multi-region, multi-Region-aware simbol mencoba mendekripsi ciphertext dengan kunci Multi-region yang sama yang mengenkripsi data atau dengan kunci Multi-region terkait di Wilayah yang Anda tentukan.

Dalam multi-Region-aware gantungan kunci yang mengambil lebih dari satu kunci KMS, Anda dapat menentukan beberapa kunci Single-region dan Multi-region. Namun, Anda hanya dapat menentukan satu kunci dari setiap set kunci Multi-wilayah terkait. Jika Anda menentukan lebih dari satu pengenal kunci dengan ID kunci yang sama, panggilan konstruktor gagal.

Contoh berikut membuat AWS KMS keyring dengan kunci KMS Multi-region. Contoh menentukan kunci Multi-region sebagai kunci generator dan kunci Single-region sebagai kunci anak.

Java

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

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

C# / .NET

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

Rust

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

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

Saat Anda menggunakan AWS KMS gantungan kunci Multi-wilayah, Anda dapat mendekripsi ciphertext dalam mode ketat atau mode temukan. Untuk mendekripsi ciphertext dalam mode ketat, buat instance simbol multi-Region-aware dengan kunci ARN dari kunci Multi-region terkait di wilayah yang Anda dekripsi ciphertext. Jika Anda menentukan kunci ARN dari kunci Multi-wilayah terkait di Wilayah yang berbeda (misalnya, wilayah tempat catatan dienkripsi), multi-Region-aware simbol akan membuat panggilan lintas wilayah untuk itu. AWS KMS key

Saat mendekripsi dalam mode ketat, multi-Region-aware simbol membutuhkan kunci ARN. Ini hanya menerima satu ARN kunci dari setiap set kunci Multi-wilayah terkait.

Anda juga dapat mendekripsi dalam mode penemuan dengan tombol AWS KMS Multi-wilayah. Saat mendekripsi dalam mode penemuan, Anda tidak menentukan apa pun. AWS KMS keys(Untuk informasi tentang gantungan kunci AWS KMS penemuan wilayah tunggal, lihatMenggunakan AWS KMS keyring penemuan.)

Jika Anda dienkripsi dengan kunci Multi-region, multi-Region-aware simbol dalam mode penemuan akan mencoba mendekripsi dengan menggunakan kunci Multi-wilayah terkait di Wilayah lokal. Jika tidak ada; panggilan gagal. Dalam mode penemuan, SDK Enkripsi AWS Database tidak akan mencoba panggilan lintas wilayah untuk kunci Multi-wilayah yang digunakan untuk enkripsi.

Menggunakan AWS KMS keyring penemuan

Saat mendekripsi, ini adalah praktik terbaik untuk menentukan kunci pembungkus yang dapat digunakan SDK Enkripsi AWS Database. Untuk mengikuti praktik terbaik ini, gunakan keyring AWS KMS dekripsi yang membatasi kunci AWS KMS pembungkus ke kunci yang Anda tentukan. Namun, Anda juga dapat membuat keyring AWS KMS penemuan, yaitu AWS KMS keyring yang tidak menentukan kunci pembungkus apa pun.

AWS Database Encryption SDK menyediakan keyring AWS KMS penemuan standar dan keyring penemuan untuk AWS KMS kunci Multi-region. Untuk informasi tentang menggunakan kunci Multi-region dengan AWS Database Encryption SDK, lihat. Menggunakan Multi-region AWS KMS keys

Karena tidak menentukan kunci pembungkus apa pun, keyring penemuan tidak dapat mengenkripsi data. Jika Anda menggunakan keyring penemuan untuk mengenkripsi data, sendiri atau dalam multi-keyring, operasi enkripsi gagal.

Saat mendekripsi, keyring penemuan memungkinkan SDK Enkripsi AWS Database meminta AWS KMS untuk mendekripsi kunci data terenkripsi apa pun dengan menggunakan kunci yang dienkripsi, terlepas dari siapa AWS KMS key yang memiliki atau memiliki akses ke sana. AWS KMS key Panggilan hanya berhasil ketika penelepon memiliki `kms:Decrypt` izin pada. AWS KMS key

> **Important**
>
> Jika Anda menyertakan keyring AWS KMS penemuan dalam multi-keyring dekripsi, keyring penemuan mengesampingkan semua batasan kunci KMS yang ditentukan oleh gantungan kunci lain di multi-keyring. Multi-keyring berperilaku seperti keyring yang paling tidak membatasi. Jika Anda menggunakan keyring penemuan untuk mengenkripsi data, sendiri atau dalam multi-keyring, operasi enkripsi gagal

AWS Database Encryption SDK menyediakan keyring AWS KMS penemuan untuk kenyamanan. Namun, kami menyarankan Anda menggunakan keyring yang lebih terbatas bila memungkinkan karena alasan berikut.

- Keaslian — Keyring AWS KMS penemuan dapat menggunakan apa pun AWS KMS key yang digunakan untuk mengenkripsi kunci data dalam deskripsi materi, selama penelepon memiliki izin untuk menggunakannya untuk mendekripsi. AWS KMS key Ini mungkin bukan AWS KMS key yang ingin digunakan penelepon. Misalnya, salah satu kunci data terenkripsi mungkin telah dienkripsi di bawah yang kurang aman AWS KMS key yang dapat digunakan siapa pun.
- Latensi dan kinerja — Keyring AWS KMS penemuan mungkin terlihat lebih lambat daripada keyring lain karena SDK Enkripsi AWS Database mencoba mendekripsi semua kunci data terenkripsi, termasuk yang dienkripsi oleh AWS KMS keys di lain Akun AWS dan Wilayah, dan AWS KMS keys bahwa pemanggil tidak memiliki izin untuk digunakan untuk dekripsi.

Jika Anda menggunakan keyring penemuan, kami sarankan Anda menggunakan filter penemuan untuk membatasi kunci KMS yang dapat digunakan untuk kunci yang ditentukan Akun AWS dan partisi. Untuk bantuan menemukan ID akun dan partisi Anda, lihat Akun AWS Pengenal Anda dan format ARN di. Referensi Umum AWS

Contoh kode berikut membuat instance keyring penemuan dengan filter AWS KMS penemuan yang membatasi kunci KMS yang dapat digunakan SDK Enkripsi AWS Database untuk yang ada di partisi dan akun contoh. `aws` `111122223333`

Sebelum menggunakan kode ini, ganti contoh Akun AWS dan nilai partisi dengan nilai yang valid untuk Anda Akun AWS dan partisi. Jika kunci KMS Anda berada di Wilayah Tiongkok, gunakan nilai `aws-cn` partisi. Jika kunci KMS Anda masuk AWS GovCloud (US) Regions, gunakan nilai `aws-us-gov` partisi. Untuk yang lainnya Wilayah AWS, gunakan nilai `aws` partisi.

Java

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

```
        .build();
IKeyring decryptKeyring =
 matProv.CreateAwsKmsMrkDiscoveryMultiKeyring(createAwsKmsMrkDiscoveryMultiKeyringInput);
```

C# / .NET

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

Rust

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

// Create the discovery keyring
let decrypt_keyring = mpl
    .create_aws_kms_mrk_discovery_multi_keyring()
    .discovery_filter(discovery_filter)
    .send()
    .await?;
```

Menggunakan AWS KMS keyring penemuan regional

Keyring penemuan AWS KMS regional adalah keyring yang tidak menentukan kunci ARNs KMS. Sebaliknya, ini memungkinkan SDK Enkripsi AWS Database untuk mendekripsi hanya menggunakan kunci KMS pada khususnya. Wilayah AWS

Saat mendekripsi dengan keyring penemuan AWS KMS regional, SDK Enkripsi AWS Database mendekripsi kunci data terenkripsi yang dienkripsi di bawah kunci yang ditentukan. AWS KMS key Wilayah AWS Agar berhasil, penelepon harus memiliki `kms:Decrypt` izin setidaknya satu dari yang AWS KMS keys ditentukan Wilayah AWS yang mengenkripsi kunci data.

Seperti keyrings penemuan lainnya, keyring penemuan regional tidak berpengaruh pada enkripsi. Ini hanya berfungsi saat mendekripsi bidang terenkripsi. Jika Anda menggunakan keyring penemuan regional dalam multi-keyring yang digunakan untuk mengenkripsi dan mendekripsi, ini hanya efektif saat mendekripsi. Jika Anda menggunakan keyring penemuan Multi-wilayah untuk mengenkripsi data, sendiri atau dalam multi-keyring, operasi enkripsi gagal.

> **Important**
>
> Jika Anda menyertakan keyring penemuan AWS KMS regional dalam multi-keyring dekripsi, keyring penemuan regional mengesampingkan semua batasan kunci KMS yang ditentukan oleh gantungan kunci lain di multi-keyring. Multi-keyring berperilaku seperti keyring yang paling tidak membatasi. Keyring AWS KMS penemuan tidak berpengaruh pada enkripsi saat digunakan sendiri atau dalam multi-keyring.

Keyring penemuan regional di SDK Enkripsi AWS Database mencoba mendekripsi hanya dengan kunci KMS di Wilayah yang ditentukan. Saat Anda menggunakan keyring penemuan, Anda mengonfigurasi Wilayah pada AWS KMS klien. Implementasi SDK Enkripsi AWS Database ini tidak memfilter kunci KMS menurut Wilayah, tetapi AWS KMS akan gagal dalam permintaan dekripsi untuk kunci KMS di luar Wilayah yang ditentukan.

Jika Anda menggunakan keyring penemuan, sebaiknya gunakan filter penemuan untuk membatasi kunci KMS yang digunakan dalam dekripsi ke kunci yang ditentukan dan partisi. Akun AWS

Misalnya, kode berikut membuat keyring penemuan AWS KMS regional dengan filter penemuan. Keyring ini membatasi SDK Enkripsi AWS Database ke kunci KMS di akun 111122223333 di Wilayah AS Barat (Oregon) (us-west-2).

Java

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

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

C# / .NET

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

Rust

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

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

AWS KMS Gantungan kunci hierarkis

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

> **Note**
>
> Mulai 24 Juli 2023, kunci cabang yang dibuat selama pratinjau pengembang tidak didukung. Buat kunci cabang baru untuk terus menggunakan toko kunci yang Anda buat selama pratinjau pengembang.

Dengan keyring AWS KMS Hierarkis, Anda dapat melindungi materi kriptografi Anda di bawah kunci KMS enkripsi simetris tanpa menelepon AWS KMS setiap kali Anda mengenkripsi atau mendekripsi catatan. Ini adalah pilihan yang baik untuk aplikasi yang perlu meminimalkan panggilan ke AWS KMS, dan aplikasi yang dapat menggunakan kembali beberapa materi kriptografi tanpa melanggar persyaratan keamanan mereka.

Hierarchical keyring adalah solusi caching materi kriptografi yang mengurangi jumlah AWS KMS panggilan dengan menggunakan kunci cabang yang AWS KMS dilindungi yang disimpan dalam tabel Amazon DynamoDB, dan kemudian secara lokal menyimpan materi kunci cabang yang digunakan dalam operasi enkripsi dan dekripsi. Tabel DynamoDB berfungsi sebagai penyimpanan kunci yang mengelola dan melindungi kunci cabang. Ini menyimpan kunci cabang aktif dan semua versi sebelumnya dari kunci cabang. Kunci cabang aktif adalah versi kunci cabang terbaru. Keyring Hierarkis menggunakan kunci enkripsi data unik untuk setiap permintaan enkripsi dan mengenkripsi setiap kunci enkripsi data dengan kunci pembungkus unik yang berasal dari kunci cabang aktif. Keyring Hierarkis tergantung pada hierarki yang ditetapkan antara kunci cabang aktif dan kunci pembungkus turunannya.

Keyring Hierarkis biasanya menggunakan setiap versi kunci cabang untuk memenuhi beberapa permintaan. Tetapi Anda mengontrol sejauh mana kunci cabang aktif digunakan kembali dan menentukan seberapa sering kunci cabang aktif diputar. Versi aktif dari kunci cabang tetap aktif sampai Anda memutarnya. Versi sebelumnya dari kunci cabang aktif tidak akan digunakan untuk melakukan operasi enkripsi, tetapi masih dapat ditanyakan dan digunakan dalam operasi dekripsi.

Ketika Anda membuat instance keyring Hierarchical, itu membuat cache lokal. Anda menentukan batas cache yang menentukan jumlah waktu maksimum materi kunci cabang disimpan dalam cache

lokal sebelum kedaluwarsa dan dikeluarkan dari cache. Hierarchical keyring membuat satu AWS KMS panggilan untuk mendekripsi kunci cabang dan merakit materi kunci cabang saat pertama kali a `branch-key-id` ditentukan dalam suatu operasi. Kemudian, materi kunci cabang disimpan dalam cache lokal dan digunakan kembali untuk semua operasi enkripsi dan dekripsi yang menentukan itu `branch-key-id` sampai batas cache berakhir. Menyimpan materi kunci cabang di cache lokal mengurangi AWS KMS panggilan. Misalnya, pertimbangkan batas cache 15 menit. Jika Anda melakukan 10.000 operasi enkripsi dalam batas cache tersebut, AWS KMS keyring tradisional perlu melakukan 10.000 AWS KMS panggilan untuk memenuhi 10.000 operasi enkripsi. Jika Anda memiliki satu aktif`branch-key-id`, keyring Hierarkis hanya perlu membuat satu AWS KMS panggilan untuk memenuhi 10.000 operasi enkripsi.

Cache lokal memisahkan bahan enkripsi dari bahan dekripsi. Materi enkripsi dirakit dari kunci cabang aktif dan digunakan kembali untuk semua operasi enkripsi hingga batas cache berakhir. Materi dekripsi dirakit dari ID kunci cabang dan versi yang diidentifikasi dalam metadata bidang terenkripsi, dan digunakan kembali untuk semua operasi dekripsi yang terkait dengan ID kunci cabang dan versi hingga batas cache berakhir. Cache lokal dapat menyimpan beberapa versi kunci cabang yang sama pada satu waktu. Ketika cache lokal dikonfigurasi untuk menggunakan abranch key ID supplier, itu juga dapat menyimpan materi kunci cabang dari beberapa kunci cabang aktif pada satu waktu.

> **Note**
>
> Semua penyebutan keyring Hierarkis dalam SDK Enkripsi AWS Database mengacu pada keyring Hierarkis. AWS KMS

Topik

- Cara kerjanya
- Prasyarat
- Izin yang diperlukan
- Pilih cache
- Buat keyring Hierarkis
- Menggunakan keyring Hierarkis untuk enkripsi yang dapat dicari

Cara kerjanya

Panduan berikut menjelaskan bagaimana keyring Hierarkis merakit bahan enkripsi dan dekripsi, dan panggilan berbeda yang dibuat oleh keyring untuk mengenkripsi dan mendekripsi operasi. Untuk detail teknis tentang derivasi kunci pembungkus dan proses enkripsi kunci data plaintext, lihat Detail teknis keyring AWS KMS hierarkis.

Enkripsi dan tandatangani

Panduan berikut menjelaskan bagaimana keyring Hierarkis merakit bahan enkripsi dan memperoleh kunci pembungkus yang unik.

1. Metode enkripsi meminta keyring Hierarkis untuk materi enkripsi. Keyring menghasilkan kunci data plaintext, lalu memeriksa untuk melihat apakah ada materi kunci cabang yang valid di cache lokal untuk menghasilkan kunci pembungkus. Jika ada materi kunci cabang yang valid, keyring dilanjutkan ke Langkah 4.
2. Jika tidak ada materi kunci cabang yang valid, keyring Hierarkis menanyakan penyimpanan kunci untuk kunci cabang aktif.
    a. Key store memanggil AWS KMS untuk mendekripsi kunci cabang aktif dan mengembalikan kunci cabang aktif plaintext. Data yang mengidentifikasi kunci cabang aktif diserialisasi untuk memberikan data otentikasi tambahan (AAD) dalam panggilan dekripsi ke. AWS KMS
    b. Toko kunci mengembalikan kunci cabang plaintext dan data yang mengidentifikasinya, seperti versi kunci cabang.
3. Hierarchical keyring merakit materi kunci cabang (kunci cabang plaintext dan versi kunci cabang) dan menyimpan salinannya di cache lokal.
4. Keyring Hierarchical memperoleh kunci pembungkus unik dari kunci cabang plaintext dan garam acak 16-byte. Ini menggunakan kunci pembungkus turunan untuk mengenkripsi salinan kunci data teks biasa.

Metode enkripsi menggunakan bahan enkripsi untuk mengenkripsi dan menandatangani catatan. Untuk informasi selengkapnya tentang cara catatan dienkripsi dan ditandatangani di SDK Enkripsi AWS Database, lihat Mengenkripsi dan menandatangani.

Dekripsi dan verifikasi

Panduan berikut menjelaskan bagaimana keyring Hierarkis merakit bahan dekripsi dan mendekripsi kunci data terenkripsi.

1. Metode dekripsi mengidentifikasi kunci data terenkripsi dari bidang deskripsi material dari catatan terenkripsi, dan meneruskannya ke keyring Hierarkis.
2. Hierarchical keyring deserialisasi data yang mengidentifikasi kunci data terenkripsi, termasuk versi kunci cabang, garam 16-byte, dan informasi lain yang menjelaskan bagaimana kunci data dienkripsi.

   Untuk informasi selengkapnya, lihat AWS KMS Rincian teknis keyring hierarkis.
3. Keyring hierarkis memeriksa untuk melihat apakah ada materi kunci cabang yang valid di cache lokal yang cocok dengan versi kunci cabang yang diidentifikasi pada Langkah 2. Jika ada materi kunci cabang yang valid, keyring dilanjutkan ke Langkah 6.
4. Jika tidak ada materi kunci cabang yang valid, keyring Hierarkis menanyakan penyimpanan kunci untuk kunci cabang yang cocok dengan versi kunci cabang yang diidentifikasi pada Langkah 2.

   a. Key store memanggil AWS KMS untuk mendekripsi kunci cabang dan mengembalikan kunci cabang aktif plaintext. Data yang mengidentifikasi kunci cabang aktif diserialisasi untuk memberikan data otentikasi tambahan (AAD) dalam panggilan dekripsi ke. AWS KMS

   b. Toko kunci mengembalikan kunci cabang plaintext dan data yang mengidentifikasinya, seperti versi kunci cabang.
5. Hierarchical keyring merakit materi kunci cabang (kunci cabang plaintext dan versi kunci cabang) dan menyimpan salinannya di cache lokal.
6. Keyring Hierarchical menggunakan bahan kunci cabang yang dirakit dan garam 16-byte yang diidentifikasi pada Langkah 2 untuk mereproduksi kunci pembungkus unik yang mengenkripsi kunci data.
7. Keyring Hierarkis menggunakan kunci pembungkus yang direproduksi untuk mendekripsi kunci data dan mengembalikan kunci data plaintext.

Metode dekripsi menggunakan bahan dekripsi dan kunci data teks biasa untuk mendekripsi dan memverifikasi catatan. Untuk informasi selengkapnya tentang cara rekaman didekripsi dan diverifikasi di SDK Enkripsi AWS Database, lihat Mendekripsi dan memverifikasi.

Prasyarat

Sebelum Anda membuat dan menggunakan keyring Hierarkis, pastikan prasyarat berikut terpenuhi.

- Anda, atau administrator toko kunci Anda, telah membuat toko kunci dan membuat setidaknya satu kunci cabang aktif.
- Anda telah mengonfigurasi tindakan penyimpanan utama Anda.

> **Note**
>
> Cara Anda mengonfigurasi tindakan penyimpanan kunci menentukan operasi apa yang dapat Anda lakukan dan kunci KMS apa yang dapat digunakan oleh keyring Hierarkis. Untuk informasi selengkapnya, lihat Tindakan penyimpanan kunci.

- Anda memiliki AWS KMS izin yang diperlukan untuk mengakses dan menggunakan kunci penyimpanan dan cabang kunci. Untuk informasi selengkapnya, lihat the section called “Izin yang diperlukan”.
- Anda telah meninjau jenis cache yang didukung dan mengonfigurasi jenis cache yang paling sesuai dengan kebutuhan Anda. Untuk informasi selengkapnya, silakan lihat the section called “Pilih cache”

Izin yang diperlukan

SDK Enkripsi AWS Database tidak memerlukan Akun AWS dan tidak bergantung pada apa pun Layanan AWS. Namun, untuk menggunakan keyring Hierarkis, Anda memerlukan izin minimum Akun AWS dan berikut pada enkripsi simetris di AWS KMS key toko kunci Anda.

- Untuk mengenkripsi dan mendekripsi data dengan keyring Hierarkis, Anda memerlukan KMS: Decrypt.
- Untuk membuat dan memutar kunci cabang, Anda memerlukan kms: GenerateDataKeyWithoutPlaintext dan kms:. ReEncrypt

Untuk informasi selengkapnya tentang mengontrol akses ke kunci cabang dan penyimpanan kunci Anda, lihatthe section called “Menerapkan izin yang paling tidak diistimewakan”.

Pilih cache

Keyring Hierarkis mengurangi jumlah panggilan yang dilakukan AWS KMS dengan menyimpan materi kunci cabang secara lokal yang digunakan dalam operasi enkripsi dan dekripsi. Sebelum Anda membuat keyring Hierarkis Anda, Anda perlu memutuskan jenis cache yang ingin Anda gunakan.

Anda dapat menggunakan cache default atau menyesuaikan cache agar sesuai dengan kebutuhan Anda.

Keyring Hierarkis mendukung jenis cache berikut:

- the section called “Cache default”
- the section called “MultiThreaded tembolok”
- the section called “StormTracking tembolok”
- the section called “Cache bersama”

Cache default

Untuk sebagian besar pengguna, cache Default memenuhi persyaratan threading mereka. Cache Default dirancang untuk mendukung lingkungan yang sangat multithreaded. Ketika entri materi kunci cabang kedaluwarsa, cache Default mencegah beberapa utas memanggil AWS KMS dengan memberi tahu satu utas bahwa entri materi kunci cabang akan kedaluwarsa 10 detik sebelumnya. Ini memastikan bahwa hanya satu utas yang mengirimkan permintaan AWS KMS untuk menyegarkan cache.

Default dan StormTracking cache mendukung model threading yang sama, tetapi Anda hanya perlu menentukan kapasitas entri untuk menggunakan cache Default. Untuk kustomisasi cache yang lebih terperinci, gunakan file. the section called “StormTracking tembolok”

Kecuali Anda ingin menyesuaikan jumlah entri materi kunci cabang yang dapat disimpan di cache lokal, Anda tidak perlu menentukan jenis cache saat Anda membuat keyring Hierarkis. Jika Anda tidak menentukan jenis cache, keyring Hierarkis menggunakan jenis cache Default dan menetapkan kapasitas entri ke 1000.

Untuk menyesuaikan cache Default, tentukan nilai berikut:

- Kapasitas entri: membatasi jumlah entri materi kunci cabang yang dapat disimpan di cache lokal.

Java

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

C# / .NET

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

Rust

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

MultiThreaded tembolok

MultiThreaded Cache aman digunakan di lingkungan multithreaded, tetapi tidak menyediakan fungsionalitas apa pun untuk meminimalkan atau panggilan Amazon AWS KMS DynamoDB. Akibatnya, ketika entri materi kunci cabang kedaluwarsa, semua utas akan diberitahukan pada saat yang sama. Ini dapat menghasilkan beberapa AWS KMS panggilan untuk menyegarkan cache.

Untuk menggunakan MultiThreaded cache, tentukan nilai berikut:

- Kapasitas entri: membatasi jumlah entri materi kunci cabang yang dapat disimpan di cache lokal.
- Ukuran ekor pemangkasan entri: menentukan jumlah entri yang akan dipangkas jika kapasitas masuk tercapai.

Java

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

C# / .NET

```
CacheType multithreadedCache = new CacheType
```

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

Rust

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

StormTracking tembolok

StormTracking Cache dirancang untuk mendukung lingkungan yang sangat multithreaded. Ketika entri materi kunci cabang kedaluwarsa, StormTracking cache mencegah beberapa utas memanggil AWS KMS dengan memberi tahu satu utas bahwa entri materi kunci cabang akan kedaluwarsa sebelumnya. Ini memastikan bahwa hanya satu utas yang mengirimkan permintaan AWS KMS untuk menyegarkan cache.

Untuk menggunakan StormTracking cache, tentukan nilai berikut:

- Kapasitas entri: membatasi jumlah entri materi kunci cabang yang dapat disimpan di cache lokal.

  Nilai default: 1000 entri
- Ukuran ekor pemangkasan entri: menentukan jumlah entri bahan kunci cabang untuk dipangkas sekaligus.

  Nilai default: 1 entri
- Masa tenggang: mendefinisikan jumlah detik sebelum kedaluwarsa bahwa upaya untuk menyegarkan materi kunci cabang dilakukan.

  Nilai default: 10 detik
- Interval rahmat: mendefinisikan jumlah detik antara upaya untuk menyegarkan materi kunci cabang.

Nilai default: 1 detik

- Fan out: mendefinisikan jumlah upaya simultan yang dapat dilakukan untuk menyegarkan materi kunci cabang.

  Nilai default: 20 upaya

- In flight time to live (TTL): mendefinisikan jumlah detik hingga upaya untuk menyegarkan materi kunci cabang habis waktu. Setiap kali cache kembali `NoSuchEntry` sebagai respons terhadap a`GetCacheEntry`, kunci cabang tersebut dianggap dalam penerbangan sampai kunci yang sama ditulis dengan `PutCache` entri.

  Nilai default: 10 detik

- Tidur: mendefinisikan jumlah detik bahwa sebuah utas harus tidur jika `fanOut` terlampaui.

  Nilai default: 20 milidetik

Java

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

C# / .NET

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

```
        InFlightTTL = 10,
        SleepMilli = 20
    }
};
```

Rust

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

Cache bersama

Secara default, keyring Hierarkis membuat cache lokal baru setiap kali Anda membuat instance keyring. Namun, cache Bersama dapat membantu menghemat memori dengan memungkinkan Anda berbagi cache di beberapa gantungan kunci Hierarkis. Daripada membuat cache materi kriptografi baru untuk setiap keyring Hierarkis yang Anda buat instance, cache Bersama hanya menyimpan satu cache dalam memori, yang dapat digunakan oleh semua gantungan kunci Hierarkis yang mereferensikannya. Cache bersama membantu mengoptimalkan penggunaan memori dengan menghindari duplikasi materi kriptografi di seluruh keyrings. Sebagai gantinya, gantungan kunci Hierarkis dapat mengakses cache dasar yang sama, mengurangi jejak memori secara keseluruhan.

Saat Anda membuat cache Bersama, Anda masih menentukan jenis cache. Anda dapat menentukanthe section called “Cache default”,the section called “MultiThreaded tembolok”, atau the section called “StormTracking tembolok” sebagai jenis cache, atau mengganti cache kustom yang kompatibel.

Partisi

Beberapa keyrings Hierarkis dapat menggunakan satu cache Bersama. Saat Anda membuat keyring Hierarkis dengan cache Bersama, Anda dapat menentukan ID partisi opsional. ID

partisi membedakan keyring Hierarkis mana yang menulis ke cache. Jika dua keyrings hirarkis mereferensikan ID partisi yang samalogical key store name, dan ID kunci cabang kedua keyrings akan berbagi entri cache yang sama dalam cache. Jika Anda membuat dua gantungan kunci Hierarkis dengan cache Bersama yang sama, tetapi partisi yang berbeda IDs, setiap keyring hanya akan mengakses entri cache dari partisi yang ditunjuk sendiri dalam cache Bersama. Partisi bertindak sebagai divisi logis dalam cache bersama, memungkinkan setiap keyring Hierarkis beroperasi secara independen pada partisi yang ditunjuk sendiri, tanpa mengganggu data yang disimpan di partisi lain.

Jika Anda bermaksud untuk menggunakan kembali atau berbagi entri cache di partisi, Anda harus menentukan ID partisi Anda sendiri. Saat Anda meneruskan ID partisi ke keyring Hierarkis Anda, keyring dapat menggunakan kembali entri cache yang sudah ada di cache Bersama, daripada harus mengambil dan mengotorisasi ulang materi kunci cabang lagi. Jika Anda tidak menentukan ID partisi, ID partisi unik secara otomatis ditetapkan ke keyring setiap kali Anda membuat instance keyring Hierarkis.

Prosedur berikut menunjukkan cara membuat cache Bersama dengan tipe cache Default dan meneruskannya ke keyring Hierarkis.

1. Buat `CryptographicMaterialsCache` (CMC) menggunakan Material Providers Library (MPL).

   Java

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

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

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

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

C# / .NET

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

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

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

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

Rust

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

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

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

2. Buat `CacheType` objek untuk cache Bersama.

   Lulus yang `sharedCryptographicMaterialsCache` Anda buat di Langkah 1 ke `CacheType` objek baru.

Java

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

C# / .NET

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

Rust

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

3. Lewati `sharedCache` objek dari Langkah 2 ke keyring Hierarkis Anda.

Saat Anda membuat keyring Hierarkis dengan cache Bersama, Anda dapat secara opsional menentukan entri `partitionID` untuk berbagi cache di beberapa gantungan kunci Hierarkis. Jika Anda tidak menentukan ID partisi, keyring Hierarkis secara otomatis menetapkan keyring ID partisi unik.

> **Note**
>
> Keyring Hierarkis Anda akan berbagi entri cache yang sama dalam cache Bersama jika Anda membuat dua atau lebih keyrings yang mereferensikan ID partisi yang samalogical key store name, dan ID kunci cabang. Jika Anda tidak ingin beberapa keyrings berbagi entri cache yang sama, Anda harus menggunakan ID partisi unik untuk setiap keyring Hierarkis.

Contoh berikut membuat keyring Hierarkis denganbranch key ID supplier, dan batas cache 600 detik. Untuk informasi selengkapnya tentang nilai yang ditentukan dalam konfigurasi keyring Hierarkis berikut, lihat. the section called “Buat keyring Hierarkis”

Java

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

C# / .NET

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

Rust

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

```
    .ttl_seconds(600)
    .partition_id(partition_id.clone())
    .send()
    .await?;
```

Buat keyring Hierarkis

Untuk membuat keyring Hierarkis, Anda harus memberikan nilai-nilai berikut:

- Nama toko kunci

  Nama tabel DynamoDB yang Anda, atau administrator toko utama Anda, dibuat untuk berfungsi sebagai toko kunci Anda.

-

  Batas waktu cache untuk hidup (TTL)

  Jumlah waktu dalam hitungan detik entri materi kunci cabang dalam cache lokal dapat digunakan sebelum kedaluwarsa. Batas cache TTL menentukan seberapa sering klien memanggil AWS KMS untuk mengotorisasi penggunaan kunci cabang. Nilai ini harus lebih besar dari nol. Setelah batas cache TTL berakhir, entri tidak pernah disajikan, dan akan diusir dari cache lokal.

- Pengidentifikasi kunci cabang

  Anda dapat mengonfigurasi secara statis `branch-key-id` yang mengidentifikasi satu kunci cabang aktif di toko kunci Anda, atau memberikan pemasok ID kunci cabang.

  Pemasok ID kunci cabang menggunakan bidang yang disimpan dalam konteks enkripsi untuk menentukan kunci cabang mana yang diperlukan untuk mendekripsi catatan. Secara default, hanya partisi dan kunci pengurutan yang disertakan dalam konteks enkripsi. Namun, Anda dapat menggunakan tindakan `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` kriptografi untuk memasukkan bidang tambahan dalam konteks enkripsi.

  Kami sangat menyarankan menggunakan pemasok ID kunci cabang untuk database multitenant di mana setiap penyewa memiliki kunci cabang mereka sendiri. Anda dapat menggunakan pemasok ID kunci cabang untuk membuat nama yang ramah IDs untuk kunci cabang Anda agar mudah mengenali ID kunci cabang yang benar untuk penyewa tertentu.

Misalnya, nama ramah memungkinkan Anda merujuk ke kunci cabang sebagai `tenant1` gantinya`b3f61619-4d35-48ad-a275-050f87e15122`.

Untuk operasi dekripsi, Anda dapat mengonfigurasi secara statis satu keyring Hierarkis untuk membatasi dekripsi ke penyewa tunggal, atau Anda dapat menggunakan pemasok ID kunci cabang untuk mengidentifikasi penyewa mana yang bertanggung jawab untuk mendekripsi catatan.

- (Opsional) Sebuah cache

  Jika Anda ingin menyesuaikan jenis cache atau jumlah entri materi kunci cabang yang dapat disimpan di cache lokal, tentukan jenis cache dan kapasitas entri saat Anda menginisialisasi keyring.

  Keyring Hierarkis mendukung jenis cache berikut: Default,, MultiThreaded StormTracking, dan Shared. Untuk informasi selengkapnya dan contoh yang menunjukkan cara menentukan setiap jenis cache, lihatthe section called “Pilih cache”.

  Jika Anda tidak menentukan cache, keyring Hierarkis secara otomatis menggunakan jenis cache Default dan menetapkan kapasitas entri ke 1000.

- (Opsional) Sebuah ID partisi

  Jika Anda menentukanthe section called “Cache bersama”, Anda dapat secara opsional menentukan ID partisi. ID partisi membedakan keyring Hierarkis mana yang menulis ke cache. Jika Anda bermaksud untuk menggunakan kembali atau berbagi entri cache di partisi, Anda harus menentukan ID partisi Anda sendiri. Anda dapat menentukan string apa pun untuk ID partisi. Jika Anda tidak menentukan ID partisi, ID partisi unik secara otomatis ditetapkan ke keyring saat pembuatan.

  Untuk informasi selengkapnya, lihat Partitions.

  > **Note**
  >
  > Keyring Hierarkis Anda akan berbagi entri cache yang sama dalam cache Bersama jika Anda membuat dua atau lebih keyrings yang mereferensikan ID partisi yang samalogical key store name, dan ID kunci cabang. Jika Anda tidak ingin beberapa keyrings berbagi entri cache yang sama, Anda harus menggunakan ID partisi unik untuk setiap keyring Hierarkis.

- (Opsional) Daftar Token Hibah

Jika Anda mengontrol akses ke kunci KMS di keyring Hierarkis Anda dengan hibah, Anda harus menyediakan semua token hibah yang diperlukan saat Anda menginisialisasi keyring.

Buat keyring Hierarkis dengan ID kunci cabang statis

Contoh berikut menunjukkan cara membuat keyring Hierarkis dengan ID kunci cabang statis, TTLthe section called “Cache default”, dan batas cache 600 detik.

Java

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

C# / .NET

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

Rust

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

let hierarchical_keyring = mpl
    .create_aws_kms_hierarchical_keyring()
    .branch_key_id(branch_key_id)
```

```
    .key_store(branch_key_store_name)
    .ttl_seconds(600)
    .send()
    .await?;
```

Buat keyring Hierarkis dengan pemasok ID kunci cabang

Prosedur berikut menunjukkan cara membuat keyring Hierarkis dengan pemasok ID kunci cabang.

1. Buat pemasok ID kunci cabang

   Contoh berikut membuat nama ramah untuk dua kunci cabang yang dibuat pada Langkah 1, dan panggilan `CreateDynamoDbEncryptionBranchKeyIdSupplier` untuk membuat pemasok ID kunci cabang dengan AWS Database Encryption SDK untuk klien DynamoDB.

   Java

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

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

   C# / .NET

   ```
   // Create friendly names for each branch-key-id
    class ExampleBranchKeyIdSupplier : DynamoDbKeyBranchKeyIdSupplierBase {
   ```

```
    private String _branchKeyIdForTenant1;
    private String _branchKeyIdForTenant2;

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

Rust

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

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

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

let branch_key_id_supplier = dbesdk
    .create_dynamo_db_encryption_branch_key_id_supplier()
    .ddb_key_branch_key_id_supplier(supplier)
    .send()
```

```
    .await?
    .branch_key_id_supplier
    .unwrap();
```

2. Buat keyring Hierarkis

   Contoh berikut menginisialisasi keyring Hierarkis dengan pemasok ID kunci cabang yang dibuat pada Langkah 1, batas cache TLL 600 detik, dan ukuran cache maksimum 1000.

   Java

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

   C# / .NET

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

Rust

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

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

Menggunakan keyring Hierarkis untuk enkripsi yang dapat dicari

Enkripsi yang dapat dicari memungkinkan Anda untuk mencari catatan terenkripsi tanpa mendekripsi seluruh database. Ini dicapai dengan mengindeks nilai plaintext dari bidang terenkripsi dengan suar. Untuk mengimplementasikan enkripsi yang dapat dicari, Anda harus menggunakan keyring Hierarkis.

`CreateKey`Operasi penyimpanan kunci menghasilkan kunci cabang dan kunci suar. Kunci cabang digunakan dalam enkripsi catatan dan operasi dekripsi. Kunci suar digunakan untuk menghasilkan suar.

Kunci cabang dan kunci suar dilindungi oleh yang sama dengan AWS KMS key yang Anda tentukan saat membuat layanan toko kunci Anda. Setelah panggilan `CreateKey` operasi AWS KMS untuk menghasilkan kunci cabang, ia memanggil kms: GenerateDataKeyWithoutPlaintext kedua kalinya untuk menghasilkan kunci suar menggunakan permintaan berikut.

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

```
}
```

Setelah menghasilkan kedua kunci, `CreateKey` operasi memanggil ddb: TransactWriteItems untuk menulis dua item baru yang akan mempertahankan kunci cabang dan kunci suar di toko kunci cabang Anda.

Saat Anda mengonfigurasi suar standar, SDK Enkripsi AWS Database menanyakan penyimpanan kunci untuk kunci suar. Kemudian, ia menggunakan fungsi derivasi extract-and-expand kunci berbasis HMAC (HKDF) untuk menggabungkan kunci suar dengan nama suar standar untuk membuat kunci HMAC untuk suar yang diberikan.

Tidak seperti kunci cabang, hanya ada satu versi kunci suar per `branch-key-id` di toko kunci. Kunci suar tidak pernah diputar.

Mendefinisikan sumber kunci suar Anda

Saat Anda menentukan versi beacon untuk beacon standar dan gabungan Anda, Anda harus mengidentifikasi kunci suar dan menentukan batas waktu cache untuk hidup (TTL) untuk materi kunci suar. Materi kunci suar disimpan dalam cache lokal terpisah dari kunci cabang. Cuplikan berikut menunjukkan bagaimana mendefinisikan untuk database penyewa `keySource` tunggal. Identifikasi kunci suar Anda dengan yang `branch-key-id` terkait dengannya.

Java

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

C# / .NET

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

```
}
```

Rust

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

Mendefinisikan sumber suar dalam database multitenant

Jika Anda memiliki database multitenant, Anda harus menentukan nilai-nilai berikut saat mengkonfigurasi. `keySource`

- keyFieldName

  Mendefinisikan nama bidang yang menyimpan yang `branch-key-id` terkait dengan kunci suar yang digunakan untuk menghasilkan suar untuk penyewa tertentu. `keyFieldName`Bisa berupa string apa saja, tetapi harus unik untuk semua bidang lain di database Anda. Saat Anda menulis catatan baru ke database Anda, `branch-key-id` yang mengidentifikasi kunci suar yang digunakan untuk menghasilkan suar apa pun untuk catatan itu disimpan di bidang ini. Anda harus menyertakan bidang ini dalam kueri suar Anda dan mengidentifikasi bahan kunci suar yang sesuai yang diperlukan untuk menghitung ulang suar. Untuk informasi selengkapnya, lihat Menanyakan beacon dalam database multitenant.

- CacheTTL

  Jumlah waktu dalam hitungan detik entri bahan kunci suar dalam cache suar lokal dapat digunakan sebelum kedaluwarsa. Nilai ini harus lebih besar dari nol. Ketika batas cache TTL kedaluwarsa, entri diusir dari cache lokal.

- (Opsional) Sebuah cache

Jika Anda ingin menyesuaikan jenis cache atau jumlah entri materi kunci cabang yang dapat disimpan di cache lokal, tentukan jenis cache dan kapasitas entri saat Anda menginisialisasi keyring.

Keyring Hierarkis mendukung jenis cache berikut: Default,, MultiThreaded StormTracking, dan Shared. Untuk informasi selengkapnya dan contoh yang menunjukkan cara menentukan setiap jenis cache, lihatthe section called “Pilih cache”.

Jika Anda tidak menentukan cache, keyring Hierarkis secara otomatis menggunakan jenis cache Default dan menetapkan kapasitas entri ke 1000.

Contoh berikut membuat keyring Hierarkis dengan pemasok ID kunci cabang TLL batas cache 600 detik, dan kapasitas entri 1000.

Java

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

C# / .NET

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var keyringInput = new CreateAwsKmsHierarchicalKeyringInput
{
   KeyStore = keystore,
   BranchKeyIdSupplier = branchKeyIdSupplier,
   TtlSeconds = 600,
   Cache = new CacheType
   {
```

```
        Default = new DefaultCache { EntryCapacity = 1000 }
    }
};
var hierarchicalKeyring = matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);
```

Rust

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

AWS KMS Gantungan kunci ECDH

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Important

Gantungan kunci AWS KMS ECDH hanya tersedia dengan versi 1.5.0 atau yang lebih baru dari Perpustakaan Penyedia Material.

Gantungan kunci AWS KMS ECDH menggunakan kesepakatan kunci asimetris AWS KMS keysuntuk mendapatkan kunci pembungkus simetris bersama antara dua pihak. Pertama, keyring menggunakan algoritma perjanjian kunci Elliptic Curve Diffie-Hellman (ECDH) untuk mendapatkan rahasia bersama dari kunci pribadi di KMS key pair pengirim dan kunci publik penerima. Kemudian, keyring menggunakan rahasia bersama untuk mendapatkan kunci pembungkus bersama yang melindungi kunci enkripsi data Anda. Fungsi derivasi kunci yang digunakan SDK Enkripsi AWS Database (`KDF_CTR_HMAC_SHA384`) untuk mendapatkan kunci pembungkus bersama sesuai dengan rekomendasi NIST untuk derivasi kunci.

Fungsi derivasi kunci mengembalikan 64 byte bahan kunci. Untuk memastikan bahwa kedua belah pihak menggunakan materi kunci yang benar, SDK Enkripsi AWS Database menggunakan 32 byte pertama sebagai kunci komitmen dan 32 byte terakhir sebagai kunci pembungkus bersama. Saat mendekripsi, jika keyring tidak dapat mereproduksi kunci komitmen yang sama dan kunci pembungkus bersama yang disimpan di bidang deskripsi material dari catatan terenkripsi, operasi gagal. Misalnya, jika Anda mengenkripsi rekaman dengan keyring yang dikonfigurasi dengan kunci pribadi Alice dan kunci publik Bob, keyring yang dikonfigurasi dengan kunci pribadi Bob dan kunci publik Alice akan mereproduksi kunci komitmen yang sama dan kunci pembungkus bersama dan dapat mendekripsi catatan. Jika kunci publik Bob bukan dari key pair KMS, maka Bob dapat membuat keyring ECDH mentah untuk mendekripsi catatan.

Keyring AWS KMS ECDH mengenkripsi catatan dengan kunci simetris menggunakan AES-GCM. Kunci data kemudian dienkripsi dengan kunci pembungkus bersama turunan menggunakan AES-GCM. Setiap keyring AWS KMS ECDH hanya dapat memiliki satu kunci pembungkus bersama, tetapi Anda dapat menyertakan beberapa gantungan kunci AWS KMS ECDH, sendiri atau dengan gantungan kunci lainnya, dalam multi-keyring.

Topik

- Izin yang diperlukan untuk gantungan kunci AWS KMS ECDH
- Membuat keyring AWS KMS ECDH
- Membuat keyring AWS KMS penemuan ECDH

Izin yang diperlukan untuk gantungan kunci AWS KMS ECDH

SDK Enkripsi AWS Database tidak memerlukan AWS akun dan tidak bergantung pada AWS layanan apa pun. Namun, untuk menggunakan keyring AWS KMS ECDH, Anda memerlukan AWS akun dan izin minimum berikut pada keyring Anda. AWS KMS keys Izin bervariasi berdasarkan skema perjanjian kunci yang Anda gunakan.

- Untuk mengenkripsi dan mendekripsi catatan menggunakan skema perjanjian `KmsPrivateKeyToStaticPublicKey` kunci, Anda memerlukan kms: GetPublicKey dan kms: DeriveSharedSecret pada key pair KMS asimetris pengirim. Jika Anda langsung memberikan kunci publik yang dienkode DER pengirim saat membuat instance keyring, Anda hanya perlu DeriveSharedSecret izin kms: pada key pair KMS asimetris pengirim.
- Untuk mendekripsi catatan menggunakan skema perjanjian `KmsPublicKeyDiscovery` kunci, Anda memerlukan GetPublicKey izin kms: DeriveSharedSecret dan kms: pada key pair KMS asimetris yang ditentukan.

Membuat keyring AWS KMS ECDH

Untuk membuat keyring AWS KMS ECDH yang mengenkripsi dan mendekripsi data, Anda harus menggunakan skema perjanjian kunci. `KmsPrivateKeyToStaticPublicKey` Untuk menginisialisasi keyring AWS KMS ECDH dengan skema perjanjian `KmsPrivateKeyToStaticPublicKey` kunci, berikan nilai-nilai berikut:

- ID Pengirim AWS KMS key

  Harus mengidentifikasi asymmetric NIST recommended elliptic curve (ECC) KMS key pair dengan nilai. `KeyUsage` `KEY_AGREEMENT` Kunci pribadi pengirim digunakan untuk mendapatkan rahasia bersama.

- (Opsional) Kunci publik pengirim

  Harus berupa kunci publik X.509 yang dikodekan DER, juga dikenal sebagai `SubjectPublicKeyInfo` (SPKI), sebagaimana didefinisikan dalam RFC 5280.

  AWS KMS GetPublicKeyOperasi mengembalikan kunci publik dari key pair KMS asimetris dalam format yang dienkode DER yang diperlukan.

  Untuk mengurangi jumlah AWS KMS panggilan yang dilakukan keyring Anda, Anda dapat langsung memberikan kunci publik pengirim. Jika tidak ada nilai yang diberikan untuk kunci publik pengirim, keyring akan memanggil AWS KMS untuk mengambil kunci publik pengirim.

- Kunci publik penerima

  Anda harus memberikan kunci publik X.509 yang dikodekan DER penerima, juga dikenal sebagai `SubjectPublicKeyInfo` (SPKI), sebagaimana didefinisikan dalam RFC 5280.

  AWS KMS GetPublicKeyOperasi mengembalikan kunci publik dari key pair KMS asimetris dalam format yang dienkode DER yang diperlukan.

- Spesifikasi kurva

  Mengidentifikasi spesifikasi kurva elips dalam pasangan kunci yang ditentukan. Pasangan kunci pengirim dan penerima harus memiliki spesifikasi kurva yang sama.

  Nilai valid: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

- (Opsional) Daftar Token Hibah

Jika Anda mengontrol akses ke kunci KMS di keyring AWS KMS ECDH Anda dengan hibah, Anda harus memberikan semua token hibah yang diperlukan saat Anda menginisialisasi keyring.

C# / .NET

Contoh berikut membuat keyring AWS KMS ECDH dengan kunci KMS pengirim, kunci publik pengirim, dan kunci publik penerima. Contoh ini menggunakan `senderPublicKey` parameter opsional untuk menyediakan kunci publik pengirim. Jika Anda tidak memberikan kunci publik pengirim, keyring akan memanggil AWS KMS untuk mengambil kunci publik pengirim. Pasangan kunci pengirim dan penerima berada di `ECC_NIST_P256` kurva.

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

// Must be DER-encoded X.509 public keys
var BobPublicKey = new MemoryStream(new byte[] { });
var AlicePublicKey = new MemoryStream(new byte[] { });

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

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

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

Java

Contoh berikut membuat keyring AWS KMS ECDH dengan kunci KMS pengirim, kunci publik pengirim, dan kunci publik penerima. Contoh ini menggunakan `senderPublicKey` parameter opsional untuk menyediakan kunci publik pengirim. Jika Anda tidak memberikan kunci publik pengirim, keyring akan memanggil AWS KMS untuk mengambil kunci publik pengirim. Pasangan kunci pengirim dan penerima berada di `ECC_NIST_P256` kurva.

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

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

Rust

Contoh berikut membuat keyring AWS KMS ECDH dengan kunci KMS pengirim, kunci publik pengirim, dan kunci publik penerima. Contoh ini menggunakan `sender_public_key` parameter opsional untuk menyediakan kunci publik pengirim. Jika Anda tidak memberikan kunci publik pengirim, keyring akan memanggil AWS KMS untuk mengambil kunci publik pengirim.

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

```
let public_key_sender_utf8_bytes = parsed_public_key_file_content_sender.contents();

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

// Create KmsPrivateKeyToStaticPublicKeyInput
let kms_ecdh_static_configuration_input =
    KmsPrivateKeyToStaticPublicKeyInput::builder()
        .sender_kms_identifier(arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab)
        // Must be a UTF8 DER-encoded X.509 public key
        .sender_public_key(public_key_sender_utf8_bytes)
        // Must be a UTF8 DER-encoded X.509 public key
        .recipient_public_key(public_key_recipient_utf8_bytes)
        .build()?;

let kms_ecdh_static_configuration =
 KmsEcdhStaticConfigurations::KmsPrivateKeyToStaticPublicKey(kms_ecdh_static_configuration_i

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

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

Membuat keyring AWS KMS penemuan ECDH

Saat mendekripsi, ini adalah praktik terbaik untuk menentukan kunci yang dapat digunakan SDK Enkripsi AWS Database. Untuk mengikuti praktik terbaik ini, gunakan gantungan kunci AWS KMS ECDH dengan skema perjanjian `KmsPrivateKeyToStaticPublicKey` kunci. Namun, Anda juga dapat membuat keyring penemuan AWS KMS ECDH, yaitu keyring AWS KMS ECDH yang dapat

mendekripsi catatan apa pun di mana kunci publik dari key pair KMS yang ditentukan cocok dengan kunci publik penerima yang disimpan di bidang deskripsi material dari catatan terenkripsi.

> **Important**
>
> Ketika Anda mendekripsi catatan menggunakan skema perjanjian `KmsPublicKeyDiscovery` kunci, Anda menerima semua kunci publik, terlepas dari siapa yang memilikinya.

Untuk menginisialisasi keyring AWS KMS ECDH dengan skema perjanjian `KmsPublicKeyDiscovery` kunci, berikan nilai-nilai berikut:

- AWS KMS key ID Penerima

  Harus mengidentifikasi asymmetric NIST recommended elliptic curve (ECC) KMS key pair dengan nilai. `KeyUsage` `KEY_AGREEMENT`
- Spesifikasi kurva

  Mengidentifikasi spesifikasi kurva eliptik dalam key pair KMS penerima.

  Nilai valid: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`
- (Opsional) Daftar Token Hibah

  Jika Anda mengontrol akses ke kunci KMS di keyring AWS KMS ECDH Anda dengan hibah, Anda harus memberikan semua token hibah yang diperlukan saat Anda menginisialisasi keyring.

C# / .NET

Contoh berikut membuat keyring penemuan AWS KMS ECDH dengan key pair KMS pada kurva. `ECC_NIST_P256` Anda harus memiliki DeriveSharedSecret izin kms: GetPublicKey dan kms: pada key pair KMS yang ditentukan. Keyring ini dapat mendekripsi catatan apa pun di mana kunci publik dari key pair KMS yang ditentukan cocok dengan kunci publik penerima yang disimpan di bidang deskripsi material dari catatan terenkripsi.

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

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

```
var discoveryConfiguration = new KmsEcdhStaticConfigurations
{
    KmsPublicKeyDiscovery = new KmsPublicKeyDiscoveryInput
    {
        RecipientKmsIdentifier = "arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321"
    }

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

Java

Contoh berikut membuat keyring penemuan AWS KMS ECDH dengan key pair KMS pada kurva. ECC_NIST_P256 Anda harus memiliki DeriveSharedSecret izin kms: GetPublicKey dan kms: pada key pair KMS yang ditentukan. Keyring ini dapat mendekripsi catatan apa pun di mana kunci publik dari key pair KMS yang ditentukan cocok dengan kunci publik penerima yang disimpan di bidang deskripsi material dari catatan terenkripsi.

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

Rust

```
// Create KmsPublicKeyDiscoveryInput
```

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

let kms_ecdh_discovery_static_configuration =
 KmsEcdhStaticConfigurations::KmsPublicKeyDiscovery(kms_ecdh_discovery_static_configuration_

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

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

Gantungan kunci AES mentah

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

SDK Enkripsi AWS Database memungkinkan Anda menggunakan kunci simetris AES yang Anda berikan sebagai kunci pembungkus yang melindungi kunci data Anda. Anda perlu membuat, menyimpan, dan melindungi materi utama, sebaiknya dalam modul keamanan perangkat keras (HSM) atau sistem manajemen kunci. Gunakan keyring Raw AES saat Anda perlu memberikan kunci pembungkus dan mengenkripsi kunci data secara lokal atau offline.

Raw AES keyring mengenkripsi data dengan menggunakan algoritma AES-GCM dan kunci pembungkus yang Anda tentukan sebagai array byte. Anda hanya dapat menentukan satu kunci pembungkus di setiap keyring Raw AES, tetapi Anda dapat menyertakan beberapa gantungan kunci Raw AES, sendiri atau dengan gantungan kunci lainnya, dalam multi-keyring.

Ruang nama dan nama kunci

Untuk mengidentifikasi kunci AES dalam keyring, keyring Raw AES menggunakan namespace kunci dan nama kunci yang Anda berikan. Nilai-nilai ini bukan rahasia. Mereka muncul dalam teks biasa dalam deskripsi materi yang ditambahkan SDK Enkripsi AWS Database ke catatan. Sebaiknya gunakan namespace kunci HSM atau sistem manajemen kunci Anda dan nama kunci yang mengidentifikasi kunci AES dalam sistem itu.

> **Note**
>
> Namespace kunci dan nama kunci setara dengan kolom ID Penyedia (atau Penyedia) dan ID Kunci di kolom. `JceMasterKey`

Jika Anda membuat keyring yang berbeda untuk mengenkripsi dan mendekripsi bidang tertentu, namespace dan nilai nama sangat penting. Jika namespace kunci dan nama kunci dalam keyring dekripsi bukan kecocokan yang tepat dan peka huruf besar/kecil untuk namespace kunci dan nama kunci dalam keyring enkripsi, keyring dekripsi tidak digunakan, meskipun byte materi kunci identik.

Misalnya, Anda mungkin mendefinisikan keyring Raw AES dengan namespace `HSM_01` kunci dan nama kunci. `AES_256_012` Kemudian, Anda menggunakan keyring itu untuk mengenkripsi beberapa data. Untuk mendekripsi data tersebut, buat keyring Raw AES dengan namespace kunci, nama kunci, dan material kunci yang sama.

Contoh berikut menunjukkan cara membuat keyring Raw AES. `AESWrappingKey`Variabel mewakili materi utama yang Anda berikan.

Java

```
final CreateRawAesKeyringInput keyringInput = CreateRawAesKeyringInput.builder()
        .keyName("AES_256_012")
        .keyNamespace("HSM_01")
        .wrappingKey(AESWrappingKey)
        .wrappingAlg(AesWrappingAlg.ALG_AES256_GCM_IV12_TAG16)
        .build();
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
IKeyring rawAesKeyring = matProv.CreateRawAesKeyring(keyringInput);
```

C# / .NET

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

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

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

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

Rust

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

Gantungan kunci RSA mentah

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Raw RSA keyring melakukan enkripsi asimetris dan dekripsi kunci data dalam memori lokal dengan kunci publik dan pribadi RSA yang Anda berikan. Anda perlu membuat, menyimpan, dan melindungi kunci pribadi, sebaiknya dalam modul keamanan perangkat keras (HSM) atau sistem manajemen kunci. Fungsi enkripsi mengenkripsi kunci data di bawah kunci publik RSA. Fungsi dekripsi mendekripsi kunci data menggunakan kunci pribadi. Anda dapat memilih dari antara beberapa mode padding RSA.

Raw RSA keyring yang mengenkripsi dan mendekripsi harus menyertakan kunci publik asimetris dan private key pair. Namun, Anda dapat mengenkripsi data dengan keyring Raw RSA yang hanya memiliki kunci publik, dan Anda dapat mendekripsi data dengan keyring Raw RSA yang hanya memiliki kunci pribadi. Anda dapat menyertakan keyring Raw RSA apa pun dalam multi-keyring. Jika Anda mengonfigurasi keyring Raw RSA dengan kunci publik dan pribadi, pastikan bahwa mereka adalah bagian dari key pair yang sama.

Raw RSA keyring setara dengan dan berinteraksi dengan JceMasterKeyin AWS Encryption SDK for Java ketika mereka digunakan dengan kunci enkripsi asimetris RSA.

> **Note**
>
> Raw RSA keyring tidak mendukung kunci KMS asimetris. Untuk menggunakan kunci KMS RSA asimetris, buat keyring.AWS KMS

Ruang nama dan nama

Untuk mengidentifikasi materi kunci RSA dalam keyring, keyring Raw RSA menggunakan namespace kunci dan nama kunci yang Anda berikan. Nilai-nilai ini bukan rahasia. Mereka muncul dalam teks biasa dalam deskripsi materi yang ditambahkan SDK Enkripsi AWS Database ke catatan. Sebaiknya gunakan namespace kunci dan nama kunci yang mengidentifikasi key pair RSA (atau kunci privatnya) di HSM atau sistem manajemen kunci Anda.

> **Note**
>
> Namespace kunci dan nama kunci setara dengan kolom ID Penyedia (atau Penyedia) dan ID Kunci di kolom. `JceMasterKey`

Jika Anda membuat keyring yang berbeda untuk mengenkripsi dan mendekripsi catatan yang diberikan, namespace dan nilai nama sangat penting. Jika namespace kunci dan nama kunci dalam

keyring dekripsi bukan kecocokan yang tepat dan peka huruf besar/kecil untuk namespace kunci dan nama kunci dalam keyring enkripsi, keyring dekripsi tidak digunakan, bahkan jika kunci tersebut berasal dari key pair yang sama.

Namespace kunci dan nama kunci dari bahan kunci dalam enkripsi dan dekripsi keyrings harus sama apakah keyring berisi kunci publik RSA, kunci pribadi RSA, atau kedua kunci dalam key pair. Misalnya, Anda mengenkripsi data dengan keyring Raw RSA untuk kunci publik RSA dengan namespace kunci dan nama `HSM_01` kunci. `RSA_2048_06` Untuk mendekripsi data tersebut, buat keyring Raw RSA dengan kunci pribadi (atau key pair), dan namespace dan nama kunci yang sama.

Mode bantalan

Anda harus menentukan mode padding untuk keyring Raw RSA yang digunakan untuk enkripsi dan dekripsi, atau menggunakan fitur implementasi bahasa Anda yang menentukannya untuk Anda.

AWS Encryption SDK Mendukung mode padding berikut, tunduk pada kendala masing-masing bahasa. Kami merekomendasikan mode padding OAEP, terutama OAEP dengan SHA-256 dan dengan SHA-256 Padding. MGF1 Mode PKCS1padding hanya didukung untuk kompatibilitas mundur.

- OAEP dengan SHA-1 dan MGF1 dengan SHA-1 Padding
- OAEP dengan SHA-256 dan dengan SHA-256 Padding MGF1
- OAEP dengan SHA-384 dan dengan Padding SHA-384 MGF1
- OAEP dengan SHA-512 dan dengan Padding SHA-512 MGF1
- PKCS1 v1.5 Bantalan

Contoh Java berikut menunjukkan cara membuat keyring RSA Raw dengan kunci publik dan pribadi dari key pair RSA dan OAEP dengan SHA-256 dan dengan mode padding SHA-256. MGF1 `RSAPrivateKey`Variabel `RSAPublicKey` dan mewakili materi utama yang Anda berikan.

Java

```
final CreateRawRsaKeyringInput keyringInput = CreateRawRsaKeyringInput.builder()
        .keyName("RSA_2048_06")
        .keyNamespace("HSM_01")
        .paddingScheme(PaddingScheme.OAEP_SHA256_MGF1)
        .publicKey(RSAPublicKey)
        .privateKey(RSAPrivateKey)
```

```
        .build();
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
IKeyring rawRsaKeyring = matProv.CreateRawRsaKeyring(keyringInput);
```

C# / .NET

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

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

// Create the keyring input
var keyringInput = new CreateRawRsaKeyringInput
{
    KeyNamespace = keyNamespace,
    KeyName = keyName,
    PaddingScheme = PaddingScheme.OAEP_SHA512_MGF1,
    PublicKey = publicKey,
    PrivateKey = privateKey
};

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

Rust

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

```
    .await?;
```

Gantungan kunci ECDH mentah

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

> **Important**
>
> Raw ECDH keyring hanya tersedia dengan versi 1.5.0 dari Perpustakaan Penyedia Material.

Raw ECDH keyring menggunakan kurva elips pasangan kunci publik-pribadi yang Anda berikan untuk mendapatkan kunci pembungkus bersama antara dua pihak. Pertama, keyring memperoleh rahasia bersama menggunakan kunci pribadi pengirim, kunci publik penerima, dan algoritma perjanjian kunci Elliptic Curve Diffie-Hellman (ECDH). Kemudian, keyring menggunakan rahasia bersama untuk mendapatkan kunci pembungkus bersama yang melindungi kunci enkripsi data Anda. Fungsi derivasi kunci yang digunakan SDK Enkripsi AWS Database (`KDF_CTR_HMAC_SHA384`) untuk mendapatkan kunci pembungkus bersama sesuai dengan rekomendasi NIST untuk derivasi kunci.

Fungsi derivasi kunci mengembalikan 64 byte bahan kunci. Untuk memastikan bahwa kedua belah pihak menggunakan materi kunci yang benar, SDK Enkripsi AWS Database menggunakan 32 byte pertama sebagai kunci komitmen dan 32 byte terakhir sebagai kunci pembungkus bersama. Saat mendekripsi, jika keyring tidak dapat mereproduksi kunci komitmen yang sama dan kunci pembungkus bersama yang disimpan di bidang deskripsi material dari catatan terenkripsi, operasi gagal. Misalnya, jika Anda mengenkripsi rekaman dengan keyring yang dikonfigurasi dengan kunci pribadi Alice dan kunci publik Bob, keyring yang dikonfigurasi dengan kunci pribadi Bob dan kunci publik Alice akan mereproduksi kunci komitmen yang sama dan kunci pembungkus bersama dan dapat mendekripsi catatan. Jika kunci publik Bob berasal dari AWS KMS key pasangan, maka Bob dapat membuat keyring AWS KMS ECDH untuk mendekripsi catatan.

Raw ECDH keyring mengenkripsi catatan dengan kunci simetris menggunakan AES-GCM. Kunci data kemudian dienkripsi dengan kunci pembungkus bersama turunan menggunakan AES-GCM. Setiap keyring ECDH Raw hanya dapat memiliki satu kunci pembungkus bersama, tetapi Anda

dapat menyertakan beberapa gantungan kunci ECDH mentah, sendiri atau dengan gantungan kunci lainnya, dalam multi-keyring.

Anda bertanggung jawab untuk membuat, menyimpan, dan melindungi kunci pribadi Anda, sebaiknya dalam modul keamanan perangkat keras (HSM) atau sistem manajemen kunci. Pasangan kunci pengirim dan penerima banyak berada pada kurva elips yang sama. AWS Database Encryption SDK mendukung spesifikasi elips cuve berikut:

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

Membuat keyring ECDH mentah

Raw ECDH keyring mendukung tiga skema perjanjian utama:`RawPrivateKeyToStaticPublicKey`,, dan. `EphemeralPrivateKeyToStaticPublicKey` `PublicKeyDiscovery` Skema perjanjian utama yang Anda pilih menentukan operasi kriptografi mana yang dapat Anda lakukan dan bagaimana bahan kunci dirakit.

Topik

- RawPrivateKeyToStaticPublicKey
- EphemeralPrivateKeyToStaticPublicKey
- PublicKeyDiscovery

RawPrivateKeyToStaticPublicKey

Gunakan skema perjanjian `RawPrivateKeyToStaticPublicKey` kunci untuk mengonfigurasi kunci pribadi pengirim dan kunci publik penerima secara statis di keyring. Skema perjanjian kunci ini dapat mengenkripsi dan mendekripsi catatan.

Untuk menginisialisasi keyring ECDH mentah dengan skema perjanjian `RawPrivateKeyToStaticPublicKey` kunci, berikan nilai-nilai berikut:

- Kunci pribadi pengirim

Anda harus memberikan kunci pribadi yang dikodekan PEM pengirim (PrivateKeyInfo struktur PKCS #8), seperti yang didefinisikan dalam RFC 5958.

- Kunci publik penerima

  Anda harus memberikan kunci publik X.509 yang dikodekan DER penerima, juga dikenal sebagai `SubjectPublicKeyInfo` (SPKI), sebagaimana didefinisikan dalam RFC 5280.

  Anda dapat menentukan kunci publik dari perjanjian kunci asimetris KMS key pair atau kunci publik dari key pair yang dihasilkan di luar. AWS

- Spesifikasi kurva

  Mengidentifikasi spesifikasi kurva elips dalam pasangan kunci yang ditentukan. Pasangan kunci pengirim dan penerima harus memiliki spesifikasi kurva yang sama.

  Nilai valid: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

C# / .NET

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

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

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

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

Java

Contoh Java berikut menggunakan skema perjanjian `RawPrivateKeyToStaticPublicKey` kunci untuk secara statis mengkonfigurasi kunci pribadi pengirim dan kunci publik penerima. Kedua pasangan kunci berada di `ECC_NIST_P256` kurva.

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

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

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

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

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

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

Rust

Contoh Python berikut menggunakan skema perjanjian `raw_ecdh_static_configuration` kunci untuk secara statis mengkonfigurasi kunci pribadi pengirim dan kunci publik penerima. Kedua pasangan kunci harus berada pada kurva yang sama.

```
// Create keyring input
let raw_ecdh_static_configuration_input =
    RawPrivateKeyToStaticPublicKeyInput::builder()
        // Must be a UTF8 PEM-encoded private key
        .sender_static_private_key(private_key_sender_utf8_bytes)
        // Must be a UTF8 DER-encoded X.509 public key
        .recipient_public_key(public_key_recipient_utf8_bytes)
        .build()?;

let raw_ecdh_static_configuration =
 RawEcdhStaticConfigurations::RawPrivateKeyToStaticPublicKey(raw_ecdh_static_configuration_i

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

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

EphemeralPrivateKeyToStaticPublicKey

Keyrings yang dikonfigurasi dengan skema perjanjian `EphemeralPrivateKeyToStaticPublicKey` kunci membuat key pair baru secara lokal dan mendapatkan kunci pembungkus bersama yang unik untuk setiap panggilan enkripsi.

Skema perjanjian kunci ini hanya dapat mengenkripsi catatan. Untuk mendekripsi catatan yang dienkripsi dengan skema perjanjian `EphemeralPrivateKeyToStaticPublicKey` kunci, Anda harus menggunakan skema perjanjian kunci penemuan yang dikonfigurasi dengan kunci publik penerima yang sama. Untuk mendekripsi, Anda dapat menggunakan keyring ECDH mentah dengan algoritma perjanjian `PublicKeyDiscovery`kunci, atau, jika kunci publik penerima berasal dari key

pair KMS perjanjian kunci asimetris, Anda dapat menggunakan keyring AWS KMS ECDH dengan skema perjanjian kunci. KmsPublicKeyDiscovery

Untuk menginisialisasi keyring ECDH mentah dengan skema perjanjian `EphemeralPrivateKeyToStaticPublicKey` kunci, berikan nilai-nilai berikut:

- Kunci publik penerima

  Anda harus memberikan kunci publik X.509 yang dikodekan DER penerima, juga dikenal sebagai `SubjectPublicKeyInfo` (SPKI), sebagaimana didefinisikan dalam RFC 5280.

  Anda dapat menentukan kunci publik dari perjanjian kunci asimetris KMS key pair atau kunci publik dari key pair yang dihasilkan di luar. AWS
- Spesifikasi kurva

  Mengidentifikasi spesifikasi kurva elips dalam kunci publik yang ditentukan.

  Pada enkripsi, keyring membuat key pair baru pada kurva yang ditentukan dan menggunakan kunci pribadi baru dan kunci publik tertentu untuk mendapatkan kunci pembungkus bersama.

  Nilai valid: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

C# / .NET

Contoh berikut membuat keyring ECDH mentah dengan skema perjanjian `EphemeralPrivateKeyToStaticPublicKey` kunci. Pada enkripsi, keyring akan membuat key pair baru secara lokal pada kurva yang ditentukan. `ECC_NIST_P256`

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

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

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

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

Java

Contoh berikut membuat keyring ECDH mentah dengan skema perjanjian `EphemeralPrivateKeyToStaticPublicKey` kunci. Pada enkripsi, keyring akan membuat key pair baru secara lokal pada kurva yang ditentukan. `ECC_NIST_P256`

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

    ByteBuffer recipientPublicKey = getPublicKeyBytes();

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

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

Rust

Contoh berikut membuat keyring ECDH mentah dengan skema perjanjian `ephemeral_raw_ecdh_static_configuration` kunci. Pada enkripsi, keyring akan membuat key pair baru secara lokal pada kurva yang ditentukan.

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

let ephemeral_raw_ecdh_static_configuration =

 RawEcdhStaticConfigurations::EphemeralPrivateKeyToStaticPublicKey(ephemeral_raw_ecdh_static

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

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

PublicKeyDiscovery

Saat mendekripsi, ini adalah praktik terbaik untuk menentukan kunci pembungkus yang dapat digunakan SDK Enkripsi AWS Database. Untuk mengikuti praktik terbaik ini, gunakan keyring ECDH yang menentukan kunci pribadi pengirim dan kunci publik penerima. Namun, Anda juga dapat membuat keyring penemuan ECDH mentah, yaitu gantungan kunci ECDH mentah yang dapat mendekripsi catatan apa pun di mana kunci publik kunci yang ditentukan cocok dengan kunci publik penerima yang disimpan di bidang deskripsi materi dari catatan terenkripsi. Skema perjanjian kunci ini hanya dapat mendekripsi catatan.

> **Important**
>
> Ketika Anda mendekripsi catatan menggunakan skema perjanjian `PublicKeyDiscovery` kunci, Anda menerima semua kunci publik, terlepas dari siapa yang memilikinya.

Untuk menginisialisasi keyring ECDH mentah dengan skema perjanjian `PublicKeyDiscovery` kunci, berikan nilai-nilai berikut:

- Kunci pribadi statis penerima

  Anda harus memberikan kunci pribadi yang disandikan PEM penerima (PrivateKeyInfo struktur PKCS #8), seperti yang didefinisikan dalam RFC 5958.

- Spesifikasi kurva

  Mengidentifikasi spesifikasi kurva elips dalam kunci pribadi yang ditentukan. Pasangan kunci pengirim dan penerima harus memiliki spesifikasi kurva yang sama.

  Nilai valid: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

C# / .NET

Contoh berikut membuat keyring ECDH mentah dengan skema perjanjian `PublicKeyDiscovery` kunci. Gantungan kunci ini dapat mendekripsi catatan apa pun di mana kunci publik dari kunci pribadi yang ditentukan cocok dengan kunci publik penerima yang disimpan di bidang deskripsi materi dari catatan terenkripsi.

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

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

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

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

Java

Contoh berikut membuat keyring ECDH mentah dengan skema perjanjian `PublicKeyDiscovery` kunci. Gantungan kunci ini dapat mendekripsi catatan apa pun di mana kunci publik dari kunci pribadi yang ditentukan cocok dengan kunci publik penerima yang disimpan di bidang deskripsi materi dari catatan terenkripsi.

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

    KeyPair recipient = GetRawEccKey();

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

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

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

```
}
```

Rust

Contoh berikut membuat keyring ECDH mentah dengan skema perjanjian `discovery_raw_ecdh_static_configuration` kunci. Keyring ini dapat mendekripsi pesan apa pun di mana kunci publik dari kunci pribadi yang ditentukan cocok dengan kunci publik penerima yang disimpan pada ciphertext pesan.

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

let discovery_raw_ecdh_static_configuration =

 RawEcdhStaticConfigurations::PublicKeyDiscovery(discovery_raw_ecdh_static_configuration_inp

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

Multi-gantungan kunci

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Anda dapat menggabungkan keyrings menjadi multi-keyring. Multi-keyring adalah keyring yang terdiri dari satu atau lebih gantungan kunci individu dari jenis yang sama atau berbeda. Efeknya seperti menggunakan beberapa gantungan kunci dalam satu seri. Bila Anda menggunakan multi-keyring untuk mengenkripsi data, salah satu kunci pembungkus di salah satu keyrings nya dapat mendekripsi data tersebut.

Saat Anda membuat multi-keyring untuk mengenkripsi data, Anda menunjuk salah satu keyring sebagai keyring generator. Semua gantungan kunci lainnya dikenal sebagai gantungan kunci anak. Generator keyring menghasilkan dan mengenkripsi kunci data plaintext. Kemudian, semua kunci pembungkus di semua keyring anak mengenkripsi kunci data teks biasa yang sama. Multi-keyring mengembalikan kunci plaintext dan satu kunci data terenkripsi untuk setiap kunci pembungkus di multi-keyring. Jika keyring generator adalah keyring [KMS, kunci generator di AWS KMS keyring] menghasilkan dan mengenkripsi kunci plaintext. Kemudian, semua tambahan AWS KMS keys di AWS KMS keyring, dan semua kunci pembungkus di semua keyring anak di multi-keyring, mengenkripsi kunci plaintext yang sama.

Saat mendekripsi, AWS Database Encryption SDK menggunakan keyrings untuk mencoba mendekripsi salah satu kunci data terenkripsi. Gantungan kunci dipanggil dalam urutan yang ditentukan dalam multi-keyring. Pemrosesan berhenti segera setelah kunci apa pun di keyring apa pun dapat mendekripsi kunci data terenkripsi.

Untuk membuat multi-keyring, pertama-tama buat instance keyrings anak. Dalam contoh ini, kami menggunakan AWS KMS keyring dan keyring Raw AES, tetapi Anda dapat menggabungkan keyrings yang didukung dalam multi-keyring.

Java

```
// 1. Create the raw AES keyring.
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
final CreateRawAesKeyringInput createRawAesKeyringInput =
 CreateRawAesKeyringInput.builder()
        .keyName("AES_256_012")
        .keyNamespace("HSM_01")
        .wrappingKey(AESWrappingKey)
        .wrappingAlg(AesWrappingAlg.ALG_AES256_GCM_IV12_TAG16)
        .build();
IKeyring rawAesKeyring = matProv.CreateRawAesKeyring(createRawAesKeyringInput);

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

C# / .NET

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

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

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

Rust

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

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

// 2. Create the AWS KMS keyring
```

```
let aws_kms_mrk_multi_keyring = mpl
    .create_aws_kms_mrk_multi_keyring()
    .generator(key_arn)
    .send()
    .await?;
```

Selanjutnya, buat multi-keyring dan tentukan keyring generatornya, jika ada. Dalam contoh ini, kami membuat multi-keyring di mana keyring adalah AWS KMS keyring generator dan keyring AES adalah keyring anak.

Java

`CreateMultiKeyringInput`Konstruktor Java memungkinkan Anda menentukan keyring generator dan keyrings anak. `createMultiKeyringInput`Objek yang dihasilkan tidak dapat diubah.

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

C# / .NET

`CreateMultiKeyringInput`Konstruktor.NET memungkinkan Anda menentukan keyring generator dan keyrings anak. `CreateMultiKeyringInput`Objek yang dihasilkan tidak dapat diubah.

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

Rust

```
let multi_keyring = mpl
    .create_multi_keyring()
```

```
    .generator(aws_kms_mrk_multi_keyring)
    .child_keyrings(vec![raw_aes_keyring.clone()])
    .send()
    .await?;
```

Sekarang, Anda dapat menggunakan multi-keyring untuk mengenkripsi dan mendekripsi data.

Enkripsi yang dapat dicari

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Enkripsi yang dapat dicari memungkinkan Anda untuk mencari catatan terenkripsi tanpa mendekripsi seluruh database. Ini dilakukan dengan menggunakan beacon, yang membuat peta antara nilai plaintext yang ditulis ke bidang dan nilai terenkripsi yang sebenarnya disimpan dalam database Anda. AWS Database Encryption SDK menyimpan beacon di bidang baru yang ditambahkan ke catatan. Bergantung pada jenis suar yang Anda gunakan, Anda dapat melakukan penelusuran pencocokan persis atau kueri kompleks yang lebih disesuaikan pada data terenkripsi Anda.

> **Note**
>
> Enkripsi yang dapat dicari dalam SDK Enkripsi AWS Database berbeda dari enkripsi simetris yang dapat dicari yang didefinisikan dalam penelitian akademis, seperti enkripsi simetris yang dapat dicari.

Beacon adalah tag Hash Based Message Authentication Code (HMAC) terpotong yang membuat peta antara plaintext dan nilai terenkripsi bidang. Saat Anda menulis nilai baru ke bidang terenkripsi yang dikonfigurasi untuk enkripsi yang dapat dicari, SDK Enkripsi AWS Database menghitung HMAC di atas nilai teks biasa. Output HMAC ini adalah kecocokan satu-ke-satu (1:1) untuk nilai plaintext dari bidang itu. Output HMAC terpotong sehingga beberapa nilai plaintext yang berbeda dipetakan ke tag HMAC terpotong yang sama. Positif palsu ini membatasi kemampuan pengguna yang tidak sah untuk mengidentifikasi informasi yang membedakan tentang nilai plaintext. Saat Anda menanyakan suar, SDK Enkripsi AWS Database secara otomatis menyaring positif palsu ini dan mengembalikan hasil teks biasa dari kueri Anda.

Jumlah rata-rata positif palsu yang dihasilkan untuk setiap suar ditentukan oleh panjang suar yang tersisa setelah pemotongan. Untuk bantuan menentukan panjang suar yang sesuai untuk implementasi Anda, lihat Menentukan panjang suar.

> **Note**
>
> Enkripsi yang dapat dicari dirancang untuk diimplementasikan dalam database baru yang tidak terisi. Setiap suar yang dikonfigurasi dalam database yang ada hanya akan memetakan catatan baru yang diunggah ke database, tidak ada cara bagi suar untuk memetakan data yang ada.

Topik

- Apakah beacon tepat untuk dataset saya?
- Skenario enkripsi yang dapat dicari

Apakah beacon tepat untuk dataset saya?

Menggunakan beacon untuk melakukan kueri pada data terenkripsi mengurangi biaya kinerja yang terkait dengan database terenkripsi sisi klien. Saat Anda menggunakan beacon, ada tradeoff yang melekat antara seberapa efisien kueri Anda dan seberapa banyak informasi yang terungkap tentang distribusi data Anda. Beacon tidak mengubah status lapangan yang dienkripsi. Saat Anda mengenkripsi dan menandatangani bidang dengan AWS Database Encryption SDK, nilai plaintext bidang tersebut tidak pernah terpapar ke database. Basis data menyimpan nilai bidang yang dienkripsi secara acak.

Beacon disimpan di samping bidang terenkripsi tempat mereka dihitung. Ini berarti bahwa meskipun pengguna yang tidak sah tidak dapat melihat nilai teks biasa dari bidang terenkripsi, mereka mungkin dapat melakukan analisis statistik pada beacon untuk mempelajari lebih lanjut tentang distribusi kumpulan data Anda, dan, dalam kasus ekstrim, mengidentifikasi nilai teks biasa yang dipetakan oleh beacon. Cara Anda mengkonfigurasi beacon Anda dapat mengurangi risiko ini. Secara khusus, memilih panjang suar yang tepat dapat membantu Anda menjaga kerahasiaan kumpulan data Anda.

Keamanan vs Kinerja

- Semakin pendek panjang suar, semakin banyak keamanan yang dipertahankan.
- Semakin panjang panjang suar, semakin banyak kinerja yang dipertahankan.

Enkripsi yang dapat dicari mungkin tidak dapat memberikan tingkat kinerja dan keamanan yang diinginkan untuk semua kumpulan data. Tinjau model ancaman, persyaratan keamanan, dan kebutuhan kinerja Anda sebelum mengonfigurasi suar apa pun.

Pertimbangkan persyaratan keunikan kumpulan data berikut saat Anda menentukan apakah enkripsi yang dapat dicari tepat untuk kumpulan data Anda.

Distribusi

Jumlah keamanan yang dipertahankan oleh suar tergantung pada distribusi kumpulan data Anda. Saat Anda mengonfigurasi bidang terenkripsi untuk enkripsi yang dapat dicari, SDK Enkripsi AWS Database menghitung HMAC atas nilai teks biasa yang ditulis ke bidang tersebut. Semua beacon yang dihitung untuk bidang tertentu dihitung menggunakan kunci yang sama, dengan pengecualian database multitenant yang menggunakan kunci berbeda untuk setiap penyewa. Ini berarti bahwa jika nilai plaintext yang sama ditulis ke bidang beberapa kali, tag HMAC yang sama dibuat untuk setiap instance dari nilai plaintext tersebut.

Anda harus menghindari membangun beacon dari bidang yang berisi nilai yang sangat umum. Misalnya, pertimbangkan database yang menyimpan alamat setiap penduduk negara bagian Illinois. Jika Anda membangun suar dari `City` bidang terenkripsi, suar yang dihitung di atas “Chicago” akan terwakili secara berlebihan karena persentase besar populasi Illinois yang tinggal di Chicago. Bahkan jika pengguna yang tidak sah hanya dapat membaca nilai terenkripsi dan nilai suar, mereka mungkin dapat mengidentifikasi catatan mana yang berisi data untuk penduduk Chicago jika suar mempertahankan distribusi ini. Untuk meminimalkan jumlah informasi pembeda yang terungkap tentang distribusi Anda, Anda harus memotong suar Anda dengan cukup. Panjang suar yang diperlukan untuk menyembunyikan distribusi yang tidak merata ini memiliki biaya kinerja yang signifikan yang mungkin tidak memenuhi kebutuhan aplikasi Anda.

Anda harus hati-hati menganalisis distribusi dataset Anda untuk menentukan berapa banyak beacon Anda perlu dipotong. Panjang suar yang tersisa setelah pemotongan secara langsung berkorelasi dengan jumlah informasi statistik yang dapat diidentifikasi tentang distribusi Anda. Anda mungkin perlu memilih panjang suar yang lebih pendek untuk meminimalkan jumlah informasi pembeda yang terungkap tentang kumpulan data Anda.

Dalam kasus ekstrim, Anda tidak dapat menghitung panjang suar untuk kumpulan data terdistribusi tidak merata yang secara efektif menyeimbangkan kinerja dan keamanan. Misalnya, Anda tidak boleh membuat suar dari bidang yang menyimpan hasil tes medis untuk penyakit langka. Karena `NEGATIVE` hasil diharapkan secara signifikan lebih umum dalam kumpulan data, `POSITIVE` hasil dapat dengan mudah diidentifikasi dengan seberapa jarang hasilnya.

Sangat menantang untuk menyembunyikan distribusi ketika bidang hanya memiliki dua nilai yang mungkin. Jika Anda menggunakan panjang suar yang cukup pendek untuk menyembunyikan distribusi, semua nilai plaintext dipetakan ke tag HMAC yang sama. Jika Anda menggunakan panjang suar yang lebih panjang, jelas beacon mana yang memetakan ke nilai teks biasa. `POSITIVE`

Korelasi

Kami sangat menyarankan agar Anda menghindari pembuatan beacon yang berbeda dari bidang dengan nilai yang berkorelasi. Beacon yang dibangun dari bidang berkorelasi memerlukan panjang suar yang lebih pendek untuk meminimalkan jumlah informasi yang terungkap tentang distribusi setiap kumpulan data ke pengguna yang tidak sah. Anda harus menganalisis kumpulan data Anda dengan cermat, termasuk entropi dan distribusi gabungan dari nilai yang berkorelasi, untuk menentukan berapa banyak beacon Anda perlu dipotong. Jika panjang suar yang dihasilkan tidak memenuhi kebutuhan kinerja Anda, maka beacon mungkin tidak cocok untuk kumpulan data Anda.

Misalnya, Anda tidak boleh membuat dua beacon terpisah dari `City` dan `ZIPCode` bidang karena kode ZIP kemungkinan akan dikaitkan dengan hanya satu kota. Biasanya, positif palsu yang dihasilkan oleh suar membatasi kemampuan pengguna yang tidak sah untuk mengidentifikasi informasi yang membedakan tentang kumpulan data Anda. Tetapi korelasi antara `ZIPCode` bidang `City` dan berarti bahwa pengguna yang tidak sah dapat dengan mudah mengidentifikasi hasil mana yang positif palsu dan membedakan kode ZIP yang berbeda.

Anda juga harus menghindari pembuatan beacon dari bidang yang berisi nilai plaintext yang sama. Misalnya, Anda tidak boleh membuat suar dari `mobilePhone` dan `preferredPhone` bidang karena kemungkinan memiliki nilai yang sama. Jika Anda membuat beacon yang berbeda dari kedua bidang, AWS Database Encryption SDK akan membuat beacon untuk setiap bidang di bawah kunci yang berbeda. Ini menghasilkan dua tag HMAC yang berbeda untuk nilai plaintext yang sama. Dua beacon yang berbeda tidak mungkin memiliki positif palsu yang sama dan pengguna yang tidak sah mungkin dapat membedakan nomor telepon yang berbeda.

Bahkan jika kumpulan data Anda berisi bidang yang berkorelasi atau memiliki distribusi yang tidak merata, Anda mungkin dapat membangun beacon yang menjaga kerahasiaan kumpulan data Anda dengan menggunakan panjang suar yang lebih pendek. Namun, panjang suar tidak menjamin bahwa setiap nilai unik dalam kumpulan data Anda akan menghasilkan sejumlah positif palsu yang secara efektif meminimalkan jumlah informasi pembeda yang terungkap tentang kumpulan data Anda. Panjang suar hanya memperkirakan jumlah rata-rata positif palsu yang dihasilkan. Semakin

terdistribusi kumpulan data Anda secara tidak merata, panjang suar yang kurang efektif dalam menentukan jumlah rata-rata positif palsu yang dihasilkan.

Pertimbangkan dengan cermat distribusi bidang tempat Anda membangun suar dan pertimbangkan berapa banyak yang Anda perlukan untuk memotong panjang suar untuk memenuhi persyaratan keamanan Anda. Topik-topik berikut dalam Bab ini mengasumsikan bahwa beacon Anda didistribusikan secara seragam dan tidak mengandung data yang berkorelasi.

Skenario enkripsi yang dapat dicari

Contoh berikut menunjukkan solusi enkripsi yang dapat dicari sederhana. Dalam aplikasi, bidang contoh yang digunakan dalam contoh ini mungkin tidak memenuhi rekomendasi keunikan distribusi dan korelasi untuk beacon. Anda dapat menggunakan contoh ini untuk referensi saat Anda membaca tentang konsep enkripsi yang dapat dicari di Bab ini.

Pertimbangkan database bernama `Employees` yang melacak data karyawan untuk perusahaan. Setiap record dalam database berisi bidang yang disebut EmployeeId LastName, FirstName, dan Address. Setiap bidang dalam `Employees` database diidentifikasi oleh kunci utama`EmployeeID`.

Berikut ini adalah contoh catatan plaintext dalam database.

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

Jika Anda menandai `LastName` dan `FirstName` bidang seperti `ENCRYPT_AND_SIGN` dalam tindakan kriptografi Anda, nilai dalam bidang ini dienkripsi secara lokal sebelum diunggah ke database. Data terenkripsi yang diunggah sepenuhnya acak, database tidak mengenali data ini sebagai dilindungi. Itu hanya mendeteksi entri data yang khas. Ini berarti bahwa catatan yang sebenarnya disimpan dalam database mungkin terlihat seperti berikut.

```
{
```

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

Jika Anda perlu menanyakan database untuk kecocokan persis di `LastName` bidang, konfigurasikan suar standar bernama LastNameuntuk memetakan nilai teks biasa yang ditulis ke `LastName` bidang ke nilai terenkripsi yang disimpan dalam database.

Beacon ini menghitung HMACs dari nilai plaintext di lapangan. `LastName` Setiap output HMAC terpotong sehingga tidak lagi cocok dengan nilai plaintext. Misalnya, hash lengkap dan hash terpotong untuk `Jones` mungkin terlihat seperti berikut ini.

Hash lengkap

`2aa4e9b404c68182562b6ec761fcca5306de527826a69468885e59dc36d0c3f824bdd44cab45526f`

Hash terpotong

`b35099d408c833`

Setelah suar standar dikonfigurasi, Anda dapat melakukan pencarian kesetaraan di lapangan. `LastName` Misalnya, jika Anda ingin mencari`Jones`, gunakan LastNamesuar untuk melakukan kueri berikut.

```
LastName = Jones
```

SDK Enkripsi AWS Database secara otomatis menyaring positif palsu dan mengembalikan hasil teks biasa dari kueri Anda.

Beacon

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Beacon adalah tag Hash Based Message Authentication Code (HMAC) terpotong yang membuat peta antara nilai plaintext yang ditulis ke bidang dan nilai terenkripsi yang sebenarnya disimpan dalam database Anda. Beacon tidak mengubah status lapangan yang dienkripsi. Beacon menghitung HMAC atas nilai plaintext bidang dan menyimpannya di samping nilai terenkripsi. Output HMAC ini adalah kecocokan satu-ke-satu (1:1) untuk nilai plaintext dari bidang itu. Output HMAC terpotong sehingga beberapa nilai plaintext yang berbeda dipetakan ke tag HMAC terpotong yang sama. Positif palsu ini membatasi kemampuan pengguna yang tidak sah untuk mengidentifikasi informasi yang membedakan tentang nilai plaintext.

Beacon hanya dapat dibangun dari bidang yang ditandai`ENCRYPT_AND_SIGN`,`SIGN_ONLY`, atau `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` dalam tindakan kriptografi Anda. Suar itu sendiri tidak ditandatangani atau dienkripsi. Anda tidak dapat membangun suar dengan bidang yang ditandai. `DO_NOTHING`

Jenis suar yang Anda konfigurasikan menentukan jenis kueri yang dapat Anda lakukan. Ada dua jenis beacon yang mendukung enkripsi yang dapat dicari. Suar standar melakukan pencarian kesetaraan. Compound beacon menggabungkan string plaintext literal dan beacon standar untuk melakukan operasi database yang kompleks. Setelah Anda mengkonfigurasi beacon Anda, Anda harus mengkonfigurasi indeks sekunder untuk setiap suar sebelum Anda dapat mencari di bidang terenkripsi. Untuk informasi selengkapnya, lihat Mengkonfigurasi indeks sekunder dengan beacon.

Topik

- Suar standar
- Suar majemuk

Suar standar

Beacon standar adalah cara paling sederhana untuk menerapkan enkripsi yang dapat dicari di database Anda. Mereka hanya dapat melakukan pencarian kesetaraan untuk satu bidang terenkripsi atau virtual. Untuk mempelajari cara mengonfigurasi suar standar, lihat Mengonfigurasi suar standar.

Bidang tempat suar standar dibangun dari disebut sumber suar. Ini mengidentifikasi lokasi data yang perlu dipetakan oleh suar. Sumber suar dapat berupa bidang terenkripsi atau bidang virtual. Sumber suar di setiap suar standar harus unik. Anda tidak dapat mengonfigurasi dua beacon dengan sumber suar yang sama.

Beacon standar dapat digunakan untuk melakukan pencarian kesetaraan untuk bidang terenkripsi atau virtual. Atau, mereka dapat digunakan untuk membangun suar majemuk untuk melakukan operasi database yang lebih kompleks. Untuk membantu Anda mengatur dan mengelola beacon standar, AWS Database Encryption SDK menyediakan gaya beacon opsional berikut yang menentukan tujuan penggunaan suar standar. Untuk informasi selengkapnya lihat, Mendefinisikan gaya suar.

Anda dapat membuat suar standar yang melakukan pencarian kesetaraan untuk satu bidang terenkripsi, atau Anda dapat membuat suar standar yang melakukan pencarian kesetaraan pada rangkaian beberapa,, dan bidang dengan membuat bidang virtual. `ENCRYPT_AND_SIGN` `SIGN_ONLY` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`

Bidang virtual

Bidang virtual adalah bidang konseptual yang dibangun dari satu atau lebih bidang sumber. Membuat bidang virtual tidak menulis bidang baru ke catatan Anda. Bidang virtual tidak secara eksplisit disimpan dalam database Anda. Ini digunakan dalam konfigurasi suar standar untuk memberikan instruksi suar tentang cara mengidentifikasi segmen tertentu dari bidang atau menggabungkan beberapa bidang dalam catatan untuk melakukan kueri tertentu. Bidang virtual membutuhkan setidaknya satu bidang terenkripsi.

> **Note**
>
> Contoh berikut menunjukkan jenis transformasi dan kueri yang dapat Anda lakukan dengan bidang virtual. Dalam aplikasi, bidang contoh yang digunakan dalam contoh ini mungkin tidak memenuhi rekomendasi keunikan distribusi dan korelasi untuk beacon.

Misalnya, jika Anda ingin melakukan pencarian kesetaraan pada rangkaian `FirstName` dan `LastName` bidang, Anda dapat membuat salah satu bidang virtual berikut.

- `NameTag`Bidang virtual, dibangun dari huruf pertama `FirstName` lapangan, diikuti oleh `LastName` bidang, semuanya dalam huruf kecil. Bidang virtual ini memungkinkan Anda untuk melakukan query`NameTag=mjones`.
- `LastFirst`Bidang virtual, yang dibangun dari `LastName` lapangan, diikuti oleh `FirstName` lapangan. Bidang virtual ini memungkinkan Anda untuk melakukan query`LastFirst=JonesMary`.

Atau, jika Anda ingin melakukan pencarian kesetaraan pada segmen tertentu dari bidang terenkripsi, buat bidang virtual yang mengidentifikasi segmen yang ingin Anda kueri.

Misalnya, jika Anda ingin menanyakan `IPAddress` bidang terenkripsi menggunakan tiga segmen pertama dari alamat IP, buat bidang virtual berikut.

- `IPSegment`Bidang virtual, dibangun dari`Segments('.', 0, 3)`. Bidang virtual ini memungkinkan Anda untuk melakukan query`IPSegment=192.0.2`. Kueri mengembalikan semua catatan dengan `IPAddress` nilai yang dimulai dengan "192.0.2".

Bidang virtual harus unik. Dua bidang virtual tidak dapat dibangun dari bidang sumber yang sama persis.

Untuk bantuan mengonfigurasi bidang virtual dan beacon yang menggunakannya, lihat Membuat bidang virtual.

Suar majemuk

Compound beacon membuat indeks yang meningkatkan kinerja kueri dan memungkinkan Anda melakukan operasi database yang lebih kompleks. Anda dapat menggunakan suar majemuk untuk menggabungkan string teks biasa literal dan suar standar untuk melakukan kueri kompleks pada catatan terenkripsi, seperti menanyakan dua jenis rekaman yang berbeda dari satu indeks atau menanyakan kombinasi bidang dengan kunci pengurutan. Untuk contoh solusi suar majemuk lainnya, lihat Memilih jenis suar.

Suar majemuk dapat dibangun dari suar standar atau kombinasi beacon standar dan bidang yang ditandatangani. Mereka dibangun dari daftar bagian. Semua suar majemuk harus menyertakan daftar bagian terenkripsi yang mengidentifikasi `ENCRYPT_AND_SIGN` bidang yang termasuk dalam suar. Setiap `ENCRYPT_AND_SIGN` bidang harus diidentifikasi dengan suar standar. Suar majemuk yang lebih kompleks mungkin juga mencakup daftar bagian yang ditandatangani yang mengidentifikasi teks biasa `SIGN_ONLY` atau `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` bidang yang termasuk dalam suar, dan daftar bagian konstruktor yang mengidentifikasi semua kemungkinan cara suar majemuk dapat merakit bidang.

> **Note**
>
> AWS Database Encryption SDK juga mendukung beacon bertanda tangan yang dapat dikonfigurasi seluruhnya dari `SIGN_ONLY` plaintext dan field. `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Signed beacon adalah jenis

suar majemuk yang mengindeks dan melakukan kueri kompleks pada bidang yang ditandatangani, tetapi tidak dienkripsi. Untuk informasi selengkapnya, lihat Membuat suar yang ditandatangani.

Untuk bantuan mengonfigurasi suar majemuk, lihat Mengonfigurasi suar majemuk.

Cara Anda mengonfigurasi suar majemuk menentukan jenis kueri yang dapat dilakukannya. Misalnya, Anda dapat membuat beberapa bagian terenkripsi dan ditandatangani opsional untuk memungkinkan lebih banyak fleksibilitas dalam kueri Anda. Untuk informasi lebih lanjut tentang jenis kueri yang dapat dilakukan oleh suar majemuk, lihat. Meminta suar

Merencanakan suar

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Beacon dirancang untuk diimplementasikan dalam database baru yang tidak berpenghuni. Setiap suar yang dikonfigurasi dalam database yang ada hanya akan memetakan catatan baru yang ditulis ke database. Beacon dihitung dari nilai plaintext bidang, setelah bidang dienkripsi, tidak ada cara bagi beacon untuk memetakan data yang ada. Setelah Anda menulis catatan baru dengan suar, Anda tidak dapat memperbarui konfigurasi suar. Namun, Anda dapat menambahkan beacon baru untuk bidang baru yang Anda tambahkan ke catatan Anda.

Untuk menerapkan enkripsi yang dapat dicari, Anda harus menggunakan keyring AWS KMS Hierarkis untuk menghasilkan, mengenkripsi, dan mendekripsi kunci data yang digunakan untuk melindungi catatan Anda. Untuk informasi selengkapnya, lihat Menggunakan keyring Hierarkis untuk enkripsi yang dapat dicari.

Sebelum Anda dapat mengonfigurasi beacon untuk enkripsi yang dapat dicari, Anda perlu meninjau persyaratan enkripsi, pola akses database, dan model ancaman untuk menentukan solusi terbaik untuk database Anda.

Jenis suar yang Anda konfigurasikan menentukan jenis kueri yang dapat Anda lakukan. Panjang suar yang Anda tentukan dalam konfigurasi suar standar menentukan jumlah positif palsu yang diharapkan yang dihasilkan untuk suar tertentu. Kami sangat menyarankan untuk mengidentifikasi

dan merencanakan jenis kueri yang perlu Anda lakukan sebelum mengonfigurasi beacon Anda. Setelah Anda menggunakan suar, konfigurasi tidak dapat diperbarui.

Kami sangat menyarankan Anda meninjau dan menyelesaikan tugas-tugas berikut sebelum Anda mengonfigurasi beacon apa pun.

- Tentukan apakah beacon tepat untuk kumpulan data Anda
- Pilih jenis suar
- Pilih panjang suar
- Pilih nama suar

Ingat persyaratan keunikan beacon berikut saat Anda merencanakan solusi enkripsi yang dapat dicari untuk database Anda.

- Setiap suar standar harus memiliki sumber suar yang unik

  Beberapa beacon standar tidak dapat dibangun dari bidang terenkripsi atau virtual yang sama.

  Namun, suar standar tunggal dapat digunakan untuk membangun beberapa suar majemuk.
- Hindari membuat bidang virtual dengan bidang sumber yang tumpang tindih dengan beacon standar yang ada

  Membangun suar standar dari bidang virtual yang berisi bidang sumber yang digunakan untuk membuat suar standar lain dapat mengurangi keamanan kedua beacon.

  Untuk informasi selengkapnya, lihat Pertimbangan keamanan untuk bidang virtual.

Pertimbangan untuk database multitenant

Untuk menanyakan suar yang dikonfigurasi dalam database multitenant, Anda harus menyertakan bidang yang menyimpan `branch-key-id` terkait dengan penyewa yang mengenkripsi catatan dalam kueri Anda. Anda menentukan bidang ini ketika Anda menentukan sumber kunci suar. Agar kueri berhasil, nilai di bidang ini harus mengidentifikasi bahan kunci suar yang sesuai yang diperlukan untuk menghitung ulang suar.

Sebelum Anda mengkonfigurasi beacon Anda, Anda harus memutuskan bagaimana Anda berencana untuk memasukkan `branch-key-id` dalam kueri Anda. Untuk informasi selengkapnya tentang

berbagai cara yang dapat Anda sertakan `branch-key-id` dalam kueri, lihatMenanyakan beacon dalam database multitenant.

Memilih jenis suar

> Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Dengan enkripsi yang dapat dicari, Anda dapat mencari catatan terenkripsi dengan memetakan nilai teks biasa di bidang terenkripsi dengan suar. Jenis suar yang Anda konfigurasikan menentukan jenis kueri yang dapat Anda lakukan.

Kami sangat menyarankan untuk mengidentifikasi dan merencanakan jenis kueri yang perlu Anda lakukan sebelum mengonfigurasi beacon Anda. Setelah Anda mengkonfigurasi beacon Anda, Anda harus mengkonfigurasi indeks sekunder untuk setiap suar sebelum Anda dapat mencari di bidang terenkripsi. Untuk informasi selengkapnya, lihat Mengkonfigurasi indeks sekunder dengan beacon.

Beacon membuat peta antara nilai plaintext yang ditulis ke bidang dan nilai terenkripsi yang sebenarnya disimpan dalam database Anda. Anda tidak dapat membandingkan nilai dari dua beacon standar, bahkan jika mereka mengandung plaintext dasar yang sama. Dua beacon standar akan menghasilkan dua tag HMAC yang berbeda untuk nilai plaintext yang sama. Akibatnya, beacon standar tidak dapat melakukan kueri berikut.

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

Anda hanya dapat melakukan kueri di atas jika Anda membandingkan bagian yang ditandatangani dari suar majemuk, dengan pengecualian `CONTAINS` operator, yang dapat Anda gunakan dengan suar majemuk untuk mengidentifikasi seluruh nilai bidang terenkripsi atau ditandatangani yang berisi suar rakitan. Ketika Anda membandingkan bagian yang ditandatangani, Anda dapat secara opsional menyertakan awalan dari bagian terenkripsi, tetapi Anda tidak dapat menyertakan nilai terenkripsi bidang. Untuk informasi selengkapnya tentang jenis kueri yang dapat dilakukan oleh beacon standar dan gabungan, lihat Menanyakan suar.

Pertimbangkan solusi enkripsi yang dapat dicari berikut saat Anda meninjau pola akses database Anda. Contoh berikut menentukan suar mana yang akan dikonfigurasi untuk memenuhi persyaratan enkripsi dan kueri yang berbeda.

Beacon standar

Beacon standar hanya dapat melakukan pencarian kesetaraan. Anda dapat menggunakan beacon standar untuk melakukan kueri berikut.

Kueri satu bidang terenkripsi

Jika Anda ingin mengidentifikasi catatan yang berisi nilai tertentu untuk bidang terenkripsi, buat suar standar.

Contoh

Untuk contoh berikut, pertimbangkan database bernama `UnitInspection` yang melacak data inspeksi untuk fasilitas produksi. Setiap catatan dalam database berisi bidang yang disebut`work_id`,`inspection_date`,`inspector_id_last4`, dan`unit`. ID inspektur lengkap adalah angka antara 0—999.999.999. Namun, untuk memastikan bahwa kumpulan data didistribusikan secara merata, `inspector_id_last4` satu-satunya menyimpan empat digit terakhir dari ID inspektur. Setiap bidang dalam database diidentifikasi oleh kunci utama`work_id`. `unit`Bidang `inspector_id_last4` dan ditandai `ENCRYPT_AND_SIGN` dalam tindakan kriptografi.

Berikut ini adalah contoh entri plaintext dalam database. `UnitInspection`

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

Kueri satu bidang terenkripsi dalam catatan

> Jika `inspector_id_last4` bidang perlu dienkripsi, tetapi Anda masih memerlukan kueri untuk kecocokan yang tepat, buat suar standar dari bidang tersebut. `inspector_id_last4` Kemudian, gunakan suar standar untuk membuat indeks sekunder. Anda dapat menggunakan indeks sekunder ini untuk kueri pada bidang terenkripsi. `inspector_id_last4`

Untuk bantuan mengonfigurasi suar standar, lihat Mengonfigurasi suar standar.

Kueri bidang virtual

Bidang virtual adalah bidang konseptual yang dibangun dari satu atau lebih bidang sumber. Jika Anda ingin melakukan pencarian kesetaraan untuk segmen tertentu dari bidang terenkripsi, atau melakukan pencarian kesetaraan pada rangkaian beberapa bidang, buat suar standar dari bidang virtual. Semua bidang virtual harus menyertakan setidaknya satu bidang sumber terenkripsi.

Contoh

Contoh berikut membuat bidang virtual untuk `Employees` database. Berikut ini adalah contoh catatan plaintext dalam database. `Employees`

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

Kueri segmen bidang terenkripsi

Untuk contoh ini, `SSN` bidang dienkripsi.

Jika Anda ingin menanyakan `SSN` bidang menggunakan empat digit terakhir dari nomor jaminan sosial, buat bidang virtual yang mengidentifikasi segmen yang ingin Anda kueri.

`Last4SSN`Bidang virtual, dibangun dari `Suffix(4)` memungkinkan Anda untuk query`Last4SSN=0000`. Gunakan bidang virtual ini untuk membangun suar standar. Kemudian, gunakan suar standar untuk membuat indeks sekunder. Anda dapat menggunakan indeks sekunder ini untuk kueri pada bidang virtual. Kueri ini mengembalikan semua catatan dengan `SSN` nilai yang berakhir dengan empat digit terakhir yang Anda tentukan.

Kueri penggabungan beberapa bidang

> **Note**
>
> Contoh berikut menunjukkan jenis transformasi dan kueri yang dapat Anda lakukan dengan bidang virtual. Dalam aplikasi, bidang contoh yang digunakan dalam contoh ini mungkin tidak memenuhi rekomendasi keunikan distribusi dan korelasi untuk beacon.

Jika Anda ingin melakukan pencarian kesetaraan pada rangkaian `FirstName` dan bidang, Anda dapat membuat `LastName` bidang virtual, dibangun dari huruf pertama `NameTag` bidang, diikuti oleh `FirstName` bidang, semuanya dalam `LastName` huruf kecil. Gunakan bidang virtual ini untuk membangun suar standar. Kemudian, gunakan suar standar untuk membuat indeks sekunder. Anda dapat menggunakan indeks sekunder ini untuk kueri `NameTag=mjones` pada bidang virtual.

Setidaknya salah satu bidang sumber harus dienkripsi. Entah `FirstName` atau `LastName` bisa dienkripsi, atau keduanya bisa dienkripsi. Setiap bidang sumber teks biasa harus ditandai sebagai `SIGN_ONLY` atau `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` dalam tindakan kriptografi Anda.

Untuk bantuan mengonfigurasi bidang virtual dan beacon yang menggunakannya, lihat Membuat bidang virtual.

Suar majemuk

Compound beacon membuat indeks dari string plaintext literal dan beacon standar untuk melakukan operasi database yang kompleks. Anda dapat menggunakan suar majemuk untuk melakukan kueri berikut.

Kueri kombinasi bidang terenkripsi pada satu indeks

Jika Anda perlu menanyakan kombinasi bidang terenkripsi pada satu indeks, buat suar majemuk yang menggabungkan beacon standar individual yang dibangun untuk setiap bidang terenkripsi untuk membentuk indeks tunggal.

Setelah mengonfigurasi suar majemuk, Anda dapat membuat indeks sekunder yang menentukan suar majemuk sebagai kunci partisi untuk melakukan kueri pencocokan persis atau dengan kunci pengurutan untuk melakukan kueri yang lebih kompleks. Indeks sekunder yang menentukan suar

majemuk sebagai kunci pengurutan dapat melakukan kueri pencocokan persis dan kueri kompleks yang lebih disesuaikan.

Contoh

Untuk contoh berikut, pertimbangkan database bernama `UnitInspection` yang melacak data inspeksi untuk fasilitas produksi. Setiap catatan dalam database berisi bidang yang disebut`work_id`,`inspection_date`,`inspector_id_last4`, dan`unit`. ID inspektur lengkap adalah angka antara 0—999.999.999. Namun, untuk memastikan bahwa kumpulan data didistribusikan secara merata, `inspector_id_last4` satu-satunya menyimpan empat digit terakhir dari ID inspektur. Setiap bidang dalam database diidentifikasi oleh kunci utama`work_id`. `unit`Bidang `inspector_id_last4` dan ditandai `ENCRYPT_AND_SIGN` dalam tindakan kriptografi.

Berikut ini adalah contoh entri plaintext dalam database. `UnitInspection`

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

Lakukan pencarian kesetaraan pada kombinasi bidang terenkripsi

- Jika Anda ingin menanyakan `UnitInspection` database untuk pencocokan yang tepat`inspector_id_last4.unit`, pertama-tama buat beacon standar yang berbeda untuk bidang `inspector_id_last4` dan`unit`. Kemudian, buat suar majemuk dari dua suar standar.

- Setelah Anda mengkonfigurasi suar majemuk, buat indeks sekunder yang menentukan suar majemuk sebagai kunci partisi. Gunakan indeks sekunder ini untuk menanyakan kecocokan yang tepat pada`inspector_id_last4.unit`. Misalnya, Anda dapat menanyakan suar ini untuk menemukan daftar inspeksi yang dilakukan inspektur untuk unit tertentu.

Lakukan kueri kompleks pada kombinasi bidang terenkripsi

- Jika Anda ingin menanyakan `UnitInspection` database pada `inspector_id_last4` dan`inspector_id_last4.unit`, pertama buat beacon standar yang berbeda untuk `inspector_id_last4` dan `unit` bidang. Kemudian, buat suar majemuk dari dua suar standar.

- Setelah Anda mengkonfigurasi suar majemuk, buat indeks sekunder yang menentukan suar majemuk sebagai kunci pengurutan. Gunakan indeks sekunder ini untuk menanyakan

`UnitInspection` database untuk entri yang dimulai dengan inspektur tertentu atau kueri database untuk daftar semua unit dalam rentang ID unit tertentu yang diperiksa oleh inspektur tertentu. Anda juga dapat melakukan pencarian kecocokan tepat pada`inspector_id_last4.unit`.

Untuk bantuan mengonfigurasi suar majemuk, lihat Mengonfigurasi suar majemuk.

Kueri kombinasi bidang terenkripsi dan teks biasa pada satu indeks

Jika Anda perlu menanyakan kombinasi bidang terenkripsi dan teks biasa pada satu indeks, buat suar majemuk yang menggabungkan beacon standar individu dan bidang teks biasa untuk membentuk indeks tunggal. Bidang plaintext yang digunakan untuk membangun suar majemuk harus ditandai `SIGN_ONLY` atau `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` dalam tindakan kriptografi Anda.

Setelah mengonfigurasi suar majemuk, Anda dapat membuat indeks sekunder yang menentukan suar majemuk sebagai kunci partisi untuk melakukan kueri pencocokan persis atau dengan kunci pengurutan untuk melakukan kueri yang lebih kompleks. Indeks sekunder yang menentukan suar majemuk sebagai kunci pengurutan dapat melakukan kueri pencocokan persis dan kueri kompleks yang lebih disesuaikan.

Contoh

Untuk contoh berikut, pertimbangkan database bernama `UnitInspection` yang melacak data inspeksi untuk fasilitas produksi. Setiap catatan dalam database berisi bidang yang disebut`work_id`,`inspection_date`,`inspector_id_last4`, dan`unit`. ID inspektur lengkap adalah angka antara 0—999.999.999. Namun, untuk memastikan bahwa kumpulan data didistribusikan secara merata, `inspector_id_last4` satu-satunya menyimpan empat digit terakhir dari ID inspektur. Setiap bidang dalam database diidentifikasi oleh kunci utama`work_id`. `unit`Bidang `inspector_id_last4` dan ditandai `ENCRYPT_AND_SIGN` dalam tindakan kriptografi.

Berikut ini adalah contoh entri plaintext dalam database. `UnitInspection`

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

Lakukan pencarian kesetaraan pada kombinasi bidang

Jika Anda ingin menanyakan `UnitInspection` database untuk inspeksi yang dilakukan oleh inspektur tertentu pada tanggal tertentu, pertama-tama buat suar standar untuk bidang tersebut. `inspector_id_last4` `inspector_id_last4`Bidang ditandai `ENCRYPT_AND_SIGN` dalam tindakan kriptografi. Semua bagian terenkripsi memerlukan suar standar mereka sendiri. `inspection_date`Bidang ditandai `SIGN_ONLY` dan tidak memerlukan suar standar. Selanjutnya, buat suar majemuk dari `inspection_date` lapangan dan suar `inspector_id_last4` standar.

Setelah Anda mengkonfigurasi suar majemuk, buat indeks sekunder yang menentukan suar majemuk sebagai kunci partisi. Gunakan indeks sekunder ini untuk menanyakan database untuk catatan dengan kecocokan persis dengan inspektur dan tanggal inspeksi tertentu. Misalnya, Anda dapat menanyakan database untuk daftar semua inspeksi yang `8744` dilakukan oleh inspektur yang ID-nya berakhir pada tanggal tertentu.

Lakukan kueri kompleks pada kombinasi bidang

Jika Anda ingin menanyakan database untuk inspeksi yang dilakukan dalam `inspection_date` rentang, atau kueri database untuk inspeksi yang dilakukan pada tertentu yang `inspection_date` dibatasi oleh `inspector_id_last4` atau`inspector_id_last4.unit`, pertama-tama buat beacon standar yang berbeda untuk bidang dan. `inspector_id_last4` `unit` Kemudian, buat suar majemuk dari `inspection_date` bidang plaintext dan dua beacon standar.

Setelah Anda mengkonfigurasi suar majemuk, buat indeks sekunder yang menentukan suar majemuk sebagai kunci pengurutan. Gunakan indeks sekunder ini untuk melakukan kueri untuk inspeksi yang dilakukan pada tanggal tertentu oleh inspektur tertentu. Misalnya, Anda dapat menanyakan database untuk daftar semua unit yang diperiksa pada tanggal yang sama. Atau, Anda dapat menanyakan database untuk daftar semua inspeksi yang dilakukan pada unit tertentu di antara rentang tanggal inspeksi tertentu.

Untuk bantuan mengonfigurasi suar majemuk, lihat Mengonfigurasi suar majemuk.

Memilih panjang suar

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Saat Anda menulis nilai baru ke bidang terenkripsi yang dikonfigurasi untuk enkripsi yang dapat dicari, SDK Enkripsi AWS Database menghitung HMAC di atas nilai teks biasa. Output HMAC ini adalah kecocokan satu-ke-satu (1:1) untuk nilai plaintext dari bidang itu. Output HMAC terpotong sehingga beberapa nilai plaintext yang berbeda dipetakan ke tag HMAC terpotong yang sama. Tabrakan ini, atau positif palsu, membatasi kemampuan pengguna yang tidak sah untuk mengidentifikasi informasi yang membedakan tentang nilai plaintext.

Jumlah rata-rata positif palsu yang dihasilkan untuk setiap suar ditentukan oleh panjang suar yang tersisa setelah pemotongan. Anda hanya perlu menentukan panjang suar saat mengonfigurasi suar standar. Suar majemuk menggunakan panjang suar dari suar standar tempat mereka dibangun.

Beacon tidak mengubah status lapangan yang dienkripsi. Namun, ketika Anda menggunakan beacon, ada tradeoff yang melekat antara seberapa efisien kueri Anda dan seberapa banyak informasi yang terungkap tentang distribusi data Anda.

Tujuan enkripsi yang dapat dicari adalah untuk mengurangi biaya kinerja yang terkait dengan database terenkripsi sisi klien dengan menggunakan beacon untuk melakukan kueri pada data terenkripsi. Beacon disimpan di samping bidang terenkripsi tempat mereka dihitung. Ini berarti bahwa mereka dapat mengungkapkan informasi yang membedakan tentang distribusi dataset Anda. Dalam kasus ekstrim, pengguna yang tidak sah mungkin dapat menganalisis informasi yang diungkapkan tentang distribusi Anda dan menggunakannya untuk mengidentifikasi nilai plaintext bidang. Memilih panjang suar yang tepat dapat membantu mengurangi risiko ini dan menjaga kerahasiaan distribusi Anda.

Tinjau model ancaman Anda untuk menentukan tingkat keamanan yang Anda butuhkan. Misalnya, semakin banyak individu yang memiliki akses ke database Anda, tetapi tidak memiliki akses ke data teks biasa, semakin Anda mungkin ingin melindungi kerahasiaan distribusi kumpulan data Anda. Untuk meningkatkan kerahasiaan, suar perlu menghasilkan lebih banyak positif palsu. Peningkatan kerahasiaan menghasilkan kinerja kueri yang berkurang.

Keamanan vs Kinerja

- Panjang suar yang terlalu panjang menghasilkan terlalu sedikit positif palsu dan mungkin mengungkapkan informasi yang membedakan tentang distribusi kumpulan data Anda.
- Panjang suar yang terlalu pendek menghasilkan terlalu banyak positif palsu dan meningkatkan biaya kinerja kueri karena memerlukan pemindaian database yang lebih luas.

Saat menentukan panjang suar yang sesuai untuk solusi Anda, Anda harus menemukan panjang yang cukup menjaga keamanan data Anda tanpa memengaruhi kinerja kueri Anda lebih dari

yang mutlak diperlukan. Jumlah keamanan yang dipertahankan oleh suar tergantung pada distribusi kumpulan data Anda dan korelasi bidang tempat beacon Anda dibangun. Topik berikut mengasumsikan bahwa beacon Anda didistribusikan secara seragam dan tidak mengandung data yang berkorelasi.

Topik

- Menghitung panjang suar
- Contoh

Menghitung panjang suar

Panjang suar didefinisikan dalam bit dan mengacu pada jumlah bit tag HMAC yang disimpan setelah pemotongan. Panjang suar yang direkomendasikan bervariasi tergantung pada distribusi kumpulan data, keberadaan nilai yang berkorelasi, dan persyaratan keamanan dan kinerja spesifik Anda. Jika kumpulan data Anda terdistribusi secara seragam, Anda dapat menggunakan persamaan dan prosedur berikut untuk membantu mengidentifikasi panjang suar terbaik untuk implementasi Anda. Persamaan ini hanya memperkirakan jumlah rata-rata positif palsu yang akan dihasilkan suar, mereka tidak menjamin bahwa setiap nilai unik dalam kumpulan data Anda akan menghasilkan sejumlah positif palsu tertentu.

> **Note**
>
> Efektivitas persamaan ini tergantung pada distribusi dataset Anda. Jika kumpulan data Anda tidak terdistribusi secara seragam, lihat. Apakah beacon tepat untuk dataset saya? Secara umum, semakin jauh dataset Anda dari distribusi yang seragam, semakin Anda perlu mempersingkat panjang suar Anda.

1.

   Perkirakan populasi

   Populasi adalah jumlah nilai unik yang diharapkan di bidang tempat suar standar Anda dibangun, itu bukan jumlah total nilai yang diharapkan yang disimpan di lapangan. Misalnya, pertimbangkan `Room` bidang terenkripsi yang mengidentifikasi lokasi rapat karyawan. `Room`Bidang ini diharapkan menyimpan 100.000 nilai total, tetapi hanya ada 50 kamar berbeda yang dapat dipesan karyawan untuk rapat. Ini berarti bahwa populasinya adalah 50 karena hanya ada 50 kemungkinan nilai unik yang dapat disimpan di `Room` lapangan.

> **Note**
>
> Jika suar standar Anda dibangun dari bidang virtual, populasi yang digunakan untuk menghitung panjang suar adalah jumlah kombinasi unik yang dibuat oleh bidang virtual.

Saat memperkirakan populasi Anda, pastikan untuk mempertimbangkan proyeksi pertumbuhan kumpulan data. Setelah Anda menulis catatan baru dengan suar, Anda tidak dapat memperbarui panjang suar. Tinjau model ancaman Anda dan solusi database yang ada untuk membuat perkiraan jumlah nilai unik yang Anda harapkan untuk disimpan dalam lima tahun ke depan.

Populasi Anda tidak perlu tepat. Pertama, identifikasi jumlah nilai unik dalam database Anda saat ini, atau perkirakan jumlah nilai unik yang Anda harapkan untuk disimpan di tahun pertama. Selanjutnya, gunakan pertanyaan-pertanyaan berikut untuk membantu Anda menentukan proyeksi pertumbuhan nilai-nilai unik selama lima tahun ke depan.

- Apakah Anda mengharapkan nilai unik dikalikan dengan 10?
- Apakah Anda mengharapkan nilai unik dikalikan dengan 100?
- Apakah Anda mengharapkan nilai unik dikalikan dengan 1000?

Perbedaan antara 50.000 dan 60.000 nilai unik tidak signifikan dan keduanya akan menghasilkan panjang suar yang direkomendasikan yang sama. Namun, perbedaan antara 50.000 dan 500.000 nilai unik akan berdampak signifikan pada panjang suar yang direkomendasikan.

Pertimbangkan untuk meninjau data publik tentang frekuensi tipe data umum, seperti kode pos atau nama belakang. Misalnya, ada 41,707 kode pos di Amerika Serikat. Populasi yang Anda gunakan harus proporsional dengan database Anda sendiri. Jika `ZIPCode` bidang dalam database Anda menyertakan data dari seluruh Amerika Serikat, maka Anda dapat menentukan populasi Anda sebagai 41.707, bahkan jika bidang `ZIPCode` tersebut saat ini tidak memiliki 41.707 nilai unik. Jika `ZIPCode` bidang dalam database Anda hanya menyertakan data dari satu status, dan hanya akan menyertakan data dari satu status, maka Anda dapat mendefinisikan populasi Anda sebagai jumlah total kode pos dalam status tersebut, bukan 41.704.

2. Hitung rentang yang disarankan untuk jumlah tabrakan yang diharapkan

Untuk menentukan panjang suar yang sesuai untuk bidang tertentu, Anda harus terlebih dahulu mengidentifikasi rentang yang sesuai untuk jumlah tabrakan yang diharapkan. Jumlah tabrakan yang diharapkan mewakili jumlah rata-rata yang diharapkan dari nilai plaintext unik yang dipetakan ke tag HMAC tertentu. Jumlah positif palsu yang diharapkan untuk satu nilai plaintext unik adalah satu kurang dari jumlah tabrakan yang diharapkan.

Kami merekomendasikan bahwa jumlah tabrakan yang diharapkan lebih besar dari atau sama dengan dua, dan kurang dari akar kuadrat populasi Anda. Persamaan berikut hanya berfungsi jika populasi Anda memiliki 16 atau lebih nilai unik.

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

Jika jumlah tabrakan kurang dari dua, suar akan menghasilkan terlalu sedikit positif palsu. Kami merekomendasikan dua sebagai jumlah minimum tabrakan yang diharapkan karena itu berarti, rata-rata, setiap nilai unik di lapangan akan menghasilkan setidaknya satu positif palsu dengan memetakan ke satu nilai unik lainnya.

3. Hitung rentang yang disarankan untuk panjang suar

   Setelah mengidentifikasi jumlah minimum dan maksimum tabrakan yang diharapkan, gunakan persamaan berikut untuk mengidentifikasi rentang panjang suar yang sesuai.

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

   Pertama, selesaikan panjang suar di mana jumlah tabrakan yang diharapkan sama dengan dua (jumlah minimum tabrakan yang diharapkan).

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

   Kemudian, selesaikan panjang suar di mana jumlah tabrakan yang diharapkan sama dengan akar kuadrat populasi Anda (jumlah maksimum tabrakan yang diharapkan yang disarankan).

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

   Kami merekomendasikan untuk membulatkan output yang dihasilkan oleh persamaan ini ke panjang suar yang lebih pendek. Misalnya, jika persamaan menghasilkan panjang suar 15,6, kami sarankan untuk membulatkan nilai itu menjadi 15 bit alih-alih membulatkan hingga 16 bit.

4. Pilih panjang suar

   Persamaan ini hanya mengidentifikasi rentang panjang suar yang direkomendasikan untuk bidang Anda. Sebaiknya gunakan panjang suar yang lebih pendek untuk menjaga keamanan kumpulan data Anda bila memungkinkan. Namun, panjang suar yang sebenarnya Anda gunakan ditentukan oleh model ancaman Anda. Pertimbangkan persyaratan kinerja Anda saat Anda meninjau model ancaman Anda untuk menentukan panjang suar terbaik untuk bidang Anda.

   Menggunakan panjang suar yang lebih pendek mengurangi kinerja kueri, sementara menggunakan panjang suar yang lebih panjang mengurangi keamanan. Secara umum, jika kumpulan data Anda tidak terdistribusi secara merata, atau jika Anda membuat suar yang berbeda dari bidang yang berkorelasi, Anda perlu menggunakan panjang suar yang lebih pendek untuk meminimalkan jumlah informasi yang diungkapkan tentang distribusi kumpulan data Anda.

   Jika Anda meninjau model ancaman Anda dan memutuskan bahwa informasi pembeda apa pun yang diungkapkan tentang distribusi bidang tidak menimbulkan ancaman terhadap keamanan Anda secara keseluruhan, Anda dapat memilih untuk menggunakan panjang suar yang lebih panjang dari rentang yang disarankan yang Anda hitung. Misalnya, jika Anda menghitung rentang panjang suar yang disarankan untuk bidang sebagai 9-16 bit, Anda dapat memilih untuk menggunakan panjang suar 24 bit untuk menghindari kehilangan kinerja.

   Pilih panjang suar Anda dengan hati-hati. Setelah Anda menulis catatan baru dengan suar, Anda tidak dapat memperbarui panjang suar.

Contoh

Pertimbangkan database yang menandai `unit` bidang seperti `ENCRYPT_AND_SIGN` dalam tindakan kriptografi. Untuk mengkonfigurasi suar standar untuk `unit` bidang, kita perlu menentukan jumlah positif palsu dan panjang suar yang diharapkan untuk bidang tersebut. `unit`

1. Perkirakan populasi

   Setelah meninjau model ancaman kami dan solusi database saat ini, kami berharap `unit` bidang tersebut pada akhirnya memiliki 100.000 nilai unik.

   Ini berarti bahwa Populasi = 100.000.

2. Hitung rentang yang disarankan untuk jumlah tabrakan yang diharapkan.

   Untuk contoh ini, jumlah tabrakan yang diharapkan harus antara 2-316.

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

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

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

3. Hitung kisaran yang disarankan untuk panjang suar.

   Untuk contoh ini, panjang suar harus antara 9-16 bit.

   ```
   number of collisions = Population * 2^-(beacon length)
   ```

   a. Hitung panjang suar di mana jumlah tabrakan yang diharapkan sama dengan minimum yang diidentifikasi pada Langkah 2.

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

      Panjang suar = 15,6, atau 15 bit

   b. Hitung panjang suar di mana jumlah tabrakan yang diharapkan sama dengan maksimum yang diidentifikasi pada Langkah 2.

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

      Panjang suar = 8,3, atau 8 bit

4. Tentukan panjang suar yang sesuai dengan persyaratan keamanan dan kinerja Anda.

   Untuk setiap bit di bawah 15, biaya kinerja dan keamanan berlipat ganda.

   - 16 bit
     - Rata-rata, setiap nilai unik akan dipetakan ke 1,5 unit lainnya.
     - Keamanan: dua catatan dengan tag HMAC terpotong yang sama 66% kemungkinan memiliki nilai plaintext yang sama.
     - Kinerja: kueri akan mengambil 15 catatan untuk setiap 10 catatan yang sebenarnya Anda minta.
   - 14 bit

- Rata-rata, setiap nilai unik akan dipetakan menjadi 6,1 unit lainnya.
- Keamanan: dua catatan dengan tag HMAC terpotong yang sama adalah 33% kemungkinan memiliki nilai plaintext yang sama.
- Kinerja: kueri akan mengambil 30 catatan untuk setiap 10 catatan yang sebenarnya Anda minta.

Memilih nama suar

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Setiap suar diidentifikasi dengan nama suar yang unik. Setelah suar dikonfigurasi, nama suar adalah nama yang Anda gunakan saat menanyakan bidang terenkripsi. Nama suar dapat berupa nama yang sama dengan bidang terenkripsi atau bidang virtual, tetapi tidak bisa nama yang sama dengan bidang yang tidak terenkripsi. Dua beacon yang berbeda tidak dapat memiliki nama suar yang sama.

Untuk contoh yang mendemonstrasikan cara memberi nama dan mengkonfigurasi beacon, lihat Mengkonfigurasi beacon.

Penamaan suar standar

Saat menamai suar standar, kami sangat menyarankan agar nama suar Anda diselesaikan ke sumber suar bila memungkinkan. Ini berarti bahwa nama suar dan nama bidang terenkripsi atau virtual tempat suar standar Anda dibangun adalah sama. Misalnya, jika Anda membuat suar standar untuk bidang terenkripsi bernama`LastName`, nama suar Anda juga harus. `LastName`

Ketika nama suar Anda sama dengan sumber suar, Anda dapat menghilangkan sumber suar dari konfigurasi Anda dan SDK Enkripsi AWS Database akan secara otomatis menggunakan nama suar sebagai sumber suar.

Mengkonfigurasi beacon

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Ada dua jenis beacon yang mendukung enkripsi yang dapat dicari. Beacon standar melakukan pencarian kesetaraan. Mereka adalah cara paling sederhana untuk menerapkan enkripsi yang dapat dicari di database Anda. Compound beacon menggabungkan string plaintext literal dan beacon standar untuk melakukan kueri yang lebih kompleks.

Beacon dirancang untuk diimplementasikan dalam database baru yang tidak berpenghuni. Setiap suar yang dikonfigurasi dalam database yang ada hanya akan memetakan catatan baru yang ditulis ke database. Beacon dihitung dari nilai plaintext bidang, setelah bidang dienkripsi, tidak ada cara bagi beacon untuk memetakan data yang ada. Setelah Anda menulis catatan baru dengan suar, Anda tidak dapat memperbarui konfigurasi suar. Namun, Anda dapat menambahkan beacon baru untuk bidang baru yang Anda tambahkan ke catatan Anda.

Setelah menentukan pola akses Anda, mengonfigurasi beacon harus menjadi langkah kedua dalam implementasi database Anda. Kemudian, setelah Anda mengkonfigurasi semua beacon Anda, Anda perlu membuat keyring AWS KMS Hierarkis, menentukan versi suar, mengonfigurasi indeks sekunder untuk setiap suar, menentukan tindakan kriptografi Anda, dan mengkonfigurasi database dan klien SDK Enkripsi Database Anda. AWS Untuk informasi selengkapnya, lihat Menggunakan beacon.

Untuk membuatnya lebih mudah untuk menentukan versi beacon, kami sarankan membuat daftar untuk beacon standar dan majemuk. Tambahkan setiap suar yang Anda buat ke daftar suar standar atau gabungan masing-masing saat Anda mengonfigurasinya.

Topik

- Mengkonfigurasi beacon standar
- Mengkonfigurasi suar majemuk
- Contoh konfigurasi

Mengkonfigurasi beacon standar

Beacon standar adalah cara paling sederhana untuk mengimplementasikan enkripsi yang dapat dicari di database Anda. Mereka hanya dapat melakukan pencarian kesetaraan untuk satu bidang terenkripsi atau virtual.

Contoh sintaks konfigurasi

Java

```
List<StandardBeacon> standardBeaconList = new ArrayList<>();
```

```
StandardBeacon exampleStandardBeacon = StandardBeacon.builder()
    .name("beaconName")
    .length(beaconLengthInBits)
    .build();
standardBeaconList.add(exampleStandardBeacon);
```

C# / .NET

```
var standardBeaconList = new List<StandardBeacon>();
StandardBeacon exampleStandardBeacon = new StandardBeacon
  {
    Name = "beaconName",
    Length = 10
  };
standardBeaconList.Add(exampleStandardBeacon);
```

Rust

```
let standard_beacon_list = vec![

 StandardBeacon::builder().name("beacon_name").length(beacon_length_in_bits).build()?,
```

Untuk mengkonfigurasi suar standar, berikan nilai berikut.

Nama suar

Nama yang Anda gunakan saat menanyakan bidang terenkripsi.

Nama suar dapat berupa nama yang sama dengan bidang terenkripsi atau bidang virtual, tetapi tidak bisa nama yang sama dengan bidang yang tidak terenkripsi. Kami sangat menyarankan untuk menggunakan nama bidang terenkripsi atau bidang virtual tempat suar standar Anda dibangun kapan pun memungkinkan. Dua beacon yang berbeda tidak dapat memiliki nama suar yang sama. Untuk bantuan menentukan nama beacon terbaik untuk implementasi Anda, lihat Memilih nama suar.

Panjang suar

Jumlah bit dari nilai hash beacon yang disimpan setelah pemotongan.

Panjang suar menentukan jumlah rata-rata positif palsu yang dihasilkan oleh suar yang diberikan. Untuk informasi selengkapnya dan membantu menentukan panjang suar yang sesuai untuk implementasi Anda, lihat Menentukan panjang suar.

Sumber suar (Opsional)

Bidang tempat suar standar dibangun.

Sumber suar harus berupa nama bidang atau indeks yang mengacu pada nilai bidang bersarang. Ketika nama suar Anda sama dengan sumber suar, Anda dapat menghilangkan sumber suar dari konfigurasi Anda dan SDK Enkripsi AWS Database akan secara otomatis menggunakan nama suar sebagai sumber suar.

Membuat bidang virtual

Untuk membuat bidang virtual, Anda harus memberikan nama untuk bidang virtual dan daftar bidang sumber. Urutan yang Anda tambahkan bidang sumber ke daftar bagian virtual menentukan urutan penggabungannya untuk membangun bidang virtual. Contoh berikut menggabungkan dua bidang sumber secara keseluruhan untuk membuat bidang virtual.

> **Note**
>
> Kami menyarankan untuk memverifikasi bahwa bidang virtual Anda menghasilkan hasil yang diharapkan sebelum Anda mengisi database Anda. Untuk informasi selengkapnya, lihat Menguji output suar.

Java

Lihat contoh kode lengkapnya: VirtualBeaconSearchableEncryptionExample.java

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

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

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

```
    virtualFieldList.add(virtualFieldName);
```

C# / .NET

Lihat contoh kode lengkapnya: VirtualBeaconSearchableEncryptionExample.cs

```
var virtualPartList = new List<VirtualPart> { sourceField1, sourceField2 };

var virtualFieldName = new VirtualField
{
    Name = "virtualFieldName",
    Parts = virtualPartList
};

var virtualFieldList = new List<VirtualField> { virtualFieldName };
```

Rust

Lihat contoh kode lengkap: virtual_beacon_searchable_encryption.rs

```
let virtual_part_list = vec![source_field_one, source_field_two];

let state_and_has_test_result_field = VirtualField::builder()
    .name("virtual_field_name")
    .parts(virtual_part_list)
    .build()?;

let virtual_field_list = vec![virtual_field_name];
```

Untuk membuat bidang virtual dengan segmen tertentu dari bidang sumber, Anda harus menentukan transformasi itu sebelum menambahkan bidang sumber ke daftar bagian virtual Anda.

Pertimbangan keamanan untuk bidang virtual

Beacon tidak mengubah status lapangan yang dienkripsi. Namun, ketika Anda menggunakan beacon, ada tradeoff yang melekat antara seberapa efisien kueri Anda dan seberapa banyak informasi yang terungkap tentang distribusi data Anda. Cara Anda mengonfigurasi suar Anda menentukan tingkat keamanan yang dipertahankan oleh suar itu.

Hindari membuat bidang virtual dengan bidang sumber yang tumpang tindih dengan beacon standar yang ada. Membuat bidang virtual yang menyertakan bidang sumber yang telah digunakan untuk

membuat suar standar dapat mengurangi tingkat keamanan untuk kedua beacon. Sejauh mana keamanan berkurang tergantung pada tingkat entropi yang ditambahkan oleh bidang sumber tambahan. Tingkat entropi ditentukan oleh distribusi nilai unik di bidang sumber tambahan dan jumlah bit yang disumbangkan oleh bidang sumber tambahan pada ukuran keseluruhan bidang virtual.

Anda dapat menggunakan populasi dan panjang suar untuk menentukan apakah bidang sumber untuk bidang virtual menjaga keamanan kumpulan data Anda. Populasi adalah jumlah nilai unik yang diharapkan dalam suatu bidang. Populasi Anda tidak perlu tepat. Untuk bantuan memperkirakan populasi suatu bidang, lihat Memperkirakan populasi.

Pertimbangkan contoh berikut saat Anda meninjau keamanan bidang virtual Anda.

- Beacon1 dibangun dari. `FieldA` `FieldA`memiliki populasi lebih besar dari $2^{\text{(panjang Beacon1)}}$.
- Beacon2 dibangun dari`VirtualField`, yang dibangun dari,`FieldA`,`FieldB`, `FieldC` dan. `FieldD` Bersama-sama`FieldB`,`FieldC`,, dan `FieldD` memiliki populasi lebih besar dari 2^{N}

Beacon2 menjaga keamanan Beacon1 dan Beacon2 jika pernyataan berikut benar:

```
N ≥ (Beacon1 length)/2
```

and

```
N ≥ (Beacon2 length)/2
```

Mendefinisikan gaya suar

Beacon standar dapat digunakan untuk melakukan pencarian kesetaraan untuk bidang terenkripsi atau virtual. Atau, mereka dapat digunakan untuk membangun suar majemuk untuk melakukan operasi database yang lebih kompleks. Untuk membantu Anda mengatur dan mengelola beacon standar, AWS Database Encryption SDK menyediakan gaya beacon opsional berikut yang menentukan tujuan penggunaan suar standar.

> **Note**
>
> Untuk menentukan gaya suar, Anda harus menggunakan SDK Enkripsi AWS Database versi 3.2 atau yang lebih baru. Terapkan versi baru ke semua pembaca sebelum menambahkan gaya suar ke konfigurasi suar Anda.

PartOnly

Sebuah beacon standar didefinisikan sebagai hanya `PartOnly` dapat digunakan untuk mendefinisikan bagian terenkripsi dari suar majemuk. Anda tidak dapat langsung menanyakan suar `PartOnly` standar.

Java

```
List<StandardBeacon> standardBeaconList = new ArrayList<>();
StandardBeacon exampleStandardBeacon = StandardBeacon.builder()
    .name("beaconName")
    .length(beaconLengthInBits)
    .style(
        BeaconStyle.builder()
           .partOnly(PartOnly.builder().build())
        .build()
    )
    .build();
standardBeaconList.add(exampleStandardBeacon);
```

C #/.NET

```
new StandardBeacon
{
    Name = "beaconName",
    Length = beaconLengthInBits,
    Style = new BeaconStyle
    {
        PartOnly = new PartOnly()
    }
}
```

Rust

```
StandardBeacon::builder()
    .name("beacon_name")
    .length(beacon_length_in_bits)
    .style(BeaconStyle::PartOnly(PartOnly::builder().build()?))
    .build()?
```

Shared

Secara default, setiap suar standar menghasilkan kunci HMAC unik untuk perhitungan suar. Akibatnya, Anda tidak dapat melakukan pencarian kesetaraan pada bidang terenkripsi dari dua suar standar terpisah. Sebuah suar standar didefinisikan sebagai `Shared` menggunakan kunci HMAC dari suar standar lain untuk perhitungannya.

Misalnya, jika Anda perlu membandingkan `beacon1` bidang dengan `beacon2` bidang, tentukan `beacon2` sebagai `Shared` suar yang menggunakan kunci HMAC dari `beacon1` untuk perhitungannya.

> **Note**
>
> Pertimbangkan kebutuhan keamanan dan kinerja Anda sebelum mengonfigurasi `Shared` beacon apa pun. `Shared`beacon dapat meningkatkan jumlah informasi statistik yang dapat diidentifikasi tentang distribusi dataset Anda. Misalnya, mereka mungkin mengungkapkan bidang bersama mana yang berisi nilai plaintext yang sama.

Java

```
List<StandardBeacon> standardBeaconList = new ArrayList<>();
StandardBeacon exampleStandardBeacon = StandardBeacon.builder()
    .name("beacon2")
    .length(beaconLengthInBits)
    .style(
        BeaconStyle.builder()
           .shared(Shared.builder().other("beacon1").build())
        .build()
    )
    .build();
standardBeaconList.add(exampleStandardBeacon);
```

C #/.NET

```
new StandardBeacon
{
    Name = "beacon2",
    Length = beaconLengthInBits,
    Style = new BeaconStyle
    {
```

```
        Shared = new Shared { Other = "beacon1" }
    }
}
```

Rust

```
StandardBeacon::builder()
    .name("beacon2")
    .length(beacon_length_in_bits)
    .style(BeaconStyle::Shared(
       Shared::builder().other("beacon1").build()?,
    ))
    .build()?
```

AsSet

Secara default, jika nilai bidang adalah satu set, SDK Enkripsi AWS Database menghitung satu suar standar untuk set tersebut. Akibatnya, Anda tidak dapat melakukan kueri di `CONTAINS(a, :value)` mana *a* adalah bidang terenkripsi. Sebuah suar standar didefinisikan sebagai `AsSet` menghitung nilai beacon standar individu untuk setiap elemen individu dari himpunan dan menyimpan nilai beacon dalam item sebagai satu set. Ini memungkinkan SDK Enkripsi AWS Database untuk melakukan kueri`CONTAINS(a, :value)`.

Untuk menentukan suar `AsSet` standar, elemen dalam himpunan harus dari populasi yang sama sehingga mereka semua dapat menggunakan panjang suar yang sama. Set beacon mungkin memiliki elemen lebih sedikit daripada set plaintext jika ada tabrakan saat menghitung nilai beacon.

> **Note**
>
> Pertimbangkan kebutuhan keamanan dan kinerja Anda sebelum mengonfigurasi `AsSet` beacon apa pun. `AsSet`beacon dapat meningkatkan jumlah informasi statistik yang dapat diidentifikasi tentang distribusi dataset Anda. Misalnya, mereka mungkin mengungkapkan ukuran set plaintext.

Java

```
List<StandardBeacon> standardBeaconList = new ArrayList<>();
```

```
StandardBeacon exampleStandardBeacon = StandardBeacon.builder()
    .name("beaconName")
    .length(beaconLengthInBits)
    .style(
        BeaconStyle.builder()
           .asSet(AsSet.builder().build())
        .build()
    )
    .build();
standardBeaconList.add(exampleStandardBeacon);
```

C #/.NET

```
new StandardBeacon
{
    Name = "beaconName",
    Length = beaconLengthInBits,
    Style = new BeaconStyle
    {
        AsSet = new AsSet()
    }
}
```

Rust

```
StandardBeacon::builder()
    .name("beacon_name")
    .length(beacon_length_in_bits)
    .style(BeaconStyle::AsSet(AsSet::builder().build()?))
    .build()?
```

SharedSet

Sebuah beacon standar didefinisikan sebagai `SharedSet` menggabungkan `Shared` dan `AsSet` fungsi sehingga Anda dapat melakukan pencarian kesetaraan pada nilai terenkripsi dari set dan bidang. Ini memungkinkan SDK Enkripsi AWS Database untuk melakukan kueri `CONTAINS(a, b)` di mana *a* kumpulan terenkripsi dan *b* merupakan bidang terenkripsi.

> **Note**
>
> Pertimbangkan kebutuhan keamanan dan kinerja Anda sebelum mengonfigurasi `Shared` beacon apa pun. `SharedSet`beacon dapat meningkatkan jumlah informasi statistik yang dapat diidentifikasi tentang distribusi dataset Anda. Misalnya, mereka mungkin mengungkapkan ukuran set teks biasa atau bidang bersama mana yang berisi nilai plaintext yang sama.

Java

```
List<StandardBeacon> standardBeaconList = new ArrayList<>();
StandardBeacon exampleStandardBeacon = StandardBeacon.builder()
    .name("beacon2")
    .length(beaconLengthInBits)
    .style(
        BeaconStyle.builder()
           .sharedSet(SharedSet.builder().other("beacon1").build())
        .build()
    )
    .build();
standardBeaconList.add(exampleStandardBeacon);
```

C #/.NET

```
new StandardBeacon
{
    Name = "beacon2",
    Length = beaconLengthInBits,
    Style = new BeaconStyle
    {
        SharedSet = new SharedSet { Other = "beacon1" }
    }
}
```

Rust

```
StandardBeacon::builder()
    .name("beacon2")
    .length(beacon_length_in_bits)
    .style(BeaconStyle::SharedSet(
```

```
        SharedSet::builder().other("beacon1").build()?,
    ))
    .build()?
```

Mengkonfigurasi suar majemuk

Compound beacon menggabungkan string plaintext literal dan beacon standar untuk melakukan operasi database yang kompleks, seperti menanyakan dua jenis rekaman yang berbeda dari indeks tunggal atau menanyakan kombinasi bidang dengan kunci pengurutan. Suar majemuk dapat dibangun dari`ENCRYPT_AND_SIGN`,`SIGN_ONLY`, dan `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` ladang. Anda harus membuat suar standar untuk setiap bidang terenkripsi yang termasuk dalam suar majemuk.

> **Note**
>
> Sebaiknya verifikasi bahwa suar majemuk Anda menghasilkan hasil yang diharapkan sebelum Anda mengisi basis data Anda. Untuk informasi selengkapnya, lihat Menguji output suar.

Contoh sintaks konfigurasi

Java

Konfigurasi suar majemuk

Contoh berikut mendefinisikan daftar bagian terenkripsi dan ditandatangani secara lokal dalam konfigurasi suar majemuk.

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

Definisi versi suar

Contoh berikut mendefinisikan daftar bagian terenkripsi dan ditandatangani secara global dalam versi beacon. Untuk informasi selengkapnya tentang mendefinisikan versi beacon, lihat Menggunakan beacon.

```
 List<BeaconVersion> beaconVersions = new ArrayList<>();
beaconVersions.add(
    BeaconVersion.builder()
        .standardBeacons(standardBeaconList)
        .compoundBeacons(compoundBeaconList)
        .encryptedParts(encryptedPartList)
        .signedParts(signedPartList)
        .version(1) // MUST be 1
        .keyStore(keyStore)
        .keySource(BeaconKeySource.builder()
            .single(SingleKeyStore.builder()
                .keyId(branchKeyId)
                .cacheTTL(6000)
                .build())
            .build())
        .build()
);
```

C# / .NET

Lihat contoh kode lengkapnya: BeaconConfig.cs

Konfigurasi suar majemuk

Contoh berikut mendefinisikan daftar bagian terenkripsi dan ditandatangani secara lokal dalam konfigurasi suar majemuk.

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

Definisi versi suar

Contoh berikut mendefinisikan daftar bagian terenkripsi dan ditandatangani secara global dalam versi beacon. Untuk informasi selengkapnya tentang mendefinisikan versi beacon, lihat Menggunakan beacon.

```
var beaconVersions = new List<BeaconVersion>
{
    new BeaconVersion
    {
        StandardBeacons = standardBeaconList,
        CompoundBeacons = compoundBeaconList,
        EncryptedParts = encryptedPartsList,
        SignedParts = signedPartsList,
        Version = 1, // MUST be 1
        KeyStore = keyStore,
        KeySource = new BeaconKeySource
        {
            Single = new SingleKeyStore
            {
                KeyId = branchKeyId,
                CacheTTL = 6000
            }
        }
    }
};
```

Rust

Lihat contoh kode lengkap: beacon_config.rs

Konfigurasi suar majemuk

Contoh berikut mendefinisikan daftar bagian terenkripsi dan ditandatangani secara lokal dalam konfigurasi suar majemuk.

```
let compound_beacon_list = vec![
    CompoundBeacon::builder()
        .name("compound_beacon_name")
        .split(".")
        .encrypted(encrypted_parts_list)
        .signed(signed_parts_list)
        .constructors(constructor_list)
        .build()?
```

Definisi versi suar

Contoh berikut mendefinisikan daftar bagian terenkripsi dan ditandatangani secara global dalam versi beacon. Untuk informasi selengkapnya tentang mendefinisikan versi beacon, lihat Menggunakan beacon.

```
let beacon_versions = BeaconVersion::builder()
    .standard_beacons(standard_beacon_list)
    .compound_beacons(compound_beacon_list)
    .encrypted_parts(encrypted_parts_list)
    .signed_parts(signed_parts_list)
    .version(1) // MUST be 1
    .key_store(key_store.clone())
    .key_source(BeaconKeySource::Single(
        SingleKeyStore::builder()
            .key_id(branch_key_id)
            .cache_ttl(6000)
            .build()?,
    ))
    .build()?;
let beacon_versions = vec![beacon_versions];
```

Anda dapat menentukan bagian terenkripsi dan bagian yang ditandatangani dalam daftar yang ditentukan secara lokal atau global. Kami merekomendasikan untuk menentukan bagian terenkripsi dan ditandatangani Anda dalam daftar global dalam versi suar bila memungkinkan. Dengan mendefinisikan bagian terenkripsi dan ditandatangani secara global, Anda dapat menentukan setiap bagian sekali dan kemudian menggunakan kembali bagian-bagian tersebut dalam beberapa konfigurasi suar majemuk. Jika Anda hanya ingin menggunakan bagian terenkripsi atau ditandatangani sekali, Anda dapat mendefinisikannya dalam daftar lokal dalam konfigurasi suar majemuk. Anda dapat mereferensikan bagian lokal dan global dalam daftar konstruktor Anda.

Jika Anda menentukan daftar bagian terenkripsi dan ditandatangani secara global, Anda harus memberikan daftar bagian konstruktor yang mengidentifikasi semua kemungkinan cara suar majemuk dapat merakit bidang dalam konfigurasi suar majemuk Anda.

> **Note**
>
> Untuk menentukan daftar bagian terenkripsi dan ditandatangani secara global, Anda harus menggunakan SDK Enkripsi AWS Database versi 3.2 atau yang lebih baru. Terapkan versi baru ke semua pembaca sebelum mendefinisikan bagian baru secara global.

Anda tidak dapat memperbarui konfigurasi suar yang ada untuk menentukan daftar bagian terenkripsi dan ditandatangani secara global.

Untuk mengkonfigurasi suar majemuk, berikan nilai berikut.

Nama suar

Nama yang Anda gunakan saat menanyakan bidang terenkripsi.

Nama suar dapat berupa nama yang sama dengan bidang terenkripsi atau bidang virtual, tetapi tidak bisa nama yang sama dengan bidang yang tidak terenkripsi. Tidak ada dua suar yang dapat memiliki nama suar yang sama. Untuk bantuan menentukan nama beacon terbaik untuk implementasi Anda, lihat Memilih nama suar.

Karakter split

Karakter yang digunakan untuk memisahkan bagian-bagian yang membentuk suar majemuk Anda.

Karakter split tidak dapat muncul dalam nilai plaintext dari salah satu bidang tempat suar majemuk dibangun.

Daftar bagian terenkripsi

Mengidentifikasi `ENCRYPT_AND_SIGN` bidang yang termasuk dalam suar majemuk.

Setiap bagian harus menyertakan nama dan awalan. Nama bagian harus merupakan nama suar standar yang dibangun dari bidang terenkripsi. Awalan dapat berupa string apa saja, tetapi harus unik. Bagian terenkripsi tidak dapat memiliki awalan yang sama dengan bagian yang ditandatangani. Sebaiknya gunakan nilai pendek yang membedakan bagian dari bagian lain yang dilayani oleh suar majemuk.

Kami merekomendasikan untuk menentukan bagian terenkripsi Anda secara global bila memungkinkan. Anda dapat mempertimbangkan untuk mendefinisikan bagian terenkripsi secara lokal jika Anda hanya bermaksud menggunakannya dalam satu suar majemuk. Bagian terenkripsi yang didefinisikan secara lokal tidak dapat memiliki awalan atau nama yang sama dengan bagian terenkripsi yang didefinisikan secara global.

Java

```
List<EncryptedPart> encryptedPartList = new ArrayList<>);
EncryptedPart encryptedPartExample = EncryptedPart.builder()
```

```
    .name("standardBeaconName")
    .prefix("E-")
    .build();
encryptedPartList.add(encryptedPartExample);
```

C# / .NET

```
var encryptedPartList = new List<EncryptedPart>();
var encryptedPartExample = new EncryptedPart
 {
    Name = "compoundBeaconName",
    Prefix = "E-"
 };
encryptedPartList.Add(encryptedPartExample);
```

Rust

```
let encrypted_parts_list = vec![
    EncryptedPart::builder()
        .name("standard_beacon_name")
        .prefix("E-")
        .build()?
];
```

Daftar bagian yang ditandatangani

Mengidentifikasi bidang yang ditandatangani termasuk dalam suar majemuk.

> **Note**
>
> Bagian yang ditandatangani adalah opsional. Anda dapat mengonfigurasi suar majemuk yang tidak mereferensikan bagian yang ditandatangani.

Setiap bagian harus menyertakan nama, sumber, dan awalan. Sumbernya adalah `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` bidang `SIGN_ONLY` atau yang diidentifikasi oleh bagian tersebut. Sumber harus berupa nama bidang atau indeks yang mengacu pada nilai bidang bersarang. Jika nama bagian Anda mengidentifikasi sumber, Anda dapat menghilangkan sumber dan SDK Enkripsi AWS Database akan secara otomatis menggunakan nama sebagai

sumbernya. Kami merekomendasikan untuk menentukan sumber sebagai nama bagian bila memungkinkan. Awalan dapat berupa string apa saja, tetapi harus unik. Bagian yang ditandatangani tidak dapat memiliki awalan yang sama dengan bagian terenkripsi. Sebaiknya gunakan nilai pendek yang membedakan bagian dari bagian lain yang dilayani oleh suar majemuk.

Kami merekomendasikan untuk menentukan suku cadang Anda yang ditandatangani secara global bila memungkinkan. Anda dapat mempertimbangkan untuk mendefinisikan bagian yang ditandatangani secara lokal jika Anda hanya bermaksud menggunakannya dalam satu suar majemuk. Bagian yang ditandatangani secara lokal tidak dapat memiliki awalan atau nama yang sama dengan bagian ditandatangani yang ditentukan secara global.

Java

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

C# / .NET

```
var signedPartsList = new List<SignedPart>
{
    new SignedPart { Name = "signedFieldName1", Prefix = "S-" },
    new SignedPart { Name = "signedFieldName2", Prefix = "SF-" }
};
```

Rust

```
let signed_parts_list = vec![
    SignedPart::builder()
        .name("signed_field_name_1")
        .prefix("S-")
        .build()?,
   SignedPart::builder()
        .name("signed_field_name_2")
        .prefix("SF-")
        .build()?,
];
```

Daftar konstruktor

Mengidentifikasi konstruktor yang menentukan cara berbeda bahwa bagian terenkripsi dan ditandatangani dapat dirakit oleh suar majemuk. Anda dapat mereferensikan bagian lokal dan global dalam daftar konstruktor Anda.

Jika Anda membangun suar majemuk Anda dari bagian terenkripsi dan ditandatangani yang didefinisikan secara global, Anda harus memberikan daftar konstruktor.

Jika Anda tidak menggunakan bagian terenkripsi atau ditandatangani yang didefinisikan secara global untuk membangun suar majemuk Anda, daftar konstruktor bersifat opsional. Jika Anda tidak menentukan daftar konstruktor, AWS Database Encryption SDK merakit suar majemuk dengan konstruktor default berikut.

- Semua bagian yang ditandatangani dalam urutan mereka ditambahkan ke daftar bagian yang ditandatangani
- Semua bagian terenkripsi dalam urutan mereka ditambahkan ke daftar bagian terenkripsi
- Semua bagian diperlukan

Konstruktor

Setiap konstruktor adalah daftar terurut dari bagian-bagian konstruktor yang mendefinisikan satu cara bahwa suar majemuk dapat dirakit. Bagian konstruktor digabungkan bersama dalam urutan mereka ditambahkan ke daftar, dengan setiap bagian dipisahkan oleh karakter split yang ditentukan.

Setiap bagian konstruktor menamai bagian terenkripsi atau bagian yang ditandatangani, dan menentukan apakah bagian itu diperlukan atau opsional dalam konstruktor. Misalnya, jika Anda ingin menanyakan suar majemuk pada`Field1`,, dan `Field1.Field2Field1.Field2.Field3`, tandai dan `Field3` sebagai opsional `Field2` dan buat satu konstruktor.

Setiap konstruktor harus memiliki setidaknya satu bagian yang diperlukan. Sebaiknya buat bagian pertama di setiap konstruktor yang diperlukan sehingga Anda dapat menggunakan `BEGINS_WITH` operator dalam kueri Anda.

Konstruktor berhasil jika semua bagian yang diperlukan ada dalam catatan. Saat Anda menulis catatan baru, suar majemuk menggunakan daftar konstruktor untuk menentukan apakah suar dapat dirakit dari nilai yang diberikan. Ini mencoba untuk merakit suar dalam urutan bahwa konstruktor ditambahkan ke daftar konstruktor, dan menggunakan konstruktor pertama yang berhasil. Jika tidak ada konstruktor yang berhasil, suar tidak ditulis ke catatan.

Semua pembaca dan penulis harus menentukan urutan konstruktor yang sama untuk memastikan bahwa hasil kueri mereka benar.

Gunakan prosedur berikut untuk menentukan daftar konstruktor Anda sendiri.

1. Buat bagian konstruktor untuk setiap bagian terenkripsi dan bagian yang ditandatangani untuk menentukan apakah bagian itu diperlukan atau tidak.

   Nama bagian konstruktor harus nama suar standar atau bidang bertanda tangan yang diwakilinya.

   Java

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

   C# / .NET

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

   Rust

   ```
   let field_1_constructor_part = ConstructorPart::builder()
       .name("field_1")
       .required(true)
       .build()?;
   ```

2. Buat konstruktor untuk setiap cara yang mungkin bahwa suar majemuk dapat dirakit menggunakan bagian konstruktor yang Anda buat di Langkah 1.

   Misalnya, jika Anda ingin menanyakan `Field1.Field2.Field3` dan`Field4.Field2.Field3`, maka Anda harus membuat dua konstruktor. `Field1`dan keduanya `Field4` dapat diperlukan karena mereka didefinisikan dalam dua konstruktor terpisah.

   Java

   ```
   // Create a list for Field1.Field2.Field3 queries
   List<ConstructorPart> field123ConstructorPartList = new ArrayList<>();
   ```

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

C# / .NET

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

Rust

```
// Create a list for field1.field2.field3 queries
let field1_field2_field3_constructor = Constructor::builder()
    .parts(vec![
        field1_constructor_part,
        field2_constroctor_part.clone(),
        field3_constructor_part,
    ])
    .build()?;

// Create a list for field4.field2.field1 queries
let field4_field2_field1_constructor = Constructor::builder()
```

```
    .parts(vec![
        field4_constructor_part,
        field2_constroctor_part.clone(),
        field1_constructor_part,
    ])
    .build()?;
```

3. Buat daftar konstruktor yang mencakup semua konstruktor yang Anda buat di Langkah 2.

   Java

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

   C# / .NET

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

   Rust

   ```
   let constructor_list = vec![
       field1_field2_field3_constructor,
       field4_field2_field1_constructor,
   ];
   ```

4. Tentukan `constructorList` kapan Anda membuat suar majemuk Anda.

Contoh konfigurasi

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Contoh berikut menunjukkan cara mengkonfigurasi beacon standar dan majemuk. Konfigurasi berikut tidak memberikan panjang suar. Untuk bantuan menentukan panjang suar yang sesuai untuk konfigurasi Anda, lihat Memilih panjang suar.

Untuk melihat contoh kode lengkap yang menunjukkan cara mengkonfigurasi dan menggunakan beacon, lihat contoh enkripsi Java, .NET, dan Rust yang dapat dicari di repositori -dynamodb aktif. aws-database-encryption-sdk GitHub

Topik

- Beacon standar
- Suar majemuk

Beacon standar

Jika Anda ingin menanyakan `inspector_id_last4` bidang untuk kecocokan persis, buat suar standar menggunakan konfigurasi berikut.

Java

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

C# / .NET

```
var standardBeaconList = new List<StandardBeacon>>);
StandardBeacon exampleStandardBeacon = new StandardBeacon
  {
    Name = "inspector_id_last4",
    Length = 10
  };
standardBeaconList.Add(exampleStandardBeacon);
```

Rust

```
let last4_beacon = StandardBeacon::builder()
```

```
    .name("inspector_id_last4")
    .length(10)
    .build()?;

let unit_beacon = StandardBeacon::builder().name("unit").length(30).build()?;

let standard_beacon_list = vec![last4_beacon, unit_beacon];
```

Suar majemuk

Jika Anda ingin menanyakan `UnitInspection` database `inspector_id_last4` dan`inspector_id_last4.unit`, buat suar majemuk dengan konfigurasi berikut. Suar majemuk ini hanya membutuhkan bagian terenkripsi.

Java

```
// 1. Create standard beacons for the inspector_id_last4 and unit fields.
List<StandardBeacon> standardBeaconList = new ArrayList<>);
StandardBeacon inspectorBeacon = StandardBeacon.builder()
    .name("inspector_id_last4")
    .length(beaconLengthInBits)
    .build();
standardBeaconList.add(inspectorBeacon);

StandardBeacon unitBeacon = StandardBeacon.builder()
    .name("unit")
    .length(beaconLengthInBits)
    .build();
standardBeaconList.add(unitBeacon);

// 2. Define the encrypted parts.
List<EncryptedPart> encryptedPartList = new ArrayList<>);

// Each encrypted part needs a name and prefix
// The name must be the name of the standard beacon
// The prefix must be unique
// For this example we use the prefix "I-" for "inspector_id_last4"
// and "U-" for "unit"
EncryptedPart encryptedPartInspector = EncryptedPart.builder()
    .name("inspector_id_last4")
    .prefix("I-")
    .build();
```

```
encryptedPartList.add(encryptedPartInspector);

EncryptedPart encryptedPartUnit = EncryptedPart.builder()
    .name("unit")
    .prefix("U-")
    .build();
encryptedPartList.add(encryptedPartUnit);

// 3. Create the compound beacon.
// This compound beacon only requires a name, split character,
// and list of encrypted parts
CompoundBeacon inspectorUnitBeacon = CompoundBeacon.builder()
    .name("inspectorUnitBeacon")
    .split(".")
    .sensitive(encryptedPartList)
    .build();
```

C# / .NET

```
// 1. Create standard beacons for the inspector_id_last4 and unit fields.
StandardBeacon inspectorBeacon = new StandardBeacon
 {
   Name = "inspector_id_last4",
   Length = 10
 };
standardBeaconList.Add(inspectorBeacon);
StandardBeacon unitBeacon = new StandardBeacon
 {
    Name = "unit",
    Length = 30
 };
standardBeaconList.Add(unitBeacon);

// 2. Define the encrypted parts.
var last4EncryptedPart = new EncryptedPart

// Each encrypted part needs a name and prefix
// The name must be the name of the standard beacon
// The prefix must be unique
// For this example we use the prefix "I-" for "inspector_id_last4"
// and "U-" for "unit"
var last4EncryptedPart = new EncryptedPart
 {
```

```
    Name = "inspector_id_last4",
    Prefix = "I-"
 };
encryptedPartList.Add(last4EncryptedPart);

var unitEncryptedPart = new EncryptedPart
 {
    Name = "unit",
    Prefix = "U-"
 };
encryptedPartList.Add(unitEncryptedPart);

// 3. Create the compound beacon.
// This compound beacon only requires a name, split character,
// and list of encrypted parts
var compoundBeaconList = new List<CompoundBeacon>>);
var inspectorCompoundBeacon = new CompoundBeacon
  {
      Name = "inspector_id_last4",
      Split = ".",
      Encrypted = encryptedPartList
  };
compoundBeaconList.Add(inspectorCompoundBeacon);
```

Rust

```
// 1. Create standard beacons for the inspector_id_last4 and unit fields.
let last4_beacon = StandardBeacon::builder()
    .name("inspector_id_last4")
    .length(10)
    .build()?;

let unit_beacon = StandardBeacon::builder().name("unit").length(30).build()?;

let standard_beacon_list = vec![last4_beacon, unit_beacon];

// 2. Define the encrypted parts.
// The name must be the name of the standard beacon
// The prefix must be unique
// For this example we use the prefix "I-" for "inspector_id_last4"
// and "U-" for "unit"
let encrypted_parts_list = vec![
    EncryptedPart::builder()
```

```
        .name("inspector_id_last4")
        .prefix("I-")
        .build()?,
    EncryptedPart::builder().name("unit").prefix("U-").build()?,
];

// 3. Create the compound beacon
// This compound beacon only requires a name, split character,
// and list of encrypted parts
let compound_beacon_list = vec![CompoundBeacon::builder()
    .name("last4UnitCompound")
    .split(".")
    .encrypted(encrypted_parts_list)
    .build()?];
```

Menggunakan beacon

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Beacon memungkinkan Anda untuk mencari catatan terenkripsi tanpa mendekripsi seluruh database yang sedang ditanyakan. Beacon dirancang untuk diimplementasikan dalam database baru yang tidak berpenghuni. Setiap suar yang dikonfigurasi dalam database yang ada hanya akan memetakan catatan baru yang ditulis ke database. Beacon dihitung dari nilai plaintext bidang, setelah bidang dienkripsi, tidak ada cara bagi beacon untuk memetakan data yang ada. Setelah Anda menulis catatan baru dengan suar, Anda tidak dapat memperbarui konfigurasi suar. Namun, Anda dapat menambahkan menambahkan beacon baru untuk bidang baru yang Anda tambahkan ke catatan Anda.

Setelah Anda mengkonfigurasi beacon Anda, Anda harus menyelesaikan langkah-langkah berikut sebelum Anda mulai mengisi database Anda dan melakukan query pada beacon Anda.

1. Buat keyring AWS KMS Hierarkis

   Untuk menggunakan enkripsi yang dapat dicari, Anda harus menggunakan keyring AWS KMS Hierarkis untuk menghasilkan, mengenkripsi, dan mendekripsi kunci data yang digunakan untuk melindungi catatan Anda.

Setelah Anda mengkonfigurasi beacon Anda, kumpulkan prasyarat keyring Hierarkis dan buat keyring Hierarkis Anda.

Untuk detail selengkapnya tentang mengapa keyring Hierarkis diperlukan, lihat Menggunakan keyring Hierarkis untuk enkripsi yang dapat dicari.

2.

Tentukan versi beacon

Tentukan `keyStorekeySource`, daftar semua suar standar yang Anda konfigurasikan, daftar semua suar majemuk yang Anda konfigurasikan, daftar bagian terenkripsi, daftar bagian yang ditandatangani, dan versi suar. Anda harus menentukan 1 untuk versi beacon. Untuk panduan tentang mendefinisikan Anda`keySource`, lihatMendefinisikan sumber kunci suar Anda.

Contoh Java berikut mendefinisikan versi beacon untuk database penyewa tunggal. Untuk bantuan mendefinisikan versi beacon untuk database multitenant, lihat Enkripsi yang dapat dicari untuk database multitenant.

Java

```
 List<BeaconVersion> beaconVersions = new ArrayList<>();
beaconVersions.add(
    BeaconVersion.builder()
        .standardBeacons(standardBeaconList)
        .compoundBeacons(compoundBeaconList)
        .encryptedParts(encryptedPartsList)
        .signedParts(signedPartsList)
        .version(1) // MUST be 1
        .keyStore(keyStore)
        .keySource(BeaconKeySource.builder()
            .single(SingleKeyStore.builder()
                .keyId(branchKeyId)
                .cacheTTL(6000)
                .build())
            .build())
        .build()
);
```

C# / .NET

```
var beaconVersions = new List<BeaconVersion>
{
    new BeaconVersion
    {
        StandardBeacons = standardBeaconList,
        CompoundBeacons = compoundBeaconList,
        EncryptedParts = encryptedPartsList,
        SignedParts = signedPartsList,
        Version = 1, // MUST be 1
        KeyStore = branchKeyStoreName,
        KeySource = new BeaconKeySource
        {
            Single = new SingleKeyStore
            {
                KeyId = branch-key-id,
                CacheTTL = 6000
            }
        }
    }
};
```

Rust

```
let beacon_version = BeaconVersion::builder()
    .standard_beacons(standard_beacon_list)
    .compound_beacons(compound_beacon_list)
    .version(1) // MUST be 1
    .key_store(key_store.clone())
    .key_source(BeaconKeySource::Single(
        SingleKeyStore::builder()
            // `keyId` references a beacon key.
            // For every branch key we create in the keystore,
            // we also create a beacon key.
            // This beacon key is not the same as the branch key,
            // but is created with the same ID as the branch key.
            .key_id(branch_key_id)
            .cache_ttl(6000)
            .build()?,
    ))
    .build()?;
```

```
let beacon_versions = vec![beacon_version];
```

3. Konfigurasikan indeks sekunder

   Setelah Anda mengkonfigurasi beacon Anda, Anda harus mengkonfigurasi indeks sekunder yang mencerminkan setiap suar sebelum Anda dapat mencari di bidang terenkripsi. Untuk informasi selengkapnya, lihat Mengkonfigurasi indeks sekunder dengan beacon.

4. Tentukan tindakan kriptografi Anda

   Semua bidang yang digunakan untuk membangun suar standar harus ditandai. `ENCRYPT_AND_SIGN` Semua bidang lain yang digunakan untuk membangun beacon harus ditandai atau. `SIGN_ONLY` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`

5. Konfigurasikan klien SDK Enkripsi AWS Database

   Untuk mengonfigurasi klien SDK Enkripsi AWS Database yang melindungi item tabel di tabel DynamoDB Anda, lihat pustaka enkripsi sisi klien Java untuk DynamoDB.

Meminta suar

Jenis suar yang Anda konfigurasikan menentukan jenis kueri yang dapat Anda lakukan. Beacon standar menggunakan ekspresi filter untuk melakukan pencarian kesetaraan. Compound beacon menggabungkan string plaintext literal dan beacon standar untuk melakukan kueri kompleks. Saat Anda menanyakan data terenkripsi, Anda mencari nama suar.

Anda tidak dapat membandingkan nilai dari dua beacon standar, bahkan jika mereka mengandung plaintext dasar yang sama. Dua beacon standar akan menghasilkan dua tag HMAC yang berbeda untuk nilai plaintext yang sama. Akibatnya, beacon standar tidak dapat melakukan kueri berikut.

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

Compound beacon dapat melakukan query berikut.

- `BEGINS_WITH(`*a*`)`, di mana *a* mencerminkan seluruh nilai bidang tempat suar majemuk rakitan dimulai. Anda tidak dapat menggunakan `BEGINS_WITH` operator untuk mengidentifikasi nilai yang

dimulai dengan substring tertentu. Namun, Anda dapat menggunakanBEGINS_WITH(*S_*), di mana *S_* mencerminkan awalan untuk bagian yang dimulai dengan suar majemuk rakitan.

- CONTAINS(*a*), di mana *a* mencerminkan seluruh nilai bidang yang terkandung dalam suar majemuk rakitan. Anda tidak dapat menggunakan CONTAINS operator untuk mengidentifikasi catatan yang berisi substring tertentu atau nilai dalam satu set.

  Misalnya, Anda tidak dapat melakukan CONTAINS(*path*, "*a*" kueri yang *a* mencerminkan nilai dalam satu set.

- Anda dapat membandingkan bagian yang ditandatangani dari suar majemuk. Saat membandingkan bagian yang ditandatangani, Anda dapat menambahkan awalan bagian terenkripsi secara opsional ke satu atau beberapa bagian yang ditandatangani, tetapi Anda tidak dapat menyertakan nilai bidang terenkripsi dalam kueri apa pun.

  Misalnya, Anda dapat membandingkan bagian yang ditandatangani dan kueri pada *signedField1* = *signedField2* atau*value* IN (*signedField1*, *signedField2*, ...).

  Anda juga dapat membandingkan bagian yang ditandatangani dan awalan dari bagian terenkripsi dengan kueri pada. *signedField1.A_* = *signedField2.B_*

- *field* BETWEEN *a* AND *b*, di mana *a* dan *b* merupakan bagian yang ditandatangani. Anda secara opsional dapat menambahkan awalan dari bagian terenkripsi ke satu atau beberapa bagian yang ditandatangani, tetapi Anda tidak dapat menyertakan nilai bidang terenkripsi dalam kueri apa pun.

Anda harus menyertakan awalan untuk setiap bagian yang Anda sertakan dalam kueri pada suar majemuk. Misalnya, jika Anda membuat suar majemuk,, dari dua bidangcompoundBeacon, encryptedField dansignedField, Anda harus menyertakan awalan yang dikonfigurasi untuk dua bagian tersebut saat Anda menanyakan suar.

```
compoundBeacon = E_encryptedFieldValue.S_signedFieldValue
```

Enkripsi yang dapat dicari untuk database multitenant

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Untuk mengimplementasikan enkripsi yang dapat dicari di database Anda, Anda harus menggunakan keyring AWS KMS Hierarkis. Keyring AWS KMS Hierarkis menghasilkan, mengenkripsi, dan mendekripsi kunci data yang digunakan untuk melindungi catatan Anda. Ini juga menciptakan kunci suar yang digunakan untuk menghasilkan suar. Saat menggunakan keyring AWS KMS Hierarkis dengan database multitenant, ada kunci cabang dan kunci suar yang berbeda untuk setiap penyewa. Untuk menanyakan data terenkripsi dalam database multitenant, Anda harus mengidentifikasi bahan kunci suar yang digunakan untuk menghasilkan suar yang Anda kueri. Untuk informasi selengkapnya, lihat the section called “Menggunakan keyring Hierarkis untuk enkripsi yang dapat dicari”.

Saat Anda menentukan versi beacon untuk database multitenant, tentukan daftar semua beacon standar yang Anda konfigurasikan, daftar semua suar majemuk yang Anda konfigurasikan, versi suar, dan a. `keySource` Anda harus mendefinisikan sumber kunci suar Anda sebagai`MultiKeyStore`, dan menyertakan`keyFieldName`, waktu cache untuk hidup untuk cache kunci suar lokal, dan ukuran cache maksimum untuk cache kunci suar lokal.

Jika Anda mengonfigurasi suar yang ditandatangani, mereka harus disertakan dalam suar Anda. `compoundBeaconList` Signed beacon adalah jenis suar majemuk yang mengindeks dan melakukan kueri kompleks pada dan bidang. `SIGN_ONLY` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`

Java

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

C# / .NET

```
var beaconVersions = new List<BeaconVersion>
{
    new BeaconVersion
    {
        StandardBeacons = standardBeaconList,
        CompoundBeacons = compoundBeaconList,
        EncryptedParts = encryptedPartsList,
        SignedParts = signedPartsList,
        Version = 1, // MUST be 1
        KeyStore = branchKeyStoreName,
        KeySource = new BeaconKeySource
        {
            Multi = new MultiKeyStore
            {
                KeyId = branch-key-id,
                CacheTTL = 6000,
                MaxCacheSize = 10
            }
        }
    }
};
```

Rust

```
let beacon_version = BeaconVersion::builder()
    .standard_beacons(standard_beacon_list)
    .compound_beacons(compound_beacon_list)
    .version(1) // MUST be 1
    .key_store(key_store.clone())
    .key_source(BeaconKeySource::Multi(
        MultiKeyStore::builder()
            // `keyId` references a beacon key.
            // For every branch key we create in the keystore,
            // we also create a beacon key.
            // This beacon key is not the same as the branch key,
            // but is created with the same ID as the branch key.
            .key_id(branch_key_id)
            .cache_ttl(6000)
            .max_cache_size(10)
            .build()?,
    ))
```

```
    .build()?;
let beacon_versions = vec![beacon_version];
```

keyFieldName

`keyFieldName`Mendefinisikan nama bidang yang menyimpan yang `branch-key-id` terkait dengan kunci suar yang digunakan untuk menghasilkan suar untuk penyewa tertentu.

Saat Anda menulis catatan baru ke database Anda, `branch-key-id` yang mengidentifikasi kunci suar yang digunakan untuk menghasilkan suar apa pun untuk catatan itu disimpan di bidang ini.

Secara default, `keyField` adalah bidang konseptual yang tidak secara eksplisit disimpan dalam database Anda. SDK Enkripsi AWS Database mengidentifikasi `branch-key-id` dari kunci data terenkripsi dalam deskripsi material dan menyimpan nilai dalam konseptual `keyField` untuk Anda referensikan di suar majemuk dan suar bertanda tangan. Karena deskripsi materi ditandatangani, konseptual `keyField` dianggap sebagai bagian yang ditandatangani.

Anda juga dapat memasukkan `keyField` dalam tindakan kriptografi Anda sebagai `SIGN_ONLY` atau `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` bidang untuk secara eksplisit menyimpan bidang dalam database Anda. Jika Anda melakukan ini, Anda harus secara manual memasukkan `branch-key-id` dalam `keyField` setiap kali Anda menulis catatan ke database Anda.

Menanyakan beacon dalam database multitenant

Untuk menanyakan suar, Anda harus menyertakan kueri `keyField` dalam kueri Anda untuk mengidentifikasi bahan kunci suar yang sesuai yang diperlukan untuk menghitung ulang suar. Anda harus menentukan yang `branch-key-id` terkait dengan kunci suar yang digunakan untuk menghasilkan suar untuk catatan. Anda tidak dapat menentukan nama ramah yang mengidentifikasi penyewa `branch-key-id` di pemasok ID kunci cabang. Anda dapat memasukkan `keyField` dalam kueri Anda dengan cara berikut.

Suar majemuk

Apakah Anda secara eksplisit menyimpan `keyField` dalam catatan Anda atau tidak, Anda dapat memasukkan `keyField` langsung ke dalam suar majemuk Anda sebagai bagian yang ditandatangani. Bagian yang `keyField` ditandatangani harus diperlukan.

Misalnya, jika Anda ingin membangun suar majemuk,, dari dua bidangcompoundBeacon, encryptedField dansignedField, Anda juga harus menyertakan keyField sebagai bagian yang ditandatangani. Hal ini memungkinkan Anda untuk melakukan query berikut padacompoundBeacon.

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

Suar yang ditandatangani

AWS Database Encryption SDK menggunakan beacon standar dan gabungan untuk menyediakan solusi enkripsi yang dapat dicari. Beacon ini harus menyertakan setidaknya satu bidang terenkripsi. Namun, AWS Database Encryption SDK juga mendukung beacon bertanda tangan yang dapat dikonfigurasi seluruhnya dari SIGN_ONLY plaintext dan field. SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT

Beacon yang ditandatangani dapat dibangun dari satu bagian. Apakah Anda secara eksplisit menyimpan keyField dalam catatan Anda atau tidak, Anda dapat membuat suar yang ditandatangani dari keyField dan menggunakannya untuk membuat kueri gabungan yang menggabungkan kueri pada suar yang keyField ditandatangani dengan kueri di salah satu beacon Anda yang lain. Misalnya, Anda dapat melakukan kueri berikut.

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

Untuk bantuan mengonfigurasi suar bertanda tangan, lihat Membuat suar yang ditandatangani

Query langsung pada **keyField**

Jika Anda menentukan keyField dalam tindakan kriptografi Anda dan secara eksplisit menyimpan bidang dalam catatan Anda, Anda dapat membuat kueri gabungan yang menggabungkan kueri pada suar Anda dengan kueri di file. keyField Anda dapat memilih untuk menanyakan langsung keyField jika Anda ingin menanyakan suar standar. Misalnya, Anda dapat melakukan kueri berikut.

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

AWS SDK Enkripsi Database untuk DynamoDB

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

SDK Enkripsi AWS Database untuk DynamoDB adalah pustaka perangkat lunak yang memungkinkan Anda menyertakan enkripsi sisi klien dalam desain Amazon DynamoDB Anda. SDK Enkripsi AWS Database untuk DynamoDB menyediakan enkripsi tingkat atribut dan memungkinkan Anda menentukan item mana yang akan dienkripsi dan item mana yang akan disertakan dalam tanda tangan yang memastikan keaslian data Anda. Mengenkripsi data sensitif Anda saat transit dan diam membantu memastikan bahwa data teks biasa Anda tidak tersedia untuk pihak ketiga mana pun, termasuk. AWS

> **Note**
>
> SDK Enkripsi AWS Database tidak mendukung PartiQL.

Dalam DynamoDB, tabel adalah koleksi item. Setiap item adalah koleksi atribut. Tiap atribut memiliki nama dan nilai. SDK Enkripsi AWS Database untuk DynamoDB mengenkripsi nilai atribut. Kemudian, DynamoDB Encryption Client menghitung tanda tangan atas atribut. Anda menentukan nilai atribut mana yang akan dienkripsi dan mana yang akan disertakan dalam tanda tangan dalam tindakan kriptografi.

Topik dalam Bab ini memberikan gambaran umum tentang SDK Enkripsi AWS Database untuk DynamoDB, termasuk bidang mana yang dienkripsi, panduan tentang instalasi dan konfigurasi klien, dan contoh Java untuk membantu Anda memulai.

Topik

- Enkripsi di sisi klien dan sisi server
- Bidang mana yang dienkripsi dan ditandatangani?
- Enkripsi yang dapat dicari di DynamoDB
- Memperbarui model data Anda
- AWS SDK Enkripsi Database untuk DynamoDB bahasa pemrograman yang tersedia
- Klien Enkripsi DynamoDB Legacy

Enkripsi di sisi klien dan sisi server

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

SDK Enkripsi AWS Database untuk DynamoDB mendukung enkripsi sisi klien, tempat Anda mengenkripsi data tabel sebelum mengirimkannya ke database. Namun, DynamoDB menyediakan fitur enkripsi saat istirahat di sisi server yang secara transparan mengenkripsi tabel Anda ketika ditahan ke disk dan mendekripsinya ketika Anda mengakses tabel.

Alat-alat yang Anda pilih bergantung pada sensitivitas data Anda dan persyaratan keamanan aplikasi Anda. Anda dapat menggunakan SDK Enkripsi AWS Database untuk DynamoDB dan enkripsi saat istirahat. Ketika Anda mengirim item yang dienkripsi dan ditandatangani ke DynamoDB, DynamoDB tidak mengenali item sebagai dilindungi. DynamoDB hanya mendeteksi item tabel khas dengan nilai-nilai atribut biner.

Enkripsi sisi server saat istirahat

DynamoDB mendukung enkripsi saat istirahat, sebuah fitur enkripsi di sisi server yang mana DynamoDB secara transparan mengenkripsi tabel Anda untuk Anda ketika tabel bertahan untuk disk, dan mendekripsinya ketika Anda mengakses data tabel.

Saat Anda menggunakan AWS SDK untuk berinteraksi dengan DynamoDB, secara default, data Anda dienkripsi saat transit melalui koneksi HTTPS, didekripsi di titik akhir DynamoDB, dan kemudian dienkripsi ulang sebelum disimpan di DynamoDB.

- Enkripsi secara default. DynamoDB secara transparan mengenkripsi dan mendekripsi semua tabel saat ditulis. Tidak ada pilihan untuk mengaktifkan atau menonaktifkan enkripsi saat istirahat.
- DynamoDB membuat dan mengelola kunci kriptografi.Kunci unik untuk setiap tabel dilindungi oleh AWS KMS keyyang tidak pernah meninggalkan AWS Key Management Service(AWS KMS) tidak terenkripsi. Secara default, DynamoDB menggunakan Kunci milik AWSdi akun layanan DynamoDB, tetapi Anda dapat memilih atau kunci Kunci yang dikelola AWSyang dikelola pelanggan di akun Anda untuk melindungi sebagian atau semua tabel Anda.
- Semua data tabel dienkripsi pada disk.Ketika tabel yang dienkripsi disimpan ke disk, DynamoDB mengenkripsi semua data tabel, termasuk kunci primer serta indeks sekunder lokal dan global. Jika tabel Anda memiliki kunci sortir, beberapa kunci sortir yang menandai batas kisaran disimpan dalam plaintext dalam metadata tabel.

- Objek yang terkait dengan tabel juga dienkripsi. Enkripsi saat istirahat melindungi DynamoDB Streams, tabel global, dan cadangan setiap kali ditulis untuk media tahan lama.
- Item Anda didekripsi saat Anda mengaksesnya.Ketika Anda mengakses tabel, DynamoDB mendekripsi bagian dari tabel yang mencakup item target Anda, dan mengembalikan item plaintext untuk Anda.

AWS SDK Enkripsi Database untuk DynamoDB

Enkripsi sisi klien memberikan end-to-end perlindungan untuk data Anda, dalam perjalanan dan saat istirahat, dari sumbernya ke penyimpanan di DynamoDB. Data plaintext Anda tidak pernah diekspos ke pihak ketiga mana pun, termasuk. AWS Anda dapat menggunakan SDK Enkripsi AWS Database untuk DynamoDB dengan tabel DynamoDB baru, atau Anda dapat memigrasikan tabel Amazon DynamoDB yang ada ke versi terbaru SDK Enkripsi Database untuk DynamoDB. AWS

- Data Anda dilindungi saat transit dan saat istirahat. Itu tidak pernah diekspos ke pihak ketiga mana pun, termasuk AWS.
- Anda dapat menandatangani Item tabel Anda. Anda dapat mengarahkan SDK Enkripsi AWS Database untuk DynamoDB untuk menghitung tanda tangan atas semua atau sebagian item tabel, termasuk atribut kunci utama. Tanda tangan ini memungkinkan Anda untuk mendeteksi perubahan yang tidak sah pada item secara keseluruhan, termasuk menambahkan atau menghapus atribut, atau menukar nilai atribut.
- Anda menentukan bagaimana data Anda dilindungi dengan memilih keyring. Keyring Anda menentukan kunci pembungkus yang melindungi kunci data Anda, dan akhirnya, data Anda. Gunakan kunci pembungkus paling aman yang praktis untuk tugas Anda.
- SDK Enkripsi AWS Database untuk DynamoDB tidak mengenkripsi seluruh tabel. Anda memilih atribut mana yang dienkripsi dalam item Anda. SDK Enkripsi AWS Database untuk DynamoDB tidak mengenkripsi seluruh item. Ini tidak mengenkripsi nama atribut, atau nama atau nilai atribut kunci utama (kunci partisi dan kunci sortir).

AWS Encryption SDK

Jika Anda mengenkripsi data yang Anda simpan di DynamoDB, kami merekomendasikan SDK Enkripsi Database AWS untuk DynamoDB.

AWS Encryption SDK adalah pustaka enkripsi di sisi klien yang membantu Anda untuk mengenkripsi dan mendekripsi data generik. Meskipun dapat melindungi semua jenis data, ia tidak dirancang untuk bekerja dengan data terstruktur, seperti catatan basis data. Tidak seperti SDK Enkripsi AWS

Database untuk DynamoDB, AWS Encryption SDK tidak dapat memberikan pemeriksaan integritas tingkat item dan tidak memiliki logika untuk mengenali atribut atau mencegah enkripsi kunci utama.

Jika Anda menggunakan AWS Encryption SDK untuk mengenkripsi elemen apa pun dari tabel Anda, ingatlah bahwa itu tidak kompatibel dengan SDK Enkripsi AWS Database untuk DynamoDB. Anda tidak dapat mengenkripsi dengan satu pustaka dan mendekripsinya dengan pustaka yang lain.

Bidang mana yang dienkripsi dan ditandatangani?

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

SDK Enkripsi AWS Database untuk DynamoDB adalah pustaka enkripsi sisi klien yang dirancang khusus untuk aplikasi Amazon DynamoDB. Amazon DynamoDB menyimpan data dalam tabel, yang merupakan kumpulan item. Setiap item adalah koleksi atribut. Tiap atribut memiliki nama dan nilai. SDK Enkripsi AWS Database untuk DynamoDB mengenkripsi nilai atribut. Kemudian, DynamoDB Encryption Client menghitung tanda tangan atas atribut. Anda dapat menentukan nilai atribut untuk dienkripsi dan yang disertakan dalam tanda tangan.

Enkripsi melindungi kerahasiaan nilai atribut. Penandatanganan menyediakan integritas semua atribut yang ditandatangani dan hubungan satu sama lain, dan menyediakan autentikasi. Hal ini memungkinkan Anda untuk mendeteksi perubahan yang tidak sah pada item secara keseluruhan, termasuk menambahkan atau menghapus atribut, atau mengganti satu nilai terenkripsi dengan yang lain.

Dalam item terenkripsi, beberapa data tetap dalam teks biasa, termasuk nama tabel, semua nama atribut, nilai atribut yang tidak Anda enkripsi, nama dan nilai atribut kunci primer (kunci partisi dan kunci sortir), dan jenis atribut. Jangan menyimpan data sensitif dalam bidang ini.

Untuk informasi selengkapnya tentang cara kerja SDK Enkripsi AWS Database untuk DynamoDB, lihat. Cara kerja SDK Enkripsi AWS Database

> **Note**
>
> Semua penyebutan tindakan atribut dalam AWS Database Encryption SDK untuk topik DynamoDB mengacu pada tindakan kriptografi.

Topik

- Enkripsi nilai atribut
- Penandatanganan item

Enkripsi nilai atribut

SDK Enkripsi AWS Database untuk DynamoDB mengenkripsi nilai (tetapi bukan nama atau jenis atribut) atribut yang Anda tentukan. Untuk menentukan nilai atribut yang dienkripsi, gunakan tindakan atribut.

Sebagai contoh, item ini termasuk atribut `example` dan `test`.

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

Jika Anda mengenkripsi atribut `example`, tetapi tidak mengenkripsi atribut `test`, hasilnya terlihat seperti berikut ini. Nilai atribut `example` yang dienkripsi adalah data biner, bukan string.

```
'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'
...
```

Atribut kunci primer - kunci partisi dan kunci sortir-dari setiap item harus tetap dalam plaintext karena DynamoDB menggunakannya untuk menemukan item dalam tabel. Atribut itu harus ditandatangani, tapi tidak dienkripsi.

SDK Enkripsi AWS Database untuk DynamoDB mengidentifikasi atribut kunci utama untuk Anda dan memastikan bahwa nilainya ditandatangani, tetapi tidak dienkripsi. Dan, jika Anda mengidentifikasi kunci utama Anda dan kemudian mencoba untuk mengenkripsinya, klien akan melemparkan pengecualian.

Klien menyimpan deskripsi materi dalam atribut baru (`aws_dbe_head`) yang ditambahkan ke item. Deskripsi materi menjelaskan bagaimana item dienkripsi dan ditandatangani. Klien menggunakan informasi ini untuk memverifikasi dan mendekripsi item. Bidang yang menyimpan deskripsi materi tidak dienkripsi.

Penandatanganan item

Setelah mengenkripsi nilai atribut yang ditentukan, SDK Enkripsi AWS Database untuk DynamoDB menghitung Kode Otentikasi Pesan Berbasis Hash (HMACs) dan tanda tangan digital atas kanonikalisasi deskripsi materi, konteks enkripsi, dan setiap bidang yang ditandai, atau dalam tindakan atribut. `ENCRYPT_AND_SIGNSIGN_ONLYSIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Tanda tangan ECDSA diaktifkan secara default, tetapi tidak diperlukan. Klien menyimpan HMACs dan tanda tangan dalam atribut baru (`aws_dbe_foot`) yang ditambahkan ke item.

Enkripsi yang dapat dicari di DynamoDB

Untuk mengonfigurasi tabel Amazon DynamoDB Anda untuk enkripsi yang dapat dicari, Anda harus menggunakan AWS KMS keyring Hierarkis untuk menghasilkan, mengenkripsi, dan mendekripsi kunci data yang digunakan untuk melindungi item Anda. Anda juga harus menyertakan konfigurasi enkripsi tabel Anda. `SearchConfig`

> **Note**
>
> Jika Anda menggunakan pustaka enkripsi sisi klien Java untuk DynamoDB, Anda harus menggunakan SDK Enkripsi AWS Database tingkat rendah untuk DynamoDB API untuk mengenkripsi, menandatangani, memverifikasi, dan mendekripsi item tabel Anda. DynamoDB Enhanced Client dan `DynamoDBItemEncryptor` level yang lebih rendah tidak mendukung enkripsi yang dapat dicari.

Topik

- Mengkonfigurasi indeks sekunder dengan beacon
- Menguji output suar

Mengkonfigurasi indeks sekunder dengan beacon

Setelah Anda mengkonfigurasi beacon Anda, Anda harus mengkonfigurasi indeks sekunder yang mencerminkan setiap suar sebelum Anda dapat mencari pada atribut terenkripsi.

Saat Anda mengonfigurasi suar standar atau gabungan, SDK Enkripsi AWS Database menambahkan `aws_dbe_b_` awalan ke nama suar sehingga server dapat dengan mudah mengidentifikasi beacon.

Misalnya, jika Anda menamai suar majemukcompoundBeacon, nama suar lengkapnya sebenarnya. aws_dbe_b_compoundBeacon Jika Anda ingin mengonfigurasi indeks sekunder yang menyertakan suar standar atau majemuk, Anda harus menyertakan aws_dbe_b_ awalan saat mengidentifikasi nama suar.

Partisi dan sortir kunci

Anda tidak dapat mengenkripsi nilai kunci primer. Kunci partisi dan sortir Anda harus ditandatangani. Nilai kunci primer Anda tidak bisa menjadi suar standar atau majemuk.

Nilai kunci utama Anda harusSIGN_ONLY, kecuali Anda menentukan SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT atribut apa pun, maka atribut partisi dan sortir juga harusSIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT.

Nilai kunci utama Anda dapat ditandatangani beacon. Jika Anda mengonfigurasi beacon bertanda tangan yang berbeda untuk setiap nilai kunci utama, Anda harus menentukan nama atribut yang mengidentifikasi nilai kunci primer sebagai nama suar yang ditandatangani. Namun, SDK Enkripsi AWS Database tidak menambahkan aws_dbe_b_ awalan ke beacon yang ditandatangani. Bahkan jika Anda mengonfigurasi beacon bertanda tangan yang berbeda untuk nilai kunci utama Anda, Anda hanya perlu menentukan nama atribut untuk nilai kunci primer saat Anda mengonfigurasi indeks sekunder.

Indeks sekunder lokal

Kunci sortir untuk indeks sekunder lokal dapat berupa suar.

Jika Anda menentukan suar untuk kunci sortir, tipenya harus String. Jika Anda menentukan standar atau suar majemuk untuk kunci sortir, Anda harus menyertakan aws_dbe_b_ awalan saat Anda menentukan nama suar. Jika Anda menentukan suar yang ditandatangani, tentukan nama suar tanpa awalan apa pun.

Indeks sekunder global

Kunci partisi dan sortir untuk indeks sekunder global dapat berupa beacon.

Jika Anda menentukan suar untuk partisi atau kunci sortir, jenisnya harus String. Jika Anda menentukan standar atau suar majemuk untuk kunci sortir, Anda harus menyertakan aws_dbe_b_ awalan saat Anda menentukan nama suar. Jika Anda menentukan suar yang ditandatangani, tentukan nama suar tanpa awalan apa pun.

Proyeksi atribut

Proyeksi adalah kumpulan atribut yang disalin dari tabel ke indeks sekunder. Kunci partisi dan kunci urutan tabel selalu diproyeksikan ke dalam indeks; Anda dapat memproyeksikan atribut lain untuk mendukung persyaratan kueri aplikasi Anda. DynamoDB menyediakan tiga opsi berbeda untuk proyeksi `KEYS_ONLY` atribut:,, dan. `INCLUDE ALL`

Jika Anda menggunakan proyeksi atribut INCLUDE untuk mencari pada suar, Anda harus menentukan nama untuk semua atribut yang suar dibangun dari dan nama suar dengan awalan. `aws_dbe_b_` Misalnya, jika Anda mengkonfigurasi suar majemuk,`compoundBeacon`, dari,`field1`, dan `field2field3`, Anda harus menentukan`aws_dbe_b_compoundBeacon`,, `field1field2`, dan `field3` dalam proyeksi.

Indeks sekunder global hanya dapat menggunakan atribut yang ditentukan secara eksplisit dalam proyeksi, tetapi indeks sekunder lokal dapat menggunakan atribut apa pun.

Menguji output suar

Jika Anda mengonfigurasi suar majemuk atau membuat beacon menggunakan bidang virtual, sebaiknya verifikasi bahwa beacon ini menghasilkan output yang diharapkan sebelum mengisi tabel DynamoDB Anda.

AWS Database Encryption SDK menyediakan `DynamoDbEncryptionTransforms` layanan untuk membantu Anda memecahkan masalah bidang virtual dan output suar gabungan.

Menguji bidang virtual

Cuplikan berikut membuat item pengujian, mendefinisikan `DynamoDbEncryptionTransforms` layanan dengan konfigurasi enkripsi tabel DynamoDB, dan menunjukkan cara menggunakan `ResolveAttributes` untuk memverifikasi bahwa bidang virtual menghasilkan output yang diharapkan.

Java

Lihat contoh kode lengkapnya: VirtualBeaconSearchableEncryptionExample.java

```
// Create test items
final PutItemRequest itemWithHasTestResultPutRequest = PutItemRequest.builder()
    .tableName(ddbTableName)
```

```
    .item(itemWithHasTestResult)
    .build();

final PutItemResponse itemWithHasTestResultPutResponse =
 ddb.putItem(itemWithHasTestResultPutRequest);

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

final PutItemResponse itemWithNoHasTestResultPutResponse =
 ddb.putItem(itemWithNoHasTestResultPutRequest);

// Define the DynamoDbEncryptionTransforms service
final DynamoDbEncryptionTransforms trans = DynamoDbEncryptionTransforms.builder()
    .DynamoDbTablesEncryptionConfig(encryptionConfig).build();

// Verify configuration
final ResolveAttributesInput resolveInput = ResolveAttributesInput.builder()
    .TableName(ddbTableName)
    .Item(itemWithHasTestResult)
    .Version(1)
    .build();
final ResolveAttributesOutput resolveOutput = trans.ResolveAttributes(resolveInput);

// Verify that VirtualFields has the expected value
Map<String, String> vf = new HashMap<>();
vf.put("stateAndHasTestResult", "CAt");
assert resolveOutput.VirtualFields().equals(vf);
```

C# / .NET

Lihat contoh kode lengkapnya: VirtualBeaconSearchableEncryptionExample.cs.

```
 // Create item with hasTestResult=true
var itemWithHasTestResult = new Dictionary<String, AttributeValue>
{
    ["customer_id"] = new AttributeValue("ABC-123"),
    ["create_time"] = new AttributeValue { N = "1681495205" },
    ["state"] = new AttributeValue("CA"),
    ["hasTestResult"] = new AttributeValue { BOOL = true }
};
```

```
// Create item with hasTestResult=false
var itemWithNoHasTestResult = new Dictionary<String, AttributeValue>
{
    ["customer_id"] = new AttributeValue("DEF-456"),
    ["create_time"] = new AttributeValue { N = "1681495205" },
    ["state"] = new AttributeValue("CA"),
    ["hasTestResult"] = new AttributeValue { BOOL = false }
};

// Define the DynamoDbEncryptionTransforms service
var trans = new DynamoDbEncryptionTransforms(encryptionConfig);

// Verify configuration
var resolveInput = new ResolveAttributesInput
{
    TableName = ddbTableName,
    Item = itemWithHasTestResult,
    Version = 1
};
var resolveOutput = trans.ResolveAttributes(resolveInput);

// Verify that VirtualFields has the expected value
Debug.Assert(resolveOutput.VirtualFields.Count == 1);
Debug.Assert(resolveOutput.VirtualFields["stateAndHasTestResult"] == "CAt");
```

Rust

Lihat contoh kode lengkap: virtual_beacon_searchable_encryption.rs.

```
// Create item with hasTestResult=true
let item_with_has_test_result = HashMap::from([
    (
        "customer_id".to_string(),
        AttributeValue::S("ABC-123".to_string()),
    ),
    (
        "create_time".to_string(),
        AttributeValue::N("1681495205".to_string()),
    ),
    ("state".to_string(), AttributeValue::S("CA".to_string())),
    ("hasTestResult".to_string(), AttributeValue::Bool(true)),
]);

// Create item with hasTestResult=false
```

```
let item_with_no_has_test_result = HashMap::from([
    (
        "customer_id".to_string(),
        AttributeValue::S("DEF-456".to_string()),
    ),
    (
        "create_time".to_string(),
        AttributeValue::N("1681495205".to_string()),
    ),
    ("state".to_string(), AttributeValue::S("CA".to_string())),
    ("hasTestResult".to_string(), AttributeValue::Bool(false)),
]);

// Define the transform service
let trans = transform_client::Client::from_conf(encryption_config.clone())?;

// Verify the configuration
let resolve_output = trans
    .resolve_attributes()
    .table_name(ddb_table_name)
    .item(item_with_has_test_result.clone())
    .version(1)
    .send()
    .await?;

// Verify that VirtualFields has the expected value
let virtual_fields = resolve_output.virtual_fields.unwrap();
assert_eq!(virtual_fields.len(), 1);
assert_eq!(virtual_fields["stateAndHasTestResult"], "CAt");
```

Menguji suar majemuk

Cuplikan berikut membuat item uji, mendefinisikan `DynamoDbEncryptionTransforms` layanan dengan konfigurasi enkripsi tabel DynamoDB, dan menunjukkan cara menggunakan untuk memverifikasi bahwa suar majemuk menghasilkan output yang `ResolveAttributes` diharapkan.

Java

Lihat contoh kode lengkapnya: CompoundBeaconSearchableEncryptionExample.java

```
// Create an item with both attributes used in the compound beacon.
final HashMap<String, AttributeValue> item = new HashMap<>();
```

```
item.put("work_id", AttributeValue.builder().s("9ce39272-8068-4efd-a211-
cd162ad65d4c").build());
item.put("inspection_date", AttributeValue.builder().s("2023-06-13").build());
item.put("inspector_id_last4", AttributeValue.builder().s("5678").build());
item.put("unit", AttributeValue.builder().s("011899988199").build());

// Define the DynamoDbEncryptionTransforms service
final DynamoDbEncryptionTransforms trans = DynamoDbEncryptionTransforms.builder()
    .DynamoDbTablesEncryptionConfig(encryptionConfig).build();

// Verify configuration
final ResolveAttributesInput resolveInput = ResolveAttributesInput.builder()
    .TableName(ddbTableName)
    .Item(item)
    .Version(1)
    .build();

final ResolveAttributesOutput resolveOutput = trans.ResolveAttributes(resolveInput);

// Verify that CompoundBeacons has the expected value
Map<String, String> cbs = new HashMap<>();
cbs.put("last4UnitCompound", "L-5678.U-011899988199");
assert resolveOutput.CompoundBeacons().equals(cbs);
// Note : the compound beacon actually stored in the table is not
 "L-5678.U-011899988199"
// but rather something like "L-abc.U-123", as both parts are EncryptedParts
// and therefore the text is replaced by the associated beacon
```

C# / .NET

Lihat contoh kode lengkapnya: CompoundBeaconSearchableEncryptionExample.cs

```
// Create an item with both attributes used in the compound beacon
var item = new Dictionary<String, AttributeValue>
{
    ["work_id"] = new AttributeValue("9ce39272-8068-4efd-a211-cd162ad65d4c"),
    ["inspection_date"] = new AttributeValue("2023-06-13"),
    ["inspector_id_last4"] = new AttributeValue("5678"),
    ["unit"] = new AttributeValue("011899988199")
};

// Define the DynamoDbEncryptionTransforms service
var trans = new DynamoDbEncryptionTransforms(encryptionConfig);
```

```
// Verify configuration
var resolveInput = new ResolveAttributesInput
{
    TableName = ddbTableName,
    Item = item,
    Version = 1
};
var resolveOutput = trans.ResolveAttributes(resolveInput);


// Verify that CompoundBeacons has the expected value
Debug.Assert(resolveOutput.CompoundBeacons.Count == 1);
Debug.Assert(resolveOutput.CompoundBeacons["last4UnitCompound"] ==
 "L-5678.U-011899988199");
// Note : the compound beacon actually stored in the table is not
 "L-5678.U-011899988199"
// but rather something like "L-abc.U-123", as both parts are EncryptedParts
// and therefore the text is replaced by the associated beacon
```

Rust

Lihat contoh kode lengkap: compound_beacon_searchable_encryption.rs

```
// Create an item with both attributes used in the compound beacon
let item = HashMap::from([
    (
        "work_id".to_string(),
        AttributeValue::S("9ce39272-8068-4efd-a211-cd162ad65d4c".to_string()),
    ),
    (
        "inspection_date".to_string(),
        AttributeValue::S("2023-06-13".to_string()),
    ),
    (
        "inspector_id_last4".to_string(),
        AttributeValue::S("5678".to_string()),
    ),
    (
        "unit".to_string(),
        AttributeValue::S("011899988199".to_string()),
    ),
]);

// Define the transforms service
```

```
let trans = transform_client::Client::from_conf(encryption_config.clone())?;

// Verify configuration
let resolve_output = trans
    .resolve_attributes()
    .table_name(ddb_table_name)
    .item(item.clone())
    .version(1)
    .send()
    .await?;

// Verify that CompoundBeacons has the expected value
Dlet compound_beacons = resolve_output.compound_beacons.unwrap();
assert_eq!(compound_beacons.len(), 1);
assert_eq!(
    compound_beacons["last4UnitCompound"],
    "L-5678.U-011899988199"
);
// but rather something like "L-abc.U-123", as both parts are EncryptedParts
// and therefore the text is replaced by the associated beacon
```

Memperbarui model data Anda

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Saat mengonfigurasi SDK Enkripsi AWS Database untuk DynamoDB, Anda memberikan tindakan atribut. Pada enkripsi, AWS Database Encryption SDK menggunakan tindakan atribut untuk mengidentifikasi atribut mana yang akan dienkripsi dan ditandatangani, atribut mana yang akan ditandatangani (tetapi tidak mengenkripsi), dan mana yang harus diabaikan. Anda juga menentukan atribut unsigned yang diizinkan untuk secara eksplisit memberi tahu klien atribut mana yang dikecualikan dari tanda tangan. Saat mendekripsi, SDK Enkripsi AWS Database menggunakan atribut unsigned yang diizinkan yang Anda tetapkan untuk mengidentifikasi atribut mana yang tidak disertakan dalam tanda tangan. Tindakan atribut tidak disimpan dalam item terenkripsi dan SDK Enkripsi AWS Database tidak memperbarui tindakan atribut Anda secara otomatis.

Pilih tindakan atribut Anda dengan hati-hati. Bila ragu, gunakan Enkripsi dan tanda tangan. Setelah Anda menggunakan AWS Database Encryption SDK untuk melindungi item Anda, Anda

tidak dapat mengubah SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT atribut yang ada ENCRYPT_AND_SIGNSIGN_ONLY, atau keDO_NOTHING. Namun, Anda dapat dengan aman melakukan perubahan berikut.

- Tambahkan SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT atribut baru ENCRYPT_AND_SIGNSIGN_ONLY, dan
- Hapus atribut yang ada
- Ubah ENCRYPT_AND_SIGN atribut yang ada ke SIGN_ONLY atau SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT
- Ubah SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT atribut SIGN_ONLY atau yang sudah ada ke ENCRYPT_AND_SIGN
- Tambahkan DO_NOTHING atribut baru
- Ubah SIGN_ONLY atribut yang ada menjadi SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT
- Ubah SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT atribut yang ada menjadi SIGN_ONLY

Pertimbangan untuk enkripsi yang dapat dicari

Sebelum Anda memperbarui model data Anda, pertimbangkan dengan cermat bagaimana pembaruan Anda dapat memengaruhi beacon apa pun yang Anda buat dari atribut. Setelah Anda menulis catatan baru dengan suar, Anda tidak dapat memperbarui konfigurasi suar. Anda tidak dapat memperbarui tindakan atribut yang terkait dengan atribut yang Anda gunakan untuk membangun beacon. Jika Anda menghapus atribut yang ada dan suar terkait, Anda tidak akan dapat menanyakan catatan yang ada menggunakan suar itu. Anda dapat membuat beacon baru untuk bidang baru yang ditambahkan ke rekaman, tetapi Anda tidak dapat memperbarui beacon yang ada untuk menyertakan bidang baru.

Pertimbangan untuk atribut **SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT**

Secara default, kunci partisi dan sortir adalah satu-satunya atribut yang disertakan dalam konteks enkripsi. Anda dapat mempertimbangkan untuk mendefinisikan bidang tambahan SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT agar pemasok ID kunci cabang untuk keyring AWS KMS Hierarkis Anda dapat mengidentifikasi kunci cabang mana yang diperlukan untuk dekripsi dari konteks enkripsi. Untuk informasi selengkapnya, lihat pemasok ID kunci cabang. Jika Anda menentukan SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT atribut apa pun, maka atribut partisi dan sortir juga harusSIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT.

> **Note**
>
> Untuk menggunakan tindakan `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` kriptografi, Anda harus menggunakan SDK Enkripsi AWS Database versi 3.3 atau yang lebih baru. Terapkan versi baru ke semua pembaca sebelum memperbarui model data Anda untuk disertakan`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

Tambahkan **`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`** atribut baru **`ENCRYPT_AND_SIGNSIGN_ONLY`**, dan

Untuk menambahkan atribut baru `ENCRYPT_AND_SIGNSIGN_ONLY`, atau `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` atribut, tentukan atribut baru dalam tindakan atribut Anda.

Anda tidak dapat menghapus `DO_NOTHING` atribut yang ada dan menambahkannya kembali sebagai`ENCRYPT_AND_SIGN`,`SIGN_ONLY`, atau `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` atribut.

Menggunakan kelas data beranotasi

Jika Anda mendefinisikan tindakan atribut Anda dengan a`TableSchema`, tambahkan atribut baru ke kelas data beranotasi Anda. Jika Anda tidak menentukan anotasi tindakan atribut untuk atribut baru, klien akan mengenkripsi dan menandatangani atribut baru secara default (kecuali atribut adalah bagian dari kunci utama). Jika Anda hanya ingin menandatangani atribut baru, Anda harus menambahkan atribut baru dengan `@DynamoDBEncryptionSignOnly` atau `@DynamoDBEncryptionSignAndIncludeInEncryptionContext` anotasi.

Menggunakan model objek

Jika Anda secara manual mendefinisikan tindakan atribut Anda, tambahkan atribut baru ke tindakan atribut dalam model objek Anda dan tentukan `ENCRYPT_AND_SIGNSIGN_ONLY`,, atau `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` sebagai tindakan atribut.

Hapus atribut yang ada

Jika Anda memutuskan bahwa Anda tidak lagi memerlukan atribut, Anda dapat berhenti menulis data ke atribut tersebut atau Anda dapat secara resmi menghapusnya dari tindakan atribut Anda. Ketika Anda berhenti menulis data baru ke atribut, atribut masih muncul dalam tindakan atribut Anda. Ini

dapat membantu jika Anda perlu mulai menggunakan atribut lagi di masa mendatang. Menghapus atribut secara formal dari tindakan atribut Anda tidak menghapusnya dari kumpulan data Anda. Dataset Anda akan tetap berisi item yang menyertakan atribut itu.

Untuk menghapus `DO_NOTHING` atribut`ENCRYPT_AND_SIGN`,, `SIGN_ONLY` atau yang sudah ada secara resmi, perbarui tindakan atribut Anda. `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`

Jika Anda menghapus `DO_NOTHING` atribut, Anda tidak boleh menghapus atribut tersebut dari atribut yang tidak ditandatangani yang diizinkan. Bahkan jika Anda tidak lagi menulis nilai baru ke atribut itu, klien masih perlu tahu bahwa atribut tersebut tidak ditandatangani untuk membaca item yang ada yang berisi atribut.

Menggunakan kelas data beranotasi

Jika Anda mendefinisikan tindakan atribut Anda dengan `aTableSchema`, hapus atribut dari kelas data beranotasi Anda.

Menggunakan model objek

Jika Anda secara manual mendefinisikan tindakan atribut Anda, hapus atribut dari tindakan atribut dalam model objek Anda.

Ubah **`ENCRYPT_AND_SIGN`** atribut yang ada ke **`SIGN_ONLY`** atau **`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`**

Untuk mengubah `ENCRYPT_AND_SIGN` atribut yang ada ke `SIGN_ONLY` atau`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, Anda harus memperbarui tindakan atribut Anda. Setelah Anda menerapkan pembaruan, klien akan dapat memverifikasi dan mendekripsi nilai yang ada yang ditulis ke atribut, tetapi hanya akan menandatangani nilai baru yang ditulis ke atribut.

> **Note**
>
> Pertimbangkan persyaratan keamanan Anda dengan cermat sebelum mengubah `ENCRYPT_AND_SIGN` atribut yang ada ke `SIGN_ONLY` atau`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Atribut apa pun yang dapat menyimpan data sensitif harus dienkripsi.

Menggunakan kelas data beranotasi

Jika Anda mendefinisikan tindakan atribut dengan `aTableSchema`, perbarui atribut yang ada untuk menyertakan `@DynamoDBEncryptionSignOnly` atau `@DynamoDBEncryptionSignAndIncludeInEncryptionContext` anotasi dalam kelas data beranotasi Anda.

Menggunakan model objek

Jika Anda menentukan tindakan atribut secara manual, perbarui tindakan atribut yang terkait dengan atribut yang ada dari `ENCRYPT_AND_SIGN` ke `SIGN_ONLY` atau `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` dalam model objek Anda.

Ubah **`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`** atribut **`SIGN_ONLY`** atau yang sudah ada ke **`ENCRYPT_AND_SIGN`**

Untuk mengubah atribut `SIGN_ONLY` atau `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` atribut yang ada`ENCRYPT_AND_SIGN`, Anda harus memperbarui tindakan atribut Anda. Setelah Anda menyebarkan pembaruan, klien akan dapat memverifikasi nilai yang ada yang ditulis ke atribut, dan akan mengenkripsi dan menandatangani nilai baru yang ditulis ke atribut.

Menggunakan kelas data beranotasi

Jika Anda menentukan tindakan atribut Anda dengan `aTableSchema`, hapus `@DynamoDBEncryptionSignOnly` atau `@DynamoDBEncryptionSignAndIncludeInEncryptionContext` anotasi dari atribut yang ada.

Menggunakan model objek

Jika Anda menentukan tindakan atribut secara manual, perbarui tindakan atribut yang terkait dengan atribut dari `SIGN_ONLY` atau `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` ke `ENCRYPT_AND_SIGN` dalam model objek Anda.

Tambahkan **`DO_NOTHING`** atribut baru

Untuk mengurangi risiko kesalahan saat menambahkan `DO_NOTHING` atribut baru, sebaiknya tentukan awalan yang berbeda saat menamai `DO_NOTHING` atribut Anda, lalu gunakan awalan tersebut untuk menentukan atribut unsigned yang diizinkan.

Anda tidak dapat menghapus `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` atribut yang ada `ENCRYPT_AND_SIGNSIGN_ONLY`,, atau dari kelas data beranotasi dan kemudian menambahkan

atribut kembali sebagai DO_NOTHING atribut. Anda hanya dapat menambahkan DO_NOTHING atribut yang sama sekali baru.

Langkah-langkah yang Anda ambil untuk menambahkan DO_NOTHING atribut baru bergantung pada apakah Anda menetapkan atribut unsigned yang diizinkan secara eksplisit dalam daftar atau dengan awalan.

Menggunakan awalan atribut unsigned yang diizinkan

Jika Anda mendefinisikan tindakan atribut Anda dengan aTableSchema, tambahkan DO_NOTHING atribut baru ke kelas data beranotasi Anda dengan anotasi@DynamoDBEncryptionDoNothing. Jika Anda menentukan tindakan atribut secara manual, perbarui tindakan atribut Anda untuk menyertakan atribut baru. Pastikan untuk secara eksplisit mengkonfigurasi atribut baru dengan tindakan DO_NOTHING atribut. Anda harus menyertakan awalan berbeda yang sama dalam nama atribut baru.

Menggunakan daftar atribut unsigned yang diizinkan

1. Tambahkan DO_NOTHING atribut baru ke daftar atribut unsigned yang diizinkan dan terapkan daftar yang diperbarui.
2. Terapkan perubahan dari Langkah 1.

   Anda tidak dapat melanjutkan ke Langkah 3 sampai perubahan telah menyebar ke semua host yang perlu membaca data ini.
3. Tambahkan DO_NOTHING atribut baru ke tindakan atribut Anda.

   a. Jika Anda mendefinisikan tindakan atribut Anda dengan aTableSchema, tambahkan DO_NOTHING atribut baru ke kelas data beranotasi Anda dengan anotasi@DynamoDBEncryptionDoNothing.

   b. Jika Anda menentukan tindakan atribut secara manual, perbarui tindakan atribut Anda untuk menyertakan atribut baru. Pastikan untuk secara eksplisit mengkonfigurasi atribut baru dengan tindakan DO_NOTHING atribut.
4. Terapkan perubahan dari Langkah 3.

Ubah **SIGN_ONLY** atribut yang ada menjadi **SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT**

Untuk mengubah SIGN_ONLY atribut yang adaSIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT, Anda harus memperbarui tindakan atribut Anda. Setelah Anda menyebarkan pembaruan, klien akan dapat memverifikasi nilai yang ada yang ditulis ke atribut, dan akan terus menandatangani nilai baru yang ditulis ke atribut. Nilai baru yang ditulis ke atribut akan disertakan dalam konteks enkripsi.

Jika Anda menentukan SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT atribut apa pun, maka atribut partisi dan sortir juga harusSIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT.

Menggunakan kelas data beranotasi

Jika Anda menentukan tindakan atribut Anda dengan aTableSchema, perbarui tindakan atribut yang terkait dengan atribut dari @DynamoDBEncryptionSignOnly ke@DynamoDBEncryptionSignAndIncludeInEncryptionContext.

Menggunakan model objek

Jika Anda menentukan tindakan atribut secara manual, perbarui tindakan atribut yang terkait dengan atribut dari SIGN_ONLY ke SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT dalam model objek Anda.

Ubah **SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT** atribut yang ada menjadi **SIGN_ONLY**

Untuk mengubah SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT atribut yang adaSIGN_ONLY, Anda harus memperbarui tindakan atribut Anda. Setelah Anda menyebarkan pembaruan, klien akan dapat memverifikasi nilai yang ada yang ditulis ke atribut, dan akan terus menandatangani nilai baru yang ditulis ke atribut. Nilai baru yang ditulis ke atribut tidak akan disertakan dalam konteks enkripsi.

Sebelum mengubah SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT atribut yang adaSIGN_ONLY, pertimbangkan dengan cermat bagaimana pembaruan Anda dapat memengaruhi fungsionalitas pemasok ID kunci cabang Anda.

Menggunakan kelas data beranotasi

Jika Anda menentukan tindakan atribut Anda dengan aTableSchema, perbarui tindakan atribut yang terkait dengan atribut dari @DynamoDBEncryptionSignAndIncludeInEncryptionContext ke@DynamoDBEncryptionSignOnly.

Menggunakan model objek

Jika Anda menentukan tindakan atribut secara manual, perbarui tindakan atribut yang terkait dengan atribut dari `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` ke `SIGN_ONLY` dalam model objek Anda.

AWS SDK Enkripsi Database untuk DynamoDB bahasa pemrograman yang tersedia

SDK Enkripsi AWS Database untuk DynamoDB tersedia untuk bahasa pemrograman berikut. Pustaka spesifik-bahasa bervariasi, tetapi implementasi yang dihasilkan dapat dioperasikan. Anda dapat mengenkripsi dengan satu implementasi bahasa dan mendekripsi dengan yang lain. Interoperabilitas mungkin tunduk pada kendala bahasa. Jika demikian, kendala ini dijelaskan dalam topik tentang implementasi bahasa.

Topik

- Java
- .NET
- Rust

Java

> Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Topik ini menjelaskan cara menginstal dan menggunakan versi 3. x dari pustaka enkripsi sisi klien Java untuk DynamoDB. Untuk detail tentang pemrograman dengan AWS Database Encryption SDK untuk DynamoDB, lihat contoh Java di aws-database-encryption-sdk repositori -dynamodb aktif. GitHub

> **Note**
>
> Topik berikut fokus pada versi 3. x dari pustaka enkripsi sisi klien Java untuk DynamoDB. Pustaka enkripsi sisi klien kami diubah namanya menjadi AWS Database Encryption SDK. AWS Database Encryption SDK terus mendukung versi Klien Enkripsi DynamoDB lama.

Topik

- Prasyarat
- Penginstalan
- Menggunakan pustaka enkripsi sisi klien Java untuk DynamoDB
- Contoh Java
- Konfigurasikan tabel DynamoDB yang ada untuk menggunakan SDK Enkripsi Database untuk AWS DynamoDB
- Migrasi ke versi 3.x pustaka enkripsi sisi klien Java untuk DynamoDB

Prasyarat

Sebelum Anda menginstal versi 3. x dari pustaka enkripsi sisi klien Java untuk DynamoDB, pastikan Anda memiliki prasyarat berikut.

Lingkungan pengembangan Java

Anda akan membutuhkan Java 8 atau yang lebih baru. Di situs web Oracle, buka Unduhan Java SE, kemudian unduh dan instal Java SE Development Kit (JDK).

Jika Anda menggunakan Oracle JDK, Anda juga harus mengunduh dan menginstal File Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy.

AWS SDK for Java 2.x

SDK Enkripsi AWS Database untuk DynamoDB memerlukan modul DynamoDB Enhanced Client dari modul. AWS SDK for Java 2.x Anda dapat menginstal seluruh SDK atau modul ini saja.

Untuk informasi tentang memperbarui versi Anda AWS SDK untuk Java, lihat Memigrasi dari versi 1.x ke 2.x. AWS SDK untuk Java

Tersedia melalui Apache Maven. AWS SDK untuk Java Anda dapat mendeklarasikan ketergantungan untuk keseluruhan AWS SDK untuk Java, atau hanya modul. `dynamodb-enhanced`

Instal AWS SDK untuk Java menggunakan Apache Maven

- Untuk mengimpor keseluruhan AWS SDK untuk Java sebagai dependensi, deklarasikan dalam file Anda. `pom.xml`

- Untuk membuat dependensi hanya untuk modul Amazon DynamoDB AWS SDK untuk Java di, ikuti petunjuk untuk menentukan modul tertentu. Atur `groupId` ke `software.amazon.awssdk` dan `artifactID` kedynamodb-enhanced.

  > **Note**
  >
  > Jika Anda menggunakan AWS KMS keyring atau keyring AWS KMS Hierarchical, Anda juga perlu membuat dependensi untuk modul. AWS KMS Atur `groupId` ke `software.amazon.awssdk` dan `artifactID` kekms.

Penginstalan

Anda dapat menginstal versi 3. x dari pustaka enkripsi sisi klien Java untuk DynamoDB dengan cara berikut.

Menggunakan Apache Maven

Amazon DynamoDB Encryption Client untuk Java tersedia melalui Apache Maven dengan definisi dependensi berikut.

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

Menggunakan Gradle Kotlin

Anda dapat menggunakan Gradle untuk mendeklarasikan dependensi pada Klien Enkripsi Amazon DynamoDB untuk Java dengan menambahkan yang berikut ini ke bagian dependensi proyek Gradle Anda.

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

Secara manual

Untuk menginstal pustaka enkripsi sisi klien Java untuk DynamoDB, kloning atau unduh repositori -dynamodb. aws-database-encryption-sdk GitHub

Setelah Anda menginstal SDK, mulailah dengan melihat kode contoh dalam panduan ini dan contoh Java di repositori aws-database-encryption-sdk -dynamodb aktif. GitHub

Menggunakan pustaka enkripsi sisi klien Java untuk DynamoDB

> Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Topik ini menjelaskan beberapa fungsi dan kelas pembantu di versi 3. x dari pustaka enkripsi sisi klien Java untuk DynamoDB.

Untuk detail tentang pemrograman dengan pustaka enkripsi sisi klien Java untuk DynamoDB, lihat contoh Java, contoh Java di repositori -dynamodb aktif. aws-database-encryption-sdk GitHub

Topik

- Enkriptor item
- Tindakan atribut dalam SDK Enkripsi AWS Database untuk DynamoDB
- Konfigurasi enkripsi dalam SDK Enkripsi AWS Database untuk DynamoDB
- Memperbarui item dengan SDK Enkripsi AWS Database
- Mendekripsi set yang ditandatangani

Enkriptor item

Pada intinya, AWS Database Encryption SDK untuk DynamoDB adalah enkripsi item. Anda dapat menggunakan versi 3. x pustaka enkripsi sisi klien Java untuk DynamoDB untuk mengenkripsi, menandatangani, memverifikasi, dan mendekripsi item tabel DynamoDB Anda dengan cara berikut.

Klien yang Ditingkatkan DynamoDB

Anda dapat mengkonfigurasi DynamoDB Enhanced Client dengan untuk secara otomatis mengenkripsi dan menandatangani item sisi klien dengan `DynamoDbEncryptionInterceptor` permintaan DynamoDB Anda. `PutItem` Dengan DynamoDB Enhanced Client, Anda dapat menentukan tindakan atribut Anda menggunakan kelas data beranotasi. Sebaiknya gunakan DynamoDB Enhanced Client bila memungkinkan.

DynamoDB Enhanced Client tidak mendukung enkripsi yang dapat dicari.

> **Note**
>
> SDK Enkripsi AWS Database tidak mendukung anotasi pada atribut bersarang.

API DynamoDB tingkat rendah

Anda dapat mengonfigurasi API DynamoDB tingkat rendah dengan untuk secara otomatis mengenkripsi dan menandatangani item `DynamoDbEncryptionInterceptor` sisi klien dengan permintaan DynamoDB Anda. `PutItem`

Anda harus menggunakan API DynamoDB tingkat rendah untuk menggunakan enkripsi yang dapat dicari.

Tingkat yang lebih rendah **`DynamoDbItemEncryptor`**

Tingkat yang lebih rendah `DynamoDbItemEncryptor` secara langsung mengenkripsi dan menandatangani atau mendekripsi dan memverifikasi item tabel Anda tanpa memanggil DynamoDB. Itu tidak membuat DynamoDB atau `PutItem` permintaan`GetItem`. Misalnya, Anda dapat menggunakan level yang lebih rendah `DynamoDbItemEncryptor` untuk langsung mendekripsi dan memverifikasi item DynamoDB yang telah Anda ambil.

Tingkat yang lebih rendah `DynamoDbItemEncryptor` tidak mendukung enkripsi yang dapat dicari.

Tindakan atribut dalam SDK Enkripsi AWS Database untuk DynamoDB

Tindakan atribut menentukan nilai atribut mana yang dienkripsi dan ditandatangani, yang hanya ditandatangani, yang ditandatangani dan disertakan dalam konteks enkripsi, dan mana yang diabaikan.

> **Note**
>
> Untuk menggunakan tindakan `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` kriptografi, Anda harus menggunakan SDK Enkripsi AWS Database versi 3.3 atau yang lebih baru. Terapkan versi baru ke semua pembaca sebelum memperbarui model data Anda untuk disertakan`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

Jika Anda menggunakan API DynamoDB tingkat rendah atau level yang `DynamoDbItemEncryptor` lebih rendah, Anda harus menentukan tindakan atribut secara manual. Jika Anda menggunakan DynamoDB Enhanced Client, Anda dapat menentukan tindakan atribut secara manual, atau Anda dapat menggunakan kelas data beranotasi untuk menghasilkan. `TableSchema` Untuk menyederhanakan proses konfigurasi, kami sarankan menggunakan kelas data beranotasi. Bila Anda menggunakan kelas data beranotasi, Anda hanya perlu memodelkan objek Anda sekali.

> **Note**
>
> Setelah menentukan tindakan atribut, Anda harus menentukan atribut mana yang dikecualikan dari tanda tangan. Untuk mempermudah menambahkan atribut unsigned baru di masa mendatang, sebaiknya pilih awalan yang berbeda (seperti `":`“) untuk mengidentifikasi atribut unsigned Anda. Sertakan awalan ini dalam nama atribut untuk semua atribut yang ditandai `DO_NOTHING` saat Anda menentukan skema DynamoDB dan tindakan atribut.

Gunakan kelas data beranotasi

Gunakan kelas data beranotasi untuk menentukan tindakan atribut Anda dengan DynamoDB Enhanced Client dan. `DynamoDbEncryptionInterceptor` SDK Enkripsi AWS Database untuk DynamoDB menggunakan anotasi atribut DynamoDB standar yang menentukan jenis atribut untuk menentukan cara melindungi atribut. Secara default, semua atribut dienkripsi dan ditandatangani kecuali kunci utama, yang ditandatangani tetapi tidak dienkripsi.

> **Note**
>
> Untuk menggunakan tindakan `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` kriptografi, Anda harus menggunakan SDK Enkripsi AWS Database versi 3.3 atau yang lebih baru. Terapkan versi baru ke semua pembaca sebelum memperbarui model data Anda untuk disertakan`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

Lihat SimpleClass.java di repositori aws-database-encryption-sdk -dynamodb untuk panduan GitHub lebih lanjut tentang anotasi DynamoDB Enhanced Client.

Secara default, atribut kunci primer ditandatangani tetapi tidak dienkripsi (`SIGN_ONLY`) dan semua atribut lainnya dienkripsi dan ditandatangani (). `ENCRYPT_AND_SIGN` Jika Anda mendefinisikan atribut apa pun sebagai`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, maka atribut partisi dan sortir juga harus`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

Untuk menentukan pengecualian, gunakan anotasi enkripsi yang ditentukan dalam pustaka enkripsi sisi klien Java untuk DynamoDB. Misalnya, jika Anda ingin atribut tertentu hanya ditandatangani, gunakan `@DynamoDbEncryptionSignOnly` anotasi. Jika Anda ingin atribut tertentu ditandatangani dan disertakan dalam konteks enkripsi, gunakan`@DynamoDbEncryptionSignAndIncludeInEncryptionContext`. Jika Anda ingin atribut tertentu tidak ditandatangani atau dienkripsi (`DO_NOTHING`), gunakan anotasi`@DynamoDbEncryptionDoNothing`.

> **Note**
>
> SDK Enkripsi AWS Database tidak mendukung anotasi pada atribut bersarang.

Contoh berikut menunjukkan anotasi yang digunakan untuk mendefinisikan`ENCRYPT_AND_SIGN`,`SIGN_ONLY`, dan `DO_NOTHING` atribut tindakan. Untuk contoh yang menunjukkan anotasi yang digunakan untuk mendefinisikan`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, lihat SimpleClass4.java.

```
@DynamoDbBean
public class SimpleClass {

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

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

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

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

```
    }

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

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

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

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

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

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

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

}
```

Gunakan kelas data beranotasi Anda untuk membuat `TableSchema` seperti yang ditunjukkan dalam cuplikan berikut.

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

Tentukan tindakan atribut Anda secara manual

Untuk menentukan tindakan atribut secara manual, buat `Map` objek di mana pasangan nama-nilai mewakili nama atribut dan tindakan yang ditentukan.

Tentukan `ENCRYPT_AND_SIGN` untuk mengenkripsi dan menandatangani atribut. Tentukan `SIGN_ONLY` untuk menandatangani, tetapi tidak mengenkripsi, atribut. Tentukan `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` untuk menandatangani atribut dan sertakan dalam konteks enkripsi. Anda tidak dapat mengenkripsi atribut tanpa menandatanganinya juga. Tentukan `DO_NOTHING` untuk mengabaikan atribut.

Partisi dan atribut sortir harus salah satu `SIGN_ONLY` atau`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Jika Anda mendefinisikan atribut apa pun sebagai`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, maka atribut partisi dan sortir juga harus`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

> **Note**
>
> Untuk menggunakan tindakan `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` kriptografi, Anda harus menggunakan SDK Enkripsi AWS Database versi 3.3 atau yang lebih baru. Terapkan versi baru ke semua pembaca sebelum memperbarui model data Anda untuk disertakan`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

```
final Map<String, CryptoAction> attributeActionsOnEncrypt = new HashMap<>();
// The partition attribute must be signed
attributeActionsOnEncrypt.put("partition_key",
 CryptoAction.SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT);
// The sort attribute must be signed
attributeActionsOnEncrypt.put("sort_key",
 CryptoAction.SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT);
attributeActionsOnEncrypt.put("attribute1", CryptoAction.ENCRYPT_AND_SIGN);
attributeActionsOnEncrypt.put("attribute2", CryptoAction.SIGN_ONLY);
attributeActionsOnEncrypt.put("attribute3",
 CryptoAction.SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT);
attributeActionsOnEncrypt.put(":attribute4", CryptoAction.DO_NOTHING);
```

Konfigurasi enkripsi dalam SDK Enkripsi AWS Database untuk DynamoDB

Bila Anda menggunakan AWS Database Encryption SDK, Anda harus secara eksplisit menentukan konfigurasi enkripsi untuk tabel DynamoDB Anda. Nilai yang diperlukan dalam konfigurasi enkripsi Anda bergantung pada apakah Anda mendefinisikan tindakan atribut secara manual atau dengan kelas data beranotasi.

Cuplikan berikut mendefinisikan konfigurasi enkripsi tabel DynamoDB menggunakan DynamoDB Enhanced Client, `TableSchema`dan mengizinkan atribut unsigned yang ditentukan oleh awalan yang berbeda.

```
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());
```

Nama tabel logis

Sebuah nama tabel logis untuk tabel DynamoDB Anda.

Nama tabel logis terikat secara kriptografis ke semua data yang disimpan dalam tabel untuk menyederhanakan operasi pemulihan DynamoDB. Kami sangat menyarankan untuk menentukan nama tabel DynamoDB Anda sebagai nama tabel logis saat Anda pertama kali menentukan konfigurasi enkripsi Anda. Anda harus selalu menentukan nama tabel logis yang sama. Agar dekripsi berhasil, nama tabel logis harus sesuai dengan nama yang ditentukan pada enkripsi. Jika nama tabel DynamoDB Anda berubah setelah memulihkan tabel DynamoDB Anda dari cadangan, nama tabel logis memastikan bahwa operasi dekripsi masih mengenali tabel.

Atribut yang tidak ditandatangani yang diizinkan

Atribut yang ditandai `DO_NOTHING` dalam tindakan atribut Anda.

Atribut unsigned yang diizinkan memberi tahu klien atribut mana yang dikecualikan dari tanda tangan. Klien mengasumsikan bahwa semua atribut lainnya termasuk dalam tanda tangan. Kemudian, saat mendekripsi catatan, klien menentukan atribut mana yang perlu diverifikasi dan mana yang harus diabaikan dari atribut unsigned yang diizinkan yang Anda tentukan. Anda tidak dapat menghapus atribut dari atribut yang tidak ditandatangani yang diizinkan.

Anda dapat menentukan atribut unsigned yang diizinkan secara eksplisit dengan membuat array yang mencantumkan semua atribut Anda. `DO_NOTHING` Anda juga dapat menentukan awalan yang berbeda saat menamai `DO_NOTHING` atribut Anda dan menggunakan awalan untuk memberi tahu klien atribut mana yang tidak ditandatangani. Kami sangat menyarankan untuk menentukan awalan yang berbeda karena menyederhanakan proses penambahan `DO_NOTHING` atribut baru di masa depan. Untuk informasi selengkapnya, lihat Memperbarui model data Anda.

Jika Anda tidak menentukan awalan untuk semua `DO_NOTHING` atribut, Anda dapat mengonfigurasi `allowedUnsignedAttributes` array yang secara eksplisit mencantumkan semua atribut yang diharapkan klien tidak ditandatangani saat bertemu dengan mereka pada dekripsi. Anda hanya harus secara eksplisit mendefinisikan atribut unsigned yang diizinkan jika benar-benar diperlukan.

Konfigurasi Pencarian (Opsional)

`SearchConfig`Mendefinisikan versi beacon.

`SearchConfig`Harus ditentukan untuk menggunakan enkripsi yang dapat dicari atau suar yang ditandatangani.

Suite Algoritma (Opsional)

`algorithmSuiteId`Mendefinisikan algoritma mana yang sesuai dengan AWS Database Encryption SDK yang digunakan.

Kecuali Anda secara eksplisit menentukan rangkaian algoritme alternatif, SDK Enkripsi AWS Database menggunakan rangkaian algoritme default. Rangkaian algoritma default menggunakan algoritma AES-GCM dengan derivasi kunci, tanda tangan digital, dan komitmen kunci. Meskipun rangkaian algoritme default kemungkinan cocok untuk sebagian besar aplikasi, Anda dapat memilih rangkaian algoritma alternatif. Misalnya, beberapa model kepercayaan akan dipenuhi oleh rangkaian algoritma tanpa tanda tangan digital. Untuk informasi tentang rangkaian algoritme yang didukung SDK Enkripsi AWS Database, lihatRangkaian algoritme yang didukung di SDK Enkripsi AWS Database.

Untuk memilih rangkaian algoritma AES-GCM tanpa tanda tangan digital ECDSA, sertakan cuplikan berikut dalam konfigurasi enkripsi tabel Anda.

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

Memperbarui item dengan SDK Enkripsi AWS Database

SDK Enkripsi AWS Database tidak mendukung ddb: UpdateItem untuk item yang telah dienkripsi atau ditandatangani. Untuk memperbarui item terenkripsi atau ditandatangani, Anda harus menggunakan ddb:. PutItem Saat Anda menentukan kunci utama yang sama dengan item yang ada dalam `PutItem` permintaan Anda, item baru sepenuhnya menggantikan item yang ada. Anda juga dapat menggunakan CLOBBER untuk menghapus dan mengganti semua atribut yang disimpan setelah memperbarui item Anda.

Mendekripsi set yang ditandatangani

Di SDK Enkripsi AWS Database versi 3.0.0 dan 3.1.0, jika Anda menentukan atribut tipe set sebagai`SIGN_ONLY`, nilai himpunan akan dikanonikalisasi dalam urutan yang disediakan. DynamoDB tidak mempertahankan urutan set. Akibatnya, ada kemungkinan validasi tanda tangan dari item yang berisi set akan gagal. Validasi tanda tangan gagal ketika nilai set dikembalikan dalam urutan yang berbeda dari yang diberikan ke SDK Enkripsi AWS Database, meskipun atribut set berisi nilai yang sama.

> **Note**
>
> Versi 3.1.1 dan yang lebih baru dari AWS Database Encryption SDK mengkanonikalisasikan nilai semua atribut tipe set, sehingga nilai dibaca dalam urutan yang sama dengan yang ditulis ke DynamoDB.

Jika validasi tanda tangan gagal, operasi dekripsi gagal dan mengembalikan pesan kesalahan berikut.

```
software.amazon.cryptography.dbencryptionsdk.structuredencryption.model. StructuredEncryptionException: Tidak ada tag penerima yang cocok.
```

Jika Anda menerima pesan kesalahan di atas, dan yakin bahwa item yang Anda coba dekripsi menyertakan set yang ditandatangani menggunakan versi 3.0.0 atau 3.1.0, lihat DecryptWithPermutedirektori repositori aws-database-encryption-sdk -dynamodb-java untuk detail tentang cara berhasil memvalidasi set. GitHub

Contoh Java

> Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Contoh berikut menunjukkan cara menggunakan pustaka enkripsi sisi klien Java untuk DynamoDB untuk melindungi item tabel dalam aplikasi Anda. Anda dapat menemukan lebih banyak contoh (dan berkontribusi sendiri) di contoh Java di repositori aws-database-encryption-sdk -dynamodb di. GitHub

Contoh berikut menunjukkan cara mengonfigurasi pustaka enkripsi sisi klien Java untuk DynamoDB dalam tabel Amazon DynamoDB baru yang tidak terisi. Jika Anda ingin mengonfigurasi tabel Amazon DynamoDB yang ada untuk enkripsi sisi klien, lihat. Tambahkan versi 3.x ke tabel yang ada

Topik

- Menggunakan klien yang disempurnakan DynamoDB
- Menggunakan API DynamoDB tingkat rendah
- Menggunakan level yang lebih rendah DynamoDbItemEncryptor

Menggunakan klien yang disempurnakan DynamoDB

Contoh berikut menunjukkan cara menggunakan DynamoDB Enhanced Client `DynamoDbEncryptionInterceptor` dan dengan keyring untuk mengenkripsi item tabel DynamoDB AWS KMS sebagai bagian dari panggilan API DynamoDB Anda.

Anda dapat menggunakan keyring apa pun yang didukung dengan DynamoDB Enhanced Client, tetapi sebaiknya gunakan salah AWS KMS satu gantungan kunci bila memungkinkan.

> **Note**
>
> DynamoDB Enhanced Client tidak mendukung enkripsi yang dapat dicari. Gunakan `DynamoDbEncryptionInterceptor` dengan API DynamoDB tingkat rendah untuk menggunakan enkripsi yang dapat dicari.

Lihat contoh kode lengkapnya: EnhancedPutGetExample.java

Langkah 1: Buat AWS KMS keyring

Contoh berikut digunakan `CreateAwsKmsMrkMultiKeyring` untuk membuat AWS KMS keyring dengan kunci KMS enkripsi simetris. `CreateAwsKmsMrkMultiKeyring`Metode ini memastikan bahwa keyring akan menangani tombol Single-region dan Multi-region dengan benar.

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

Langkah 2: Buat skema tabel dari kelas data beranotasi

Contoh berikut menggunakan kelas data beranotasi untuk membuat. `TableSchema`

Contoh ini mengasumsikan bahwa kelas data beranotasi dan tindakan atribut didefinisikan menggunakan .java. SimpleClass Untuk panduan selengkapnya tentang menganotasi tindakan atribut Anda, lihat. Gunakan kelas data beranotasi

> **Note**
>
> SDK Enkripsi AWS Database tidak mendukung anotasi pada atribut bersarang.

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

Langkah 3: Tentukan atribut mana yang dikecualikan dari tanda tangan

Contoh berikut mengasumsikan bahwa semua `DO_NOTHING` atribut berbagi awalan yang berbeda "`:`“, dan menggunakan awalan untuk menentukan atribut unsigned yang diizinkan. Klien mengasumsikan bahwa nama atribut apa pun dengan awalan "`:`" dikecualikan dari tanda tangan. Untuk informasi selengkapnya, lihat Allowed unsigned attributes.

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

Langkah 4: Buat konfigurasi enkripsi

Contoh berikut mendefinisikan `tableConfigs` Peta yang mewakili konfigurasi enkripsi untuk tabel DynamoDB.

Contoh ini menentukan nama tabel DynamoDB sebagai nama tabel logis. Kami sangat menyarankan untuk menentukan nama tabel DynamoDB Anda sebagai nama tabel logis saat Anda pertama kali menentukan konfigurasi enkripsi Anda. Untuk informasi selengkapnya, lihat Konfigurasi enkripsi dalam SDK Enkripsi AWS Database untuk DynamoDB.

> **Note**
>
> Untuk menggunakan enkripsi yang dapat dicari atau suar yang ditandatangani, Anda juga harus menyertakan `SearchConfig`dalam konfigurasi enkripsi Anda.

```
final Map<String, DynamoDbEnhancedTableEncryptionConfig> tableConfigs = new
 HashMap<>();
tableConfigs.put(ddbTableName,
    DynamoDbEnhancedTableEncryptionConfig.builder()
        .logicalTableName(ddbTableName)
        .keyring(kmsKeyring)
        .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
        .schemaOnEncrypt(tableSchema)
        .build());
```

Langkah 5: Menciptakan **`DynamoDbEncryptionInterceptor`**

Contoh berikut membuat yang baru `DynamoDbEncryptionInterceptor` dengan `tableConfigs` dari Langkah 4.

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

Langkah 6: Buat klien AWS SDK DynamoDB baru

Contoh berikut membuat klien AWS SDK DynamoDB baru menggunakan **`interceptor`** dari Langkah 5.

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

Langkah 7: Buat DynamoDB Enhanced Client dan buat tabel

Contoh berikut membuat DynamoDB Enhanced Client menggunakan klien AWS SDK DynamoDB yang dibuat pada Langkah 6 dan membuat tabel menggunakan class data beranotasi.

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

Langkah 8: Enkripsi dan tandatangani item tabel

Contoh berikut menempatkan item ke dalam tabel DynamoDB menggunakan DynamoDB Enhanced Client. Item dienkripsi dan ditandatangani sisi klien sebelum dikirim ke 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);
```

Menggunakan API DynamoDB tingkat rendah

Contoh berikut menunjukkan cara menggunakan API DynamoDB tingkat rendah dengan keyring untuk secara otomatis mengenkripsi dan menandatangani item sisi klien dengan AWS KMS permintaan DynamoDB Anda. `PutItem`

Anda dapat menggunakan keyring apa pun yang didukung, tetapi kami sarankan menggunakan salah satu AWS KMS gantungan kunci bila memungkinkan.

Lihat contoh kode lengkapnya: BasicPutGetExample.java

Langkah 1: Buat AWS KMS keyring

Contoh berikut digunakan `CreateAwsKmsMrkMultiKeyring` untuk membuat AWS KMS keyring dengan kunci KMS enkripsi simetris. `CreateAwsKmsMrkMultiKeyring`Metode ini memastikan bahwa keyring akan menangani tombol Single-region dan Multi-region dengan benar.

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

Langkah 2: Konfigurasikan tindakan atribut Anda

Contoh berikut mendefinisikan `attributeActionsOnEncrypt` Peta yang mewakili tindakan atribut sampel untuk item tabel.

> **Note**
>
> Contoh berikut tidak mendefinisikan atribut apa pun sebagai`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Jika Anda menentukan `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` atribut apa pun, maka atribut partisi dan sortir juga harus`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

```
final Map<String, CryptoAction> attributeActionsOnEncrypt = new HashMap<>();
// The partition attribute must be SIGN_ONLY
```

```
attributeActionsOnEncrypt.put("partition_key", CryptoAction.SIGN_ONLY);
// The sort attribute must be SIGN_ONLY
attributeActionsOnEncrypt.put("sort_key", CryptoAction.SIGN_ONLY);
attributeActionsOnEncrypt.put("attribute1", CryptoAction.ENCRYPT_AND_SIGN);
attributeActionsOnEncrypt.put("attribute2", CryptoAction.SIGN_ONLY);
attributeActionsOnEncrypt.put(":attribute3", CryptoAction.DO_NOTHING);
```

Langkah 3: Tentukan atribut mana yang dikecualikan dari tanda tangan

Contoh berikut mengasumsikan bahwa semua `DO_NOTHING` atribut berbagi awalan yang berbeda "`:`“, dan menggunakan awalan untuk menentukan atribut unsigned yang diizinkan. Klien mengasumsikan bahwa nama atribut apa pun dengan awalan "`:`" dikecualikan dari tanda tangan. Untuk informasi selengkapnya, lihat Allowed unsigned attributes.

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

Langkah 4: Tentukan konfigurasi enkripsi tabel DynamoDB

Contoh berikut mendefinisikan `tableConfigs` Peta yang mewakili konfigurasi enkripsi untuk tabel DynamoDB ini.

Contoh ini menentukan nama tabel DynamoDB sebagai nama tabel logis. Kami sangat menyarankan untuk menentukan nama tabel DynamoDB Anda sebagai nama tabel logis saat Anda pertama kali menentukan konfigurasi enkripsi Anda. Untuk informasi selengkapnya, lihat Konfigurasi enkripsi dalam SDK Enkripsi AWS Database untuk DynamoDB.

> **Note**
>
> Untuk menggunakan enkripsi yang dapat dicari atau suar yang ditandatangani, Anda juga harus menyertakan `SearchConfig`dalam konfigurasi enkripsi Anda.

```
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);
```

Langkah 5: Buat **`DynamoDbEncryptionInterceptor`**

Contoh berikut menciptakan `DynamoDbEncryptionInterceptor` menggunakan `tableConfigs` dari Langkah 4.

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

Langkah 6: Buat klien AWS SDK DynamoDB baru

Contoh berikut membuat klien AWS SDK DynamoDB baru menggunakan **`interceptor`** dari Langkah 5.

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

Langkah 7: Enkripsi dan tandatangani item tabel DynamoDB

Contoh berikut mendefinisikan `item` Peta yang mewakili item tabel sampel dan menempatkan item dalam tabel DynamoDB. Item dienkripsi dan ditandatangani sisi klien sebelum dikirim ke 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);
```

Menggunakan level yang lebih rendah DynamoDbItemEncryptor

Contoh berikut menunjukkan cara menggunakan level yang lebih rendah `DynamoDbItemEncryptor` dengan AWS KMS keyring untuk langsung mengenkripsi dan menandatangani item tabel. `DynamoDbItemEncryptor`Itu tidak menempatkan item di tabel DynamoDB Anda.

Anda dapat menggunakan keyring apa pun yang didukung dengan DynamoDB Enhanced Client, tetapi sebaiknya gunakan salah AWS KMS satu gantungan kunci bila memungkinkan.

> **Note**
>
> Tingkat yang lebih rendah `DynamoDbItemEncryptor` tidak mendukung enkripsi yang dapat dicari. Gunakan `DynamoDbEncryptionInterceptor` dengan API DynamoDB tingkat rendah untuk menggunakan enkripsi yang dapat dicari.

Lihat contoh kode lengkapnya: ItemEncryptDecryptExample.java

Langkah 1: Buat AWS KMS keyring

Contoh berikut digunakan `CreateAwsKmsMrkMultiKeyring` untuk membuat AWS KMS keyring dengan kunci KMS enkripsi simetris. `CreateAwsKmsMrkMultiKeyring`Metode ini memastikan bahwa keyring akan menangani tombol Single-region dan Multi-region dengan benar.

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

Langkah 2: Konfigurasikan tindakan atribut Anda

Contoh berikut mendefinisikan `attributeActionsOnEncrypt` Peta yang mewakili tindakan atribut sampel untuk item tabel.

> **Note**
>
> Contoh berikut tidak mendefinisikan atribut apa pun sebagai`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Jika Anda menentukan `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` atribut apa pun, maka atribut partisi dan sortir juga harus`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

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

Langkah 3: Tentukan atribut mana yang dikecualikan dari tanda tangan

Contoh berikut mengasumsikan bahwa semua `DO_NOTHING` atribut berbagi awalan yang berbeda "`:`“, dan menggunakan awalan untuk menentukan atribut unsigned yang diizinkan. Klien mengasumsikan bahwa nama atribut apa pun dengan awalan "`:`" dikecualikan dari tanda tangan. Untuk informasi selengkapnya, lihat Allowed unsigned attributes.

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

Langkah 4: Tentukan **`DynamoDbItemEncryptor`** konfigurasi

Contoh berikut mendefinisikan konfigurasi untuk. `DynamoDbItemEncryptor`

Contoh ini menentukan nama tabel DynamoDB sebagai nama tabel logis. Kami sangat menyarankan untuk menentukan nama tabel DynamoDB Anda sebagai nama tabel logis saat Anda pertama kali menentukan konfigurasi enkripsi Anda. Untuk informasi selengkapnya, lihat Konfigurasi enkripsi dalam SDK Enkripsi AWS Database untuk DynamoDB.

```
final DynamoDbItemEncryptorConfig config = DynamoDbItemEncryptorConfig.builder()
        .logicalTableName(ddbTableName)
        .partitionKeyName("partition_key")
        .sortKeyName("sort_key")
        .attributeActionsOnEncrypt(attributeActionsOnEncrypt)
```

```
        .keyring(kmsKeyring)
        .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
        .build();
```

Langkah 5: Buat **`DynamoDbItemEncryptor`**

Contoh berikut membuat baru `DynamoDbItemEncryptor` menggunakan `config` dari Langkah 4.

```
final DynamoDbItemEncryptor itemEncryptor = DynamoDbItemEncryptor.builder()
        .DynamoDbItemEncryptorConfig(config)
        .build();
```

Langkah 6: Langsung mengenkripsi dan menandatangani item tabel

Contoh berikut langsung mengenkripsi dan menandatangani item menggunakan. `DynamoDbItemEncryptor` `DynamoDbItemEncryptor`Itu tidak menempatkan item di tabel DynamoDB Anda.

```
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();
```

Konfigurasikan tabel DynamoDB yang ada untuk menggunakan SDK Enkripsi Database untuk AWS DynamoDB

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Dengan versi 3. x dari pustaka enkripsi sisi klien Java untuk DynamoDB, Anda dapat mengonfigurasi tabel Amazon DynamoDB yang ada untuk enkripsi sisi klien. Topik ini memberikan panduan tentang tiga langkah yang harus Anda ambil untuk menambahkan versi 3. x ke tabel DynamoDB yang sudah ada dan terisi.

Prasyarat

Versi 3. x dari pustaka enkripsi sisi klien Java untuk DynamoDB memerlukan DynamoDB Enhanced Client yang disediakan di. AWS SDK for Java 2.x Jika Anda masih menggunakan Dynamo DBMapper, Anda harus bermigrasi ke untuk menggunakan DynamoDB AWS SDK for Java 2.x Enhanced Client.

Ikuti petunjuk untuk bermigrasi dari versi 1.x ke 2.x dari file. AWS SDK untuk Java

Kemudian, ikuti petunjuk untuk Memulai menggunakan DynamoDB Enhanced Client API.

Sebelum mengkonfigurasi tabel Anda untuk menggunakan pustaka enkripsi sisi klien Java untuk DynamoDB, Anda perlu membuat `TableSchema`menggunakan kelas data beranotasi dan membuat klien yang disempurnakan.

Langkah 1: Bersiaplah untuk membaca dan menulis item terenkripsi

Selesaikan langkah-langkah berikut untuk mempersiapkan klien SDK Enkripsi AWS Database Anda untuk membaca dan menulis item terenkripsi. Setelah Anda menerapkan perubahan berikut, klien Anda akan terus membaca dan menulis item teks biasa. Ini tidak akan mengenkripsi atau menandatangani item baru yang ditulis ke tabel, tetapi akan dapat mendekripsi item terenkripsi segera setelah muncul. Perubahan ini mempersiapkan klien untuk mulai mengenkripsi item baru. Perubahan berikut harus diterapkan ke setiap pembaca sebelum Anda melanjutkan ke langkah berikutnya.

1. Tentukan tindakan atribut Anda

   Perbarui kelas data beranotasi Anda untuk menyertakan tindakan atribut yang menentukan nilai atribut mana yang akan dienkripsi dan ditandatangani, yang hanya akan ditandatangani, dan mana yang akan diabaikan.

   Lihat SimpleClass.java di repositori aws-database-encryption-sdk -dynamodb untuk panduan GitHub lebih lanjut tentang anotasi DynamoDB Enhanced Client.

   Secara default, atribut kunci primer ditandatangani tetapi tidak dienkripsi (`SIGN_ONLY`) dan semua atribut lainnya dienkripsi dan ditandatangani (). `ENCRYPT_AND_SIGN` Untuk

menentukan pengecualian, gunakan anotasi enkripsi yang ditentukan dalam pustaka enkripsi sisi klien Java untuk DynamoDB. Misalnya, jika Anda ingin atribut tertentu ditandatangani hanya gunakan `@DynamoDbEncryptionSignOnly` anotasi. Jika Anda ingin atribut tertentu ditandatangani dan disertakan dalam konteks enkripsi, gunakan `@DynamoDbEncryptionSignAndIncludeInEncryptionContext` anotasi. Jika Anda ingin atribut tertentu tidak ditandatangani atau dienkripsi (`DO_NOTHING`), gunakan anotasi`@DynamoDbEncryptionDoNothing`.

> **Note**
>
> Jika Anda menentukan `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` atribut apa pun, maka atribut partisi dan sortir juga harus`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.
> Untuk contoh yang menunjukkan anotasi yang digunakan untuk mendefinisikan`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, lihat SimpleClass4.java.

Misalnya anotasi, lihatGunakan kelas data beranotasi.

2. Tentukan atribut mana yang akan dikecualikan dari tanda tangan

   Contoh berikut mengasumsikan bahwa semua `DO_NOTHING` atribut berbagi awalan yang berbeda "`:`“, dan menggunakan awalan untuk menentukan atribut unsigned yang diizinkan. Klien akan menganggap bahwa nama atribut apa pun dengan awalan `:` "" dikecualikan dari tanda tangan. Untuk informasi selengkapnya, lihat Allowed unsigned attributes.

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

3. Buat keyring

   Contoh berikut membuat AWS KMS keyring. AWS KMS Keyring menggunakan enkripsi simetris atau RSA asimetris AWS KMS keys untuk menghasilkan, mengenkripsi, dan mendekripsi kunci data.

   Contoh ini digunakan `CreateMrkMultiKeyring` untuk membuat AWS KMS keyring dengan kunci KMS enkripsi simetris. `CreateAwsKmsMrkMultiKeyring`Metode ini memastikan bahwa keyring akan menangani tombol Single-region dan Multi-region dengan benar.

   ```
   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. Tentukan konfigurasi enkripsi tabel DynamoDB

   Contoh berikut mendefinisikan `tableConfigs` Peta yang mewakili konfigurasi enkripsi untuk tabel DynamoDB ini.

   Contoh ini menentukan nama tabel DynamoDB sebagai nama tabel logis. Kami sangat menyarankan untuk menentukan nama tabel DynamoDB Anda sebagai nama tabel logis saat Anda pertama kali menentukan konfigurasi enkripsi Anda. Untuk informasi selengkapnya, lihat Konfigurasi enkripsi dalam SDK Enkripsi AWS Database untuk DynamoDB.

   Anda harus menentukan `FORCE_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT` sebagai plaintext override. Kebijakan ini terus membaca dan menulis item teks biasa, membaca item terenkripsi, dan mempersiapkan klien untuk menulis item terenkripsi.

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

 .plaintextOverride(PlaintextOverride.FORCE_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT)
        .build();
tableConfigs.put(ddbTableName, config);
```

5. Buat **`DynamoDbEncryptionInterceptor`**

   Contoh berikut menciptakan `DynamoDbEncryptionInterceptor` menggunakan `tableConfigs` dari Langkah 3.

```
DynamoDbEncryptionInterceptor interceptor = DynamoDbEncryptionInterceptor.builder()
```

```
        .config(DynamoDbTablesEncryptionConfig.builder()
                .tableEncryptionConfigs(tableConfigs)
                .build())
        .build();
```

Langkah 2: Tulis item terenkripsi dan ditandatangani

Perbarui kebijakan plaintext dalam `DynamoDbEncryptionInterceptor` konfigurasi Anda untuk memungkinkan klien menulis item terenkripsi dan ditandatangani. Setelah Anda menerapkan perubahan berikut, klien akan mengenkripsi dan menandatangani item baru berdasarkan tindakan atribut yang Anda konfigurasikan di Langkah 1. Klien akan dapat membaca item teks biasa dan item terenkripsi dan ditandatangani.

Sebelum Anda melanjutkan ke Langkah 3, Anda harus mengenkripsi dan menandatangani semua item plaintext yang ada di tabel Anda. Tidak ada metrik atau kueri tunggal yang dapat Anda jalankan untuk mengenkripsi item plaintext yang ada dengan cepat. Gunakan proses yang paling masuk akal untuk sistem Anda. Misalnya, Anda dapat menggunakan proses asinkron yang memindai tabel secara perlahan dan menulis ulang item menggunakan tindakan atribut dan konfigurasi enkripsi yang Anda tentukan. Untuk mengidentifikasi item plaintext dalam tabel Anda, kami sarankan memindai semua item yang tidak berisi `aws_dbe_head` dan `aws_dbe_foot` atribut yang ditambahkan SDK Enkripsi AWS Database ke item saat dienkripsi dan ditandatangani.

Contoh berikut memperbarui konfigurasi enkripsi tabel dari Langkah 1. Anda harus memperbarui penggantian plaintext dengan. `FORBID_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT` Kebijakan ini terus membaca item teks biasa, tetapi juga membaca dan menulis item terenkripsi. Buat yang baru `DynamoDbEncryptionInterceptor` menggunakan yang diperbarui`tableConfigs`.

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

 .plaintextOverride(PlaintextOverride.FORBID_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT)
        .build();
tableConfigs.put(ddbTableName, config);
```

Langkah 3: Hanya baca item terenkripsi dan ditandatangani

Setelah Anda mengenkripsi dan menandatangani semua item Anda, perbarui penggantian plaintext dalam `DynamoDbEncryptionInterceptor` konfigurasi Anda untuk hanya mengizinkan klien membaca dan menulis item terenkripsi dan ditandatangani. Setelah Anda menerapkan perubahan berikut, klien akan mengenkripsi dan menandatangani item baru berdasarkan tindakan atribut yang Anda konfigurasikan di Langkah 1. Klien hanya akan dapat membaca item terenkripsi dan ditandatangani.

Contoh berikut memperbarui konfigurasi enkripsi tabel dari Langkah 2. Anda dapat memperbarui penggantian plaintext dengan `FORBID_WRITE_PLAINTEXT_FORBID_READ_PLAINTEXT` atau menghapus kebijakan plaintext dari konfigurasi Anda. Klien hanya membaca dan menulis item terenkripsi dan ditandatangani secara default. Buat yang baru `DynamoDbEncryptionInterceptor` menggunakan yang diperbarui`tableConfigs`.

```
final Map<String, DynamoDbTableEncryptionConfig> tableConfigs = new HashMap<>();
final DynamoDbTableEncryptionConfig config = DynamoDbTableEncryptionConfig.builder()
        .logicalTableName(ddbTableName)
        .partitionKeyName("partition_key")
        .sortKeyName("sort_key")
        .schemaOnEncrypt(tableSchema)
        .keyring(kmsKeyring)
        .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
        // Optional: you can also remove the plaintext policy from your configuration

 .plaintextOverride(PlaintextOverride.FORBID_WRITE_PLAINTEXT_FORBID_READ_PLAINTEXT)
        .build();
tableConfigs.put(ddbTableName, config);
```

Migrasi ke versi 3.x pustaka enkripsi sisi klien Java untuk DynamoDB

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Versi 3. x dari pustaka enkripsi sisi klien Java untuk DynamoDB adalah penulisan ulang utama dari 2. basis kode x. Ini mencakup banyak pembaruan, seperti format data terstruktur baru, dukungan multitenancy yang ditingkatkan, perubahan skema yang mulus, dan dukungan enkripsi yang dapat dicari. Topik ini memberikan panduan tentang cara memigrasikan kode Anda ke versi 3. x.

Migrasi dari versi 1.x ke 2.x

Migrasi ke versi 2. x sebelum Anda bermigrasi ke versi 3. x. Versi 2. x mengubah simbol untuk Penyedia Terbaru dari `MostRecentProvider` ke`CachingMostRecentProvider`. Jika saat ini Anda menggunakan versi 1. x dari pustaka enkripsi sisi klien Java untuk DynamoDB dengan `MostRecentProvider` simbol, Anda harus memperbarui nama simbol dalam kode Anda ke. `CachingMostRecentProvider` Untuk informasi selengkapnya, lihat Pembaruan ke Penyedia Terbaru.

Migrasi dari versi 2.x ke 3.x

Prosedur berikut menjelaskan cara memigrasikan kode Anda dari versi 2. x ke versi 3. x dari pustaka enkripsi sisi klien Java untuk DynamoDB.

Langkah 1. Bersiaplah untuk membaca item dalam format baru

Selesaikan langkah-langkah berikut untuk mempersiapkan klien SDK Enkripsi AWS Database Anda untuk membaca item dalam format baru. Setelah Anda menerapkan perubahan berikut, klien Anda akan terus berperilaku dengan cara yang sama seperti di versi 2. x. Klien Anda akan terus membaca dan menulis item dalam versi 2. x format, tetapi perubahan ini mempersiapkan klien untuk membaca item dalam format baru.

Perbarui AWS SDK untuk Java ke versi 2.x

Versi 3. x dari pustaka enkripsi sisi klien Java untuk DynamoDB memerlukan DynamoDB Enhanced Client. DynamoDB Enhanced Client menggantikan Dynamo yang digunakan dalam DBMapper versi sebelumnya. Untuk menggunakan klien yang disempurnakan, Anda harus menggunakan AWS SDK for Java 2.x.

Ikuti petunjuk untuk bermigrasi dari versi 1.x ke 2.x. AWS SDK untuk Java

Untuk informasi lebih lanjut tentang AWS SDK for Java 2.x modul apa yang diperlukan, lihatPrasyarat.

Konfigurasikan klien Anda untuk membaca item yang dienkripsi oleh versi lama

Prosedur berikut memberikan gambaran tentang langkah-langkah yang ditunjukkan dalam contoh kode di bawah ini.

1. Buat keyring.

Keyrings dan manajer bahan kriptografi menggantikan penyedia bahan kriptografi yang digunakan dalam versi sebelumnya dari pustaka enkripsi sisi klien Java untuk DynamoDB.

> **Important**
>
> Kunci pembungkus yang Anda tentukan saat membuat keyring harus berupa kunci pembungkus yang sama yang Anda gunakan dengan penyedia bahan kriptografi Anda di versi 2. x.

2. Buat skema tabel di atas kelas beranotasi Anda.

   Langkah ini mendefinisikan tindakan atribut yang akan digunakan ketika Anda mulai menulis item dalam format baru.

   Untuk panduan tentang penggunaan DynamoDB Enhanced Client baru, lihat Menghasilkan *TableSchema* a di AWS SDK untuk Java Panduan Pengembang.

   Contoh berikut mengasumsikan Anda memperbarui kelas beranotasi Anda dari versi 2. x menggunakan anotasi tindakan atribut baru. Untuk panduan selengkapnya tentang anotasi tindakan atribut Anda, lihat. Gunakan kelas data beranotasi

   > **Note**
   >
   > Jika Anda menentukan `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` atribut apa pun, maka atribut partisi dan sortir juga harus`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.
   > Untuk contoh yang menunjukkan anotasi yang digunakan untuk mendefinisikan`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, lihat SimpleClass4.java.

3. Tentukan atribut mana yang dikecualikan dari tanda tangan.
4. Konfigurasikan peta eksplisit tindakan atribut yang dikonfigurasi di kelas model versi 2.x Anda.

   Langkah ini mendefinisikan tindakan atribut yang digunakan untuk menulis item dalam format lama.

5. Konfigurasikan yang `DynamoDBEncryptor` Anda gunakan di versi 2. x dari pustaka enkripsi sisi klien Java untuk DynamoDB.

6. Konfigurasikan perilaku lama.
7. Buat a`DynamoDbEncryptionInterceptor`.
8. Buat klien AWS SDK DynamoDB baru.
9. Buat `DynamoDBEnhancedClient` dan buat tabel dengan kelas model Anda.

   Untuk informasi selengkapnya tentang DynamoDB Enhanced Client, lihat membuat klien yang disempurnakan.

```
public class MigrationExampleStep1 {

    public static void MigrationStep1(String kmsKeyId, String ddbTableName, int
 sortReadValue) {
        // 1. Create a Keyring.
        //    This example creates an AWS KMS Keyring that specifies the
        //    same kmsKeyId previously used in the version 2.x configuration.
        //    It uses the 'CreateMrkMultiKeyring' method to create the
        //    keyring, so that the keyring can correctly handle both single
        //    region and Multi-Region KMS Keys.
        //    Note that this example uses the AWS SDK for Java v2 KMS client.
        final MaterialProviders matProv = MaterialProviders.builder()
                .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
                .build();
        final CreateAwsKmsMrkMultiKeyringInput keyringInput =
 CreateAwsKmsMrkMultiKeyringInput.builder()
                .generator(kmsKeyId)
                .build();
        final IKeyring kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);

        // 2. Create a Table Schema over your annotated class.
        //    For guidance on using the new attribute actions
        //    annotations, see SimpleClass.java in the
        //    aws-database-encryption-sdk-dynamodb GitHub repository.
        //    All primary key attributes must be signed but not encrypted
        //    and by default all non-primary key attributes
        //    are encrypted and signed (ENCRYPT_AND_SIGN).
        //    If you want a particular non-primary key attribute to be signed but
        //    not encrypted, use the 'DynamoDbEncryptionSignOnly' annotation.
        //    If you want a particular attribute to be neither signed nor encrypted
        //    (DO_NOTHING), use the 'DynamoDbEncryptionDoNothing' annotation.
        final TableSchema<SimpleClass> schemaOnEncrypt =
 TableSchema.fromBean(SimpleClass.class);
```

```
        // 3. Define which attributes the client should expect to be excluded
        //    from the signature when reading items.
        //    This value represents all unsigned attributes across the entire
        //    dataset.
        final List<String> allowedUnsignedAttributes = Arrays.asList("attribute3");

        // 4. Configure an explicit map of the attribute actions configured
        //    in your version 2.x modeled class.
        final Map<String, CryptoAction> legacyActions = new HashMap<>();
        legacyActions.put("partition_key", CryptoAction.SIGN_ONLY);
        legacyActions.put("sort_key", CryptoAction.SIGN_ONLY);
        legacyActions.put("attribute1", CryptoAction.ENCRYPT_AND_SIGN);
        legacyActions.put("attribute2", CryptoAction.SIGN_ONLY);
        legacyActions.put("attribute3", CryptoAction.DO_NOTHING);

        // 5. Configure the DynamoDBEncryptor that you used in version 2.x.
        final AWSKMS kmsClient = AWSKMSClientBuilder.defaultClient();
        final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kmsClient,
kmsKeyId);
        final DynamoDBEncryptor oldEncryptor = DynamoDBEncryptor.getInstance(cmp);

        // 6. Configure the legacy behavior.
        //    Input the DynamoDBEncryptor and attribute actions created in
        //    the previous steps. For Legacy Policy, use
        //    'FORCE_LEGACY_ENCRYPT_ALLOW_LEGACY_DECRYPT'. This policy continues to
read
        //    and write items using the old format, but will be able to read
        //    items written in the new format as soon as they appear.
        final LegacyOverride legacyOverride = LegacyOverride
                .builder()
                .encryptor(oldEncryptor)
                .policy(LegacyPolicy.FORCE_LEGACY_ENCRYPT_ALLOW_LEGACY_DECRYPT)
                .attributeActionsOnEncrypt(legacyActions)
                .build();

        // 7. Create a DynamoDbEncryptionInterceptor with the above configuration.
        final Map<String, DynamoDbEnhancedTableEncryptionConfig> tableConfigs = new
HashMap<>();
        tableConfigs.put(ddbTableName,
                DynamoDbEnhancedTableEncryptionConfig.builder()
                        .logicalTableName(ddbTableName)
                        .keyring(kmsKeyring)
                        .allowedUnsignedAttributes(allowedUnsignedAttributes)
```

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

        // 8. Create a new AWS SDK DynamoDb client using the
        //    interceptor from Step 7.
        final DynamoDbClient ddb = DynamoDbClient.builder()
                .overrideConfiguration(
                        ClientOverrideConfiguration.builder()
                                .addExecutionInterceptor(interceptor)
                                .build())
                .build();

        // 9. Create the DynamoDbEnhancedClient using the AWS SDK DynamoDb client
        //    created in Step 8, and create a table with your modeled class.
        final DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();
        final DynamoDbTable<SimpleClass> table = enhancedClient.table(ddbTableName,
 tableSchema);
    }
}
```

Langkah 2. Tulis item dalam format baru

Setelah Anda menerapkan perubahan dari Langkah 1 ke semua pembaca, selesaikan langkah-langkah berikut untuk mengonfigurasi klien SDK Enkripsi AWS Database Anda untuk menulis item dalam format baru. Setelah Anda menerapkan perubahan berikut, klien Anda akan melanjutkan membaca item dalam format lama dan mulai menulis dan membaca item dalam format baru.

Prosedur berikut memberikan gambaran tentang langkah-langkah yang ditunjukkan dalam contoh kode di bawah ini.

1. Lanjutkan mengonfigurasi keyring, skema tabel, tindakan atribut lama**`allowedUnsignedAttributes`**, dan **`DynamoDBEncryptor`** seperti yang Anda lakukan di Langkah 1.

2. Perbarui perilaku lama Anda untuk hanya menulis item baru menggunakan format baru.
3. Buat `DynamoDbEncryptionInterceptor`
4. Buat klien AWS SDK DynamoDB baru.
5. Buat `DynamoDBEnhancedClient` dan buat tabel dengan kelas model Anda.

   Untuk informasi selengkapnya tentang DynamoDB Enhanced Client, lihat membuat klien yang disempurnakan.

```
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);
    }
}
```

Setelah menerapkan perubahan Langkah 2, Anda harus mengenkripsi ulang semua item lama di tabel Anda dengan format baru sebelum Anda dapat melanjutkan ke Langkah 3. Tidak ada metrik atau kueri tunggal yang dapat Anda jalankan untuk mengenkripsi item yang ada dengan cepat. Gunakan proses yang paling masuk akal untuk sistem Anda. Misalnya, Anda dapat menggunakan proses asinkron yang memindai tabel secara perlahan dan menulis ulang item menggunakan tindakan atribut baru dan konfigurasi enkripsi yang Anda tentukan.

Langkah 3. Hanya membaca dan menulis item dalam format baru

Setelah mengenkripsi ulang semua item dalam tabel Anda dengan format baru, Anda dapat menghapus perilaku lama dari konfigurasi Anda. Selesaikan langkah-langkah berikut untuk mengonfigurasi klien Anda agar hanya membaca dan menulis item dalam format baru.

Prosedur berikut memberikan gambaran tentang langkah-langkah yang ditunjukkan dalam contoh kode di bawah ini.

1. Lanjutkan mengkonfigurasi keyring Anda, skema tabel, dan **`allowedUnsignedAttributes`** seperti yang Anda lakukan di Langkah 1. Hapus tindakan atribut lama dan `DynamoDBEncryptor` dari konfigurasi Anda.
2. Buat a`DynamoDbEncryptionInterceptor`.
3. Buat klien AWS SDK DynamoDB baru.
4. Buat `DynamoDBEnhancedClient` dan buat tabel dengan kelas model Anda.

   Untuk informasi selengkapnya tentang DynamoDB Enhanced Client, lihat membuat klien yang disempurnakan.

```
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);
    }
}
```

.NET

Topik ini menjelaskan cara menginstal dan menggunakan versi 3. x dari pustaka enkripsi sisi klien .NET untuk DynamoDB. Untuk detail tentang pemrograman dengan AWS Database Encryption SDK untuk DynamoDB, lihat contoh.NET di aws-database-encryption-sdk repositori -dynamodb aktif. GitHub

Pustaka enkripsi sisi klien .NET untuk DynamoDB adalah untuk pengembang yang menulis aplikasi dalam C # dan bahasa pemrograman.NET lainnya. Hal ini didukung di Windows, macOS, dan Linux.

Semua implementasi bahasa pemrograman SDK Enkripsi AWS Database untuk DynamoDB dapat dioperasikan. Namun, SDK untuk .NET tidak mendukung nilai kosong untuk tipe data daftar atau peta. Ini berarti bahwa jika Anda menggunakan pustaka enkripsi sisi klien Java untuk DynamoDB untuk menulis item yang berisi nilai kosong untuk daftar atau tipe data peta, Anda tidak dapat mendekripsi dan membaca item tersebut menggunakan pustaka enkripsi sisi klien .NET untuk DynamoDB.

Topik

- Menginstal pustaka enkripsi sisi klien .NET untuk DynamoDB
- Debugging dengan .NET
- Menggunakan pustaka enkripsi sisi klien .NET untuk DynamoDB
- .NET contoh
- Konfigurasikan tabel DynamoDB yang ada untuk menggunakan SDK Enkripsi Database untuk AWS DynamoDB

Menginstal pustaka enkripsi sisi klien .NET untuk DynamoDB

Pustaka enkripsi sisi klien .NET untuk DynamoDB tersedia sebagai AWS.Cryptography. DbEncryptionSDK. DynamoDbpaket di NuGet. Untuk detail tentang menginstal dan membangun perpustakaan, lihat file.NET README.md di repositori -dynamodb. aws-database-encryption-sdk Pustaka enkripsi sisi klien .NET untuk DynamoDB memerlukan SDK untuk .NET bahkan jika Anda tidak menggunakan kunci (). AWS Key Management Service AWS KMS SDK untuk .NET Itu diinstal dengan NuGet paket.

Versi 3. x dari pustaka enkripsi sisi klien .NET untuk DynamoDB mendukung .NET 6.0 dan .NET Framework net48 dan yang lebih baru.

Debugging dengan .NET

Pustaka enkripsi sisi klien .NET untuk DynamoDB tidak menghasilkan log apa pun. Pengecualian di pustaka enkripsi sisi klien .NET untuk DynamoDB menghasilkan pesan pengecualian, tetapi tidak ada jejak tumpukan.

Untuk membantu Anda men-debug, pastikan untuk mengaktifkan login. SDK untuk .NET Log dan pesan kesalahan dari SDK untuk .NET dapat membantu Anda membedakan kesalahan yang timbul SDK untuk .NET dari yang ada di pustaka enkripsi sisi klien .NET untuk DynamoDB. Untuk bantuan terkait SDK untuk .NET logging, lihat AWSLoggingdi Panduan AWS SDK untuk .NET Pengembang. (Untuk melihat topiknya, perluas bagian konten Open to view .NET Framework.)

Menggunakan pustaka enkripsi sisi klien .NET untuk DynamoDB

Topik ini menjelaskan beberapa fungsi dan kelas pembantu di versi 3. x dari pustaka enkripsi sisi klien .NET untuk DynamoDB.

Untuk detail tentang pemrograman dengan pustaka enkripsi sisi klien .NET untuk DynamoDB, lihat contoh.NET di repositori -dynamodb aktif. aws-database-encryption-sdk GitHub

Topik

- Enkriptor item
- Tindakan atribut dalam SDK Enkripsi AWS Database untuk DynamoDB
- Konfigurasi enkripsi dalam SDK Enkripsi AWS Database untuk DynamoDB
- Memperbarui item dengan AWS Database Encryption SDK

Enkriptor item

Pada intinya, AWS Database Encryption SDK untuk DynamoDB adalah enkripsi item. Anda dapat menggunakan versi 3. x dari pustaka enkripsi sisi klien .NET untuk DynamoDB untuk mengenkripsi, menandatangani, memverifikasi, dan mendekripsi item tabel DynamoDB Anda dengan cara berikut.

SDK Enkripsi AWS Database tingkat rendah untuk DynamoDB API

Anda dapat menggunakan konfigurasi enkripsi tabel untuk membuat klien DynamoDB yang secara otomatis mengenkripsi dan menandatangani item sisi klien dengan permintaan DynamoDB Anda. `PutItem` Anda dapat menggunakan klien ini secara langsung, atau Anda dapat membuat model dokumen atau model persistensi objek.

Anda harus menggunakan SDK Enkripsi AWS Database tingkat rendah untuk DynamoDB API untuk menggunakan enkripsi yang dapat dicari.

Tingkat yang lebih rendah **`DynamoDbItemEncryptor`**

Tingkat yang lebih rendah `DynamoDbItemEncryptor` secara langsung mengenkripsi dan menandatangani atau mendekripsi dan memverifikasi item tabel Anda tanpa memanggil DynamoDB. Itu tidak membuat DynamoDB atau `PutItem` permintaan`GetItem`. Misalnya, Anda dapat menggunakan level yang lebih rendah `DynamoDbItemEncryptor` untuk langsung mendekripsi dan memverifikasi item DynamoDB yang telah Anda ambil. Jika Anda menggunakan tingkat yang lebih rendah`DynamoDbItemEncryptor`, sebaiknya gunakan model pemrograman tingkat rendah yang SDK untuk .NET disediakan untuk berkomunikasi dengan DynamoDB.

Tingkat yang lebih rendah `DynamoDbItemEncryptor` tidak mendukung enkripsi yang dapat dicari.

Tindakan atribut dalam SDK Enkripsi AWS Database untuk DynamoDB

Tindakan atribut menentukan nilai atribut mana yang dienkripsi dan ditandatangani, yang hanya ditandatangani, yang ditandatangani dan disertakan dalam konteks enkripsi, dan mana yang diabaikan.

Untuk menentukan tindakan atribut dengan klien.NET, tentukan tindakan atribut secara manual menggunakan model objek. Tentukan tindakan atribut Anda dengan membuat `Dictionary` objek di mana pasangan nama-nilai mewakili nama atribut dan tindakan yang ditentukan.

Tentukan `ENCRYPT_AND_SIGN` untuk mengenkripsi dan menandatangani atribut.
Tentukan `SIGN_ONLY` untuk menandatangani, tetapi tidak mengenkripsi, atribut. Tentukan

`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` untuk menandatangani atribut dan sertakan dalam konteks enkripsi. Anda tidak dapat mengenkripsi atribut tanpa menandatanganinya juga. Tentukan `DO_NOTHING` untuk mengabaikan atribut.

Partisi dan atribut sortir harus salah satu `SIGN_ONLY` atau`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Jika Anda mendefinisikan atribut apa pun sebagai`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, maka atribut partisi dan sortir juga harus`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

> **Note**
>
> Setelah menentukan tindakan atribut, Anda harus menentukan atribut mana yang dikecualikan dari tanda tangan. Untuk mempermudah menambahkan atribut unsigned baru di masa mendatang, sebaiknya pilih awalan yang berbeda (seperti "`:`“) untuk mengidentifikasi atribut unsigned Anda. Sertakan awalan ini dalam nama atribut untuk semua atribut yang ditandai `DO_NOTHING` saat Anda menentukan skema DynamoDB dan tindakan atribut.

Model objek berikut menunjukkan cara menentukan`ENCRYPT_AND_SIGN`,, `SIGN_ONLY`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT, dan `DO_NOTHING` atribut tindakan dengan klien.NET. Contoh ini menggunakan awalan "`:`" untuk mengidentifikasi `DO_NOTHING` atribut.

> **Note**
>
> Untuk menggunakan tindakan `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` kriptografi, Anda harus menggunakan SDK Enkripsi AWS Database versi 3.3 atau yang lebih baru. Terapkan versi baru ke semua pembaca sebelum memperbarui model data Anda untuk disertakan`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

```
var attributeActionsOnEncrypt = new Dictionary<string, CryptoAction>
{
    ["partition_key"] = CryptoAction.SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT, // The
 partition attribute must be signed
    ["sort_key"] = CryptoAction.SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT, // The sort
 attribute must be signed
    ["attribute1"] = CryptoAction.ENCRYPT_AND_SIGN,
    ["attribute2"] = CryptoAction.SIGN_ONLY,
    ["attribute3"] = CryptoAction.SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT,
```

```
    [":attribute4"] = CryptoAction.DO_NOTHING
};
```

Konfigurasi enkripsi dalam SDK Enkripsi AWS Database untuk DynamoDB

Bila Anda menggunakan AWS Database Encryption SDK, Anda harus secara eksplisit menentukan konfigurasi enkripsi untuk tabel DynamoDB Anda. Nilai yang diperlukan dalam konfigurasi enkripsi Anda bergantung pada apakah Anda mendefinisikan tindakan atribut secara manual atau dengan kelas data beranotasi.

Cuplikan berikut mendefinisikan konfigurasi enkripsi tabel DynamoDB menggunakan SDK AWS Enkripsi Database tingkat rendah untuk DynamoDB API dan mengizinkan atribut unsigned yang ditentukan oleh awalan yang berbeda.

```
Dictionary<String, DynamoDbTableEncryptionConfig> tableConfigs =
    new Dictionary<String, DynamoDbTableEncryptionConfig>();
DynamoDbTableEncryptionConfig config = new DynamoDbTableEncryptionConfig
{
    LogicalTableName = ddbTableName,
    PartitionKeyName = "partition_key",
    SortKeyName = "sort_key",
    AttributeActionsOnEncrypt = attributeActionsOnEncrypt,
    Keyring = kmsKeyring,
    AllowedUnsignedAttributePrefix = unsignAttrPrefix,
    // Optional: SearchConfig only required if you use beacons
    Search = new SearchConfig
    {
        WriteVersion = 1, // MUST be 1
        Versions = beaconVersions
    }
};
tableConfigs.Add(ddbTableName, config);
```

Nama tabel logis

Sebuah nama tabel logis untuk tabel DynamoDB Anda.

Nama tabel logis terikat secara kriptografis ke semua data yang disimpan dalam tabel untuk menyederhanakan operasi pemulihan DynamoDB. Kami sangat menyarankan untuk menentukan nama tabel DynamoDB Anda sebagai nama tabel logis saat Anda pertama kali menentukan konfigurasi enkripsi Anda. Anda harus selalu menentukan nama tabel logis yang sama. Agar

dekripsi berhasil, nama tabel logis harus sesuai dengan nama yang ditentukan pada enkripsi. Jika nama tabel DynamoDB Anda berubah setelah memulihkan tabel DynamoDB Anda dari cadangan, nama tabel logis memastikan bahwa operasi dekripsi masih mengenali tabel.

Atribut yang tidak ditandatangani yang diizinkan

Atribut yang ditandai `DO_NOTHING` dalam tindakan atribut Anda.

Atribut unsigned yang diizinkan memberi tahu klien atribut mana yang dikecualikan dari tanda tangan. Klien mengasumsikan bahwa semua atribut lainnya termasuk dalam tanda tangan. Kemudian, saat mendekripsi catatan, klien menentukan atribut mana yang perlu diverifikasi dan mana yang harus diabaikan dari atribut unsigned yang diizinkan yang Anda tentukan. Anda tidak dapat menghapus atribut dari atribut yang tidak ditandatangani yang diizinkan.

Anda dapat menentukan atribut unsigned yang diizinkan secara eksplisit dengan membuat array yang mencantumkan semua atribut Anda. `DO_NOTHING` Anda juga dapat menentukan awalan yang berbeda saat menamai `DO_NOTHING` atribut Anda dan menggunakan awalan untuk memberi tahu klien atribut mana yang tidak ditandatangani. Kami sangat menyarankan untuk menentukan awalan yang berbeda karena menyederhanakan proses penambahan `DO_NOTHING` atribut baru di masa depan. Untuk informasi selengkapnya, lihat Memperbarui model data Anda.

Jika Anda tidak menentukan awalan untuk semua `DO_NOTHING` atribut, Anda dapat mengonfigurasi `allowedUnsignedAttributes` array yang secara eksplisit mencantumkan semua atribut yang diharapkan klien tidak ditandatangani saat bertemu dengan mereka pada dekripsi. Anda hanya harus secara eksplisit mendefinisikan atribut unsigned yang diizinkan jika benar-benar diperlukan.

Konfigurasi Pencarian (Opsional)

`SearchConfig`Mendefinisikan versi beacon.

`SearchConfig`Harus ditentukan untuk menggunakan enkripsi yang dapat dicari atau suar yang ditandatangani.

Suite Algoritma (Opsional)

`algorithmSuiteId`Mendefinisikan algoritma mana yang sesuai dengan AWS Database Encryption SDK yang digunakan.

Kecuali Anda secara eksplisit menentukan rangkaian algoritme alternatif, SDK Enkripsi AWS Database menggunakan rangkaian algoritme default. Rangkaian algoritme default menggunakan

algoritma AES-GCM dengan derivasi kunci, tanda tangan digital, dan komitmen kunci. Meskipun rangkaian algoritme default kemungkinan cocok untuk sebagian besar aplikasi, Anda dapat memilih rangkaian algoritme alternatif. Misalnya, beberapa model kepercayaan akan dipenuhi oleh rangkaian algoritma tanpa tanda tangan digital. Untuk informasi tentang rangkaian algoritme yang didukung SDK Enkripsi AWS Database, lihatRangkaian algoritme yang didukung di SDK Enkripsi AWS Database.

Untuk memilih rangkaian algoritma AES-GCM tanpa tanda tangan digital ECDSA, sertakan cuplikan berikut dalam konfigurasi enkripsi tabel Anda.

```
AlgorithmSuiteId =
 DBEAlgorithmSuiteId.ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY_SYMSIG_HMAC_SHA384
```

Memperbarui item dengan AWS Database Encryption SDK

SDK Enkripsi AWS Database tidak mendukung ddb: UpdateItem untuk item yang menyertakan atribut terenkripsi atau ditandatangani. Untuk memperbarui atribut terenkripsi atau ditandatangani, Anda harus menggunakan ddb:. PutItem Saat Anda menentukan kunci utama yang sama dengan item yang ada dalam `PutItem` permintaan Anda, item baru sepenuhnya menggantikan item yang ada. Anda juga dapat menggunakan CLOBBER untuk menghapus dan mengganti semua atribut yang disimpan setelah memperbarui item Anda.

.NET contoh

Contoh berikut menunjukkan cara menggunakan pustaka enkripsi sisi klien .NET untuk DynamoDB untuk melindungi item tabel dalam aplikasi Anda. Untuk menemukan lebih banyak contoh (dan berkontribusi sendiri), lihat contoh.NET di repositori aws-database-encryption-sdk -dynamodb di. GitHub

Contoh berikut menunjukkan cara mengonfigurasi pustaka enkripsi sisi klien .NET untuk DynamoDB dalam tabel Amazon DynamoDB baru yang tidak terisi. Jika Anda ingin mengonfigurasi tabel Amazon DynamoDB yang ada untuk enkripsi sisi klien, lihat. Tambahkan versi 3.x ke tabel yang ada

Topik

- Menggunakan SDK Enkripsi AWS Database tingkat rendah untuk DynamoDB API
- Menggunakan level yang lebih rendah DynamoDbItemEncryptor

Menggunakan SDK Enkripsi AWS Database tingkat rendah untuk DynamoDB API

Contoh berikut menunjukkan cara menggunakan SDK Enkripsi AWS Database tingkat rendah untuk DynamoDB API dengan AWS KMS keyring untuk secara otomatis mengenkripsi dan menandatangani item sisi klien dengan permintaan DynamoDB Anda. `PutItem`

Anda dapat menggunakan keyring apa pun yang didukung, tetapi kami sarankan menggunakan salah satu AWS KMS gantungan kunci bila memungkinkan.

Lihat contoh kode lengkapnya: BasicPutGetExample.cs

Langkah 1: Buat AWS KMS keyring

Contoh berikut digunakan `CreateAwsKmsMrkMultiKeyring` untuk membuat AWS KMS keyring dengan kunci KMS enkripsi simetris. `CreateAwsKmsMrkMultiKeyring`Metode ini memastikan bahwa keyring akan menangani tombol Single-region dan Multi-region dengan benar.

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var keyringInput = new CreateAwsKmsMrkMultiKeyringInput { Generator = kmsKeyId };
var kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

Langkah 2: Konfigurasikan tindakan atribut Anda

Contoh berikut mendefinisikan `attributeActionsOnEncrypt` Kamus yang mewakili tindakan atribut sampel untuk item tabel.

> **Note**
>
> Contoh berikut tidak mendefinisikan atribut apa pun sebagai`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Jika Anda menentukan `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` atribut apa pun, maka atribut partisi dan sortir juga harus`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

```
var attributeActionsOnEncrypt = new Dictionary<string, CryptoAction>
{
    ["partition_key"] = CryptoAction.SIGN_ONLY, // The partition attribute must be
 SIGN_ONLY
    ["sort_key"] = CryptoAction.SIGN_ONLY, // The sort attribute must be SIGN_ONLY
    ["attribute1"] = CryptoAction.ENCRYPT_AND_SIGN,
```

```
    ["attribute2"] = CryptoAction.SIGN_ONLY,
    [":attribute3"] = CryptoAction.DO_NOTHING
};
```

Langkah 3: Tentukan atribut mana yang dikecualikan dari tanda tangan

Contoh berikut mengasumsikan bahwa semua `DO_NOTHING` atribut berbagi awalan yang berbeda "`:`“, dan menggunakan awalan untuk menentukan atribut unsigned yang diizinkan. Klien mengasumsikan bahwa nama atribut apa pun dengan awalan "`:`" dikecualikan dari tanda tangan. Untuk informasi selengkapnya, lihat Allowed unsigned attributes.

```
const String unsignAttrPrefix = ":";
```

Langkah 4: Tentukan konfigurasi enkripsi tabel DynamoDB

Contoh berikut mendefinisikan `tableConfigs` Peta yang mewakili konfigurasi enkripsi untuk tabel DynamoDB ini.

Contoh ini menentukan nama tabel DynamoDB sebagai nama tabel logis. Kami sangat menyarankan untuk menentukan nama tabel DynamoDB Anda sebagai nama tabel logis saat Anda pertama kali menentukan konfigurasi enkripsi Anda. Untuk informasi selengkapnya, lihat Konfigurasi enkripsi dalam SDK Enkripsi AWS Database untuk DynamoDB.

> **Note**
>
> Untuk menggunakan enkripsi yang dapat dicari atau suar yang ditandatangani, Anda juga harus menyertakan `SearchConfig`dalam konfigurasi enkripsi Anda.

```
Dictionary<String, DynamoDbTableEncryptionConfig> tableConfigs =
    new Dictionary<String, DynamoDbTableEncryptionConfig>();
DynamoDbTableEncryptionConfig config = new DynamoDbTableEncryptionConfig
{
    LogicalTableName = ddbTableName,
    PartitionKeyName = "partition_key",
    SortKeyName = "sort_key",
    AttributeActionsOnEncrypt = attributeActionsOnEncrypt,
    Keyring = kmsKeyring,
    AllowedUnsignedAttributePrefix = unsignAttrPrefix
};
```

```
tableConfigs.Add(ddbTableName, config);
```

Langkah 5: Buat klien AWS SDK DynamoDB baru

Contoh berikut membuat klien AWS SDK DynamoDB baru menggunakan **`TableEncryptionConfigs`** dari Langkah 4.

```
var ddb = new Client.DynamoDbClient(
    new DynamoDbTablesEncryptionConfig { TableEncryptionConfigs = tableConfigs });
```

Langkah 6: Enkripsi dan tandatangani item tabel DynamoDB

Contoh berikut mendefinisikan `item` Kamus yang mewakili item tabel sampel dan menempatkan item dalam tabel DynamoDB. Item dienkripsi dan ditandatangani sisi klien sebelum dikirim ke DynamoDB.

```
var item = new Dictionary<String, AttributeValue>
{
    ["partition_key"] = new AttributeValue("BasicPutGetExample"),
    ["sort_key"] = new AttributeValue { N = "0" },
    ["attribute1"] = new AttributeValue("encrypt and sign me!"),
    ["attribute2"] = new AttributeValue("sign me!"),
    [":attribute3"] = new AttributeValue("ignore me!")
};

PutItemRequest putRequest = new PutItemRequest
{
    TableName = ddbTableName,
    Item = item
};

PutItemResponse putResponse = await ddb.PutItemAsync(putRequest);
```

Menggunakan level yang lebih rendah **`DynamoDbItemEncryptor`**

Contoh berikut menunjukkan cara menggunakan level yang lebih rendah `DynamoDbItemEncryptor` dengan AWS KMS keyring untuk langsung mengenkripsi dan menandatangani item tabel. `DynamoDbItemEncryptor`Itu tidak menempatkan item di tabel DynamoDB Anda.

Anda dapat menggunakan keyring apa pun yang didukung dengan DynamoDB Enhanced Client, tetapi sebaiknya gunakan salah AWS KMS satu gantungan kunci bila memungkinkan.

> **Note**
>
> Tingkat yang lebih rendah `DynamoDbItemEncryptor` tidak mendukung enkripsi yang dapat dicari. Gunakan SDK Enkripsi AWS Database tingkat rendah untuk DynamoDB API untuk menggunakan enkripsi yang dapat dicari.

Lihat contoh kode lengkapnya: ItemEncryptDecryptExample.cs

Langkah 1: Buat AWS KMS keyring

Contoh berikut digunakan `CreateAwsKmsMrkMultiKeyring` untuk membuat AWS KMS keyring dengan kunci KMS enkripsi simetris. `CreateAwsKmsMrkMultiKeyring`Metode ini memastikan bahwa keyring akan menangani tombol Single-region dan Multi-region dengan benar.

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var keyringInput = new CreateAwsKmsMrkMultiKeyringInput { Generator = kmsKeyId };
var kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

Langkah 2: Konfigurasikan tindakan atribut Anda

Contoh berikut mendefinisikan `attributeActionsOnEncrypt` Kamus yang mewakili tindakan atribut sampel untuk item tabel.

> **Note**
>
> Contoh berikut tidak mendefinisikan atribut apa pun sebagai`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Jika Anda menentukan `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` atribut apa pun, maka atribut partisi dan sortir juga harus`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

```
var attributeActionsOnEncrypt = new Dictionary<String, CryptoAction>
{
    ["partition_key"] = CryptoAction.SIGN_ONLY, // The partition attribute must be
 SIGN_ONLY
    ["sort_key"] = CryptoAction.SIGN_ONLY, // The sort attribute must be SIGN_ONLY
    ["attribute1"] = CryptoAction.ENCRYPT_AND_SIGN,
    ["attribute2"] = CryptoAction.SIGN_ONLY,
```

```
    [":attribute3"] = CryptoAction.DO_NOTHING
};
```

Langkah 3: Tentukan atribut mana yang dikecualikan dari tanda tangan

Contoh berikut mengasumsikan bahwa semua `DO_NOTHING` atribut berbagi awalan yang berbeda "**:**“, dan menggunakan awalan untuk menentukan atribut unsigned yang diizinkan. Klien mengasumsikan bahwa nama atribut apa pun dengan awalan "**:**" dikecualikan dari tanda tangan. Untuk informasi selengkapnya, lihat Allowed unsigned attributes.

```
String unsignAttrPrefix = ":";
```

Langkah 4: Tentukan **`DynamoDbItemEncryptor`** konfigurasi

Contoh berikut mendefinisikan konfigurasi untuk. `DynamoDbItemEncryptor`

Contoh ini menentukan nama tabel DynamoDB sebagai nama tabel logis. Kami sangat menyarankan untuk menentukan nama tabel DynamoDB Anda sebagai nama tabel logis saat Anda pertama kali menentukan konfigurasi enkripsi Anda. Untuk informasi selengkapnya, lihat Konfigurasi enkripsi dalam SDK Enkripsi AWS Database untuk DynamoDB.

```
var config = new DynamoDbItemEncryptorConfig
{
    LogicalTableName = ddbTableName,
    PartitionKeyName = "partition_key",
    SortKeyName = "sort_key",
    AttributeActionsOnEncrypt = attributeActionsOnEncrypt,
    Keyring = kmsKeyring,
    AllowedUnsignedAttributePrefix = unsignAttrPrefix
};
```

Langkah 5: Buat **`DynamoDbItemEncryptor`**

Contoh berikut membuat baru `DynamoDbItemEncryptor` menggunakan `config` dari Langkah 4.

```
var itemEncryptor = new DynamoDbItemEncryptor(config);
```

Langkah 6: Langsung mengenkripsi dan menandatangani item tabel

Contoh berikut langsung mengenkripsi dan menandatangani item menggunakan. `DynamoDbItemEncryptor` `DynamoDbItemEncryptor`Itu tidak menempatkan item di tabel DynamoDB Anda.

```
var originalItem = new Dictionary<String, AttributeValue>
{
    ["partition_key"] = new AttributeValue("ItemEncryptDecryptExample"),
    ["sort_key"] = new AttributeValue { N = "0" },
    ["attribute1"] = new AttributeValue("encrypt and sign me!"),
    ["attribute2"] = new AttributeValue("sign me!"),
    [":attribute3"] = new AttributeValue("ignore me!")
};

var encryptedItem = itemEncryptor.EncryptItem(
    new EncryptItemInput { PlaintextItem = originalItem }
).EncryptedItem;
```

Konfigurasikan tabel DynamoDB yang ada untuk menggunakan SDK Enkripsi Database untuk AWS DynamoDB

Dengan versi 3. x dari pustaka enkripsi sisi klien .NET untuk DynamoDB, Anda dapat mengonfigurasi tabel Amazon DynamoDB yang ada untuk enkripsi sisi klien. Topik ini memberikan panduan tentang tiga langkah yang harus Anda ambil untuk menambahkan versi 3. x ke tabel DynamoDB yang sudah ada dan terisi.

Langkah 1: Bersiaplah untuk membaca dan menulis item terenkripsi

Selesaikan langkah-langkah berikut untuk mempersiapkan klien SDK Enkripsi AWS Database Anda untuk membaca dan menulis item terenkripsi. Setelah Anda menerapkan perubahan berikut, klien Anda akan terus membaca dan menulis item teks biasa. Ini tidak akan mengenkripsi atau menandatangani item baru yang ditulis ke tabel, tetapi akan dapat mendekripsi item terenkripsi segera setelah muncul. Perubahan ini mempersiapkan klien untuk mulai mengenkripsi item baru. Perubahan berikut harus diterapkan ke setiap pembaca sebelum Anda melanjutkan ke langkah berikutnya.

1. Tentukan tindakan atribut Anda

   Buat model objek untuk menentukan nilai atribut mana yang akan dienkripsi dan ditandatangani, yang hanya akan ditandatangani, dan mana yang akan diabaikan.

   Secara default, atribut kunci primer ditandatangani tetapi tidak dienkripsi (`SIGN_ONLY`) dan semua atribut lainnya dienkripsi dan ditandatangani (). `ENCRYPT_AND_SIGN`

   Tentukan `ENCRYPT_AND_SIGN` untuk mengenkripsi dan menandatangani atribut. Tentukan `SIGN_ONLY` untuk menandatangani, tetapi tidak mengenkripsi, atribut. Tentukan `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` untuk menandatangani dan atribut dan sertakan dalam konteks enkripsi. Anda tidak dapat mengenkripsi atribut tanpa menandatanganinya juga. Tentukan `DO_NOTHING` untuk mengabaikan atribut. Untuk informasi selengkapnya, lihat Tindakan atribut dalam SDK Enkripsi AWS Database untuk DynamoDB.

   > **Note**
   >
   > Jika Anda menentukan `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` atribut apa pun, maka atribut partisi dan sortir juga harus`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

   ```
   var attributeActionsOnEncrypt = new Dictionary<string, CryptoAction>
   {
       ["partition_key"] = CryptoAction.SIGN_ONLY, // The partition attribute must be
    SIGN_ONLY
       ["sort_key"] = CryptoAction.SIGN_ONLY, // The sort attribute must be SIGN_ONLY
       ["attribute1"] = CryptoAction.ENCRYPT_AND_SIGN,
       ["attribute2"] = CryptoAction.SIGN_ONLY,
       [":attribute3"] = CryptoAction.DO_NOTHING
   };
   ```

2. Tentukan atribut mana yang akan dikecualikan dari tanda tangan

   Contoh berikut mengasumsikan bahwa semua `DO_NOTHING` atribut berbagi awalan yang berbeda "`:`“, dan menggunakan awalan untuk menentukan atribut unsigned yang diizinkan. Klien akan menganggap bahwa nama atribut apa pun dengan awalan `:` "" dikecualikan dari tanda tangan. Untuk informasi selengkapnya, lihat Allowed unsigned attributes.

   ```
   const String unsignAttrPrefix = ":";
   ```

3. Buat keyring

   Contoh berikut membuat AWS KMS keyring. AWS KMS Keyring menggunakan enkripsi simetris atau RSA asimetris AWS KMS keys untuk menghasilkan, mengenkripsi, dan mendekripsi kunci data.

   Contoh ini digunakan `CreateMrkMultiKeyring` untuk membuat AWS KMS keyring dengan kunci KMS enkripsi simetris. `CreateAwsKmsMrkMultiKeyring`Metode ini memastikan bahwa keyring akan menangani tombol Single-region dan Multi-region dengan benar.

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var keyringInput = new CreateAwsKmsMrkMultiKeyringInput { Generator = kmsKeyId };
var kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

4. Tentukan konfigurasi enkripsi tabel DynamoDB

   Contoh berikut mendefinisikan `tableConfigs` Peta yang mewakili konfigurasi enkripsi untuk tabel DynamoDB ini.

   Contoh ini menentukan nama tabel DynamoDB sebagai nama tabel logis. Kami sangat menyarankan untuk menentukan nama tabel DynamoDB Anda sebagai nama tabel logis saat Anda pertama kali menentukan konfigurasi enkripsi Anda.

   Anda harus menentukan `FORCE_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT` sebagai penggantian plaintext. Kebijakan ini terus membaca dan menulis item teks biasa, membaca item terenkripsi, dan mempersiapkan klien untuk menulis item terenkripsi.

   Untuk informasi selengkapnya tentang nilai yang disertakan dalam konfigurasi enkripsi tabel, lihatKonfigurasi enkripsi dalam SDK Enkripsi AWS Database untuk DynamoDB.

```
Dictionary<String, DynamoDbTableEncryptionConfig> tableConfigs =
    new Dictionary<String, DynamoDbTableEncryptionConfig>();
DynamoDbTableEncryptionConfig config = new DynamoDbTableEncryptionConfig
{
    LogicalTableName = ddbTableName,
    PartitionKeyName = "partition_key",
    SortKeyName = "sort_key",
    AttributeActionsOnEncrypt = attributeActionsOnEncrypt,
    Keyring = kmsKeyring,
    AllowedUnsignedAttributePrefix = unsignAttrPrefix,
    PlaintextOverride = FORCE_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT
};
```

```
tableConfigs.Add(ddbTableName, config);
```

5. Buat klien AWS SDK DynamoDB baru

   contoh berikut membuat klien AWS SDK DynamoDB baru menggunakan **`TableEncryptionConfigs`** dari Langkah 4.

```
var ddb = new Client.DynamoDbClient(
    new DynamoDbTablesEncryptionConfig { TableEncryptionConfigs = tableConfigs });
```

Langkah 2: Tulis item terenkripsi dan ditandatangani

Perbarui kebijakan plaintext dalam konfigurasi enkripsi tabel Anda untuk memungkinkan klien menulis item terenkripsi dan ditandatangani. Setelah Anda menerapkan perubahan berikut, klien akan mengenkripsi dan menandatangani item baru berdasarkan tindakan atribut yang Anda konfigurasikan di Langkah 1. Klien akan dapat membaca item teks biasa dan item terenkripsi dan ditandatangani.

Sebelum Anda melanjutkan ke Langkah 3, Anda harus mengenkripsi dan menandatangani semua item plaintext yang ada di tabel Anda. Tidak ada metrik atau kueri tunggal yang dapat Anda jalankan untuk mengenkripsi item plaintext yang ada dengan cepat. Gunakan proses yang paling masuk akal untuk sistem Anda. Misalnya, Anda dapat menggunakan proses asinkron yang memindai tabel secara perlahan dan menulis ulang item menggunakan tindakan atribut dan konfigurasi enkripsi yang Anda tentukan. Untuk mengidentifikasi item plaintext dalam tabel Anda, kami sarankan memindai semua item yang tidak berisi `aws_dbe_head` dan `aws_dbe_foot` atribut yang ditambahkan SDK Enkripsi AWS Database ke item saat dienkripsi dan ditandatangani.

Contoh berikut memperbarui konfigurasi enkripsi tabel dari Langkah 1. Anda harus memperbarui penggantian plaintext dengan. `FORBID_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT` Kebijakan ini terus membaca item teks biasa, tetapi juga membaca dan menulis item terenkripsi. Buat klien AWS SDK DynamoDB baru menggunakan yang diperbarui. `TableEncryptionConfigs`

```
Dictionary<String, DynamoDbTableEncryptionConfig> tableConfigs =
    new Dictionary<String, DynamoDbTableEncryptionConfig>();
DynamoDbTableEncryptionConfig config = new DynamoDbTableEncryptionConfig
{
    LogicalTableName = ddbTableName,
    PartitionKeyName = "partition_key",
    SortKeyName = "sort_key",
    AttributeActionsOnEncrypt = attributeActionsOnEncrypt,
```

```
    Keyring = kmsKeyring,
    AllowedUnsignedAttributePrefix = unsignAttrPrefix,
    PlaintextOverride = FORBID_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT
};
tableConfigs.Add(ddbTableName, config);
```

Langkah 3: Hanya baca item terenkripsi dan ditandatangani

Setelah Anda mengenkripsi dan menandatangani semua item Anda, perbarui penggantian plaintext dalam konfigurasi enkripsi tabel Anda untuk hanya mengizinkan klien membaca dan menulis item terenkripsi dan ditandatangani. Setelah Anda menerapkan perubahan berikut, klien akan mengenkripsi dan menandatangani item baru berdasarkan tindakan atribut yang Anda konfigurasikan di Langkah 1. Klien hanya akan dapat membaca item terenkripsi dan ditandatangani.

Contoh berikut memperbarui konfigurasi enkripsi tabel dari Langkah 2. Anda dapat memperbarui penggantian plaintext dengan `FORBID_WRITE_PLAINTEXT_FORBID_READ_PLAINTEXT` atau menghapus kebijakan plaintext dari konfigurasi Anda. Klien hanya membaca dan menulis item terenkripsi dan ditandatangani secara default. Buat klien AWS SDK DynamoDB baru menggunakan yang diperbarui. `TableEncryptionConfigs`

```
Dictionary<String, DynamoDbTableEncryptionConfig> tableConfigs =
    new Dictionary<String, DynamoDbTableEncryptionConfig>();
DynamoDbTableEncryptionConfig config = new DynamoDbTableEncryptionConfig
{
    LogicalTableName = ddbTableName,
    PartitionKeyName = "partition_key",
    SortKeyName = "sort_key",
    AttributeActionsOnEncrypt = attributeActionsOnEncrypt,
    Keyring = kmsKeyring,
    AllowedUnsignedAttributePrefix = unsignAttrPrefix,
    // Optional: you can also remove the plaintext policy from your configuration
    PlaintextOverride = FORBID_WRITE_PLAINTEXT_FORBID_READ_PLAINTEXT
};
tableConfigs.Add(ddbTableName, config);
```

Rust

Topik ini menjelaskan cara menginstal dan menggunakan versi 1. x dari pustaka enkripsi sisi klien Rust untuk DynamoDB. Untuk detail tentang pemrograman dengan AWS Database Encryption SDK untuk DynamoDB, lihat contoh Rust di aws-database-encryption-sdk repositori -dynamodb aktif. GitHub

Semua implementasi bahasa pemrograman SDK Enkripsi AWS Database untuk DynamoDB dapat dioperasikan.

Topik

- Prasyarat
- Penginstalan
- Menggunakan pustaka enkripsi sisi klien Rust untuk DynamoDB

Prasyarat

Sebelum Anda menginstal pustaka enkripsi sisi klien Rust untuk DynamoDB, pastikan Anda memiliki prasyarat berikut.

Instal Rust dan Cargo

Instal rilis stabil Rust saat ini menggunakan rustup.

Untuk informasi lebih lanjut tentang mengunduh dan menginstal rustup, lihat prosedur instalasi di The Cargo Book.

Penginstalan

Pustaka enkripsi sisi klien Rust untuk DynamoDB tersedia sebagai peti di Crates.io. aws-db-esdk Untuk detail tentang menginstal dan membangun perpustakaan, lihat file README.md di repositori -dynamodb. aws-database-encryption-sdk GitHub

Secara manual

Untuk menginstal pustaka enkripsi sisi klien Rust untuk DynamoDB, kloning atau unduh repositori -dynamodb. aws-database-encryption-sdk GitHub

Pasang versi terbaru

Jalankan perintah Cargo berikut di direktori proyek Anda:

```
cargo add aws-db-esdk
```

Atau tambahkan baris berikut ke Cargo.toml Anda:

```
aws-db-esdk = "<version>"
```

Menggunakan pustaka enkripsi sisi klien Rust untuk DynamoDB

Topik ini menjelaskan beberapa fungsi dan kelas pembantu di versi 1. x dari pustaka enkripsi sisi klien Rust untuk DynamoDB.

Untuk detail tentang pemrograman dengan pustaka enkripsi sisi klien Rust untuk DynamoDB, lihat contoh Rust di repositori -dynamodb aktif. aws-database-encryption-sdk GitHub

Topik

- Enkriptor item
- Tindakan atribut dalam SDK Enkripsi AWS Database untuk DynamoDB
- Konfigurasi enkripsi dalam SDK Enkripsi AWS Database untuk DynamoDB
- Memperbarui item dengan SDK Enkripsi AWS Database

Enkriptor item

Pada intinya, AWS Database Encryption SDK untuk DynamoDB adalah enkripsi item. Anda dapat menggunakan versi 1. x dari pustaka enkripsi sisi klien Rust untuk DynamoDB untuk mengenkripsi, menandatangani, memverifikasi, dan mendekripsi item tabel DynamoDB Anda dengan cara berikut.

SDK Enkripsi AWS Database tingkat rendah untuk DynamoDB API

Anda dapat menggunakan konfigurasi enkripsi tabel untuk membuat klien DynamoDB yang secara otomatis mengenkripsi dan menandatangani item sisi klien dengan permintaan DynamoDB Anda. `PutItem`

Anda harus menggunakan SDK Enkripsi AWS Database tingkat rendah untuk DynamoDB API untuk menggunakan enkripsi yang dapat dicari.

Untuk contoh yang mendemonstrasikan cara menggunakan SDK Enkripsi AWS Database tingkat rendah untuk DynamoDB API, lihat basic_get_put_example.rs di repositori -dynamodb pada. aws-database-encryption-sdk GitHub

Tingkat yang lebih rendah **`DynamoDbItemEncryptor`**

Tingkat yang lebih rendah `DynamoDbItemEncryptor` secara langsung mengenkripsi dan menandatangani atau mendekripsi dan memverifikasi item tabel Anda tanpa memanggil

DynamoDB. Itu tidak membuat DynamoDB atau `PutItem` permintaan`GetItem`. Misalnya, Anda dapat menggunakan level yang lebih rendah `DynamoDbItemEncryptor` untuk langsung mendekripsi dan memverifikasi item DynamoDB yang telah Anda ambil.

Tingkat yang lebih rendah `DynamoDbItemEncryptor` tidak mendukung enkripsi yang dapat dicari.

Untuk contoh yang menunjukkan cara menggunakan level yang lebih rendah, lihat `DynamoDbItemEncryptor` item_encrypt_decrypt.rs di repositori -dynamodb pada. aws-database-encryption-sdk GitHub

Tindakan atribut dalam SDK Enkripsi AWS Database untuk DynamoDB

Tindakan atribut menentukan nilai atribut mana yang dienkripsi dan ditandatangani, yang hanya ditandatangani, yang ditandatangani dan disertakan dalam konteks enkripsi, dan mana yang diabaikan.

Untuk menentukan tindakan atribut dengan klien Rust, tentukan tindakan atribut secara manual menggunakan model objek. Tentukan tindakan atribut Anda dengan membuat `HashMap` objek di mana pasangan nama-nilai mewakili nama atribut dan tindakan yang ditentukan.

Tentukan `ENCRYPT_AND_SIGN` untuk mengenkripsi dan menandatangani atribut. Tentukan `SIGN_ONLY` untuk menandatangani, tetapi tidak mengenkripsi, atribut. Tentukan `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` untuk menandatangani atribut dan sertakan dalam konteks enkripsi. Anda tidak dapat mengenkripsi atribut tanpa menandatanganinya juga. Tentukan `DO_NOTHING` untuk mengabaikan atribut.

Partisi dan atribut sortir harus salah satu `SIGN_ONLY` atau`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Jika Anda mendefinisikan atribut apa pun sebagai`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, maka atribut partisi dan sortir juga harus`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

> **Note**
>
> Setelah menentukan tindakan atribut, Anda harus menentukan atribut mana yang dikecualikan dari tanda tangan. Untuk mempermudah menambahkan atribut baru yang tidak ditandatangani di masa mendatang, sebaiknya pilih awalan yang berbeda (seperti `":`“) untuk mengidentifikasi atribut unsigned Anda. Sertakan awalan ini dalam nama atribut untuk semua

atribut yang ditandai `DO_NOTHING` saat Anda menentukan skema DynamoDB dan tindakan atribut.

Model objek berikut menunjukkan cara menentukan`ENCRYPT_AND_SIGN`,, `SIGN_ONLY`, `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, dan `DO_NOTHING` atribut tindakan dengan klien Rust. Contoh ini menggunakan awalan "`:`" untuk mengidentifikasi `DO_NOTHING` atribut.

```
let attribute_actions_on_encrypt = HashMap::from([
    ("partition_key".to_string(), CryptoAction::SignOnly),
    ("sort_key".to_string(), CryptoAction::SignOnly),
    ("attribute1".to_string(), CryptoAction::EncryptAndSign),
    ("attribute2".to_string(), CryptoAction::SignOnly),
    (":attribute3".to_string(), CryptoAction::DoNothing),
]);
```

Konfigurasi enkripsi dalam SDK Enkripsi AWS Database untuk DynamoDB

Bila Anda menggunakan AWS Database Encryption SDK, Anda harus secara eksplisit menentukan konfigurasi enkripsi untuk tabel DynamoDB Anda. Nilai yang diperlukan dalam konfigurasi enkripsi Anda bergantung pada apakah Anda mendefinisikan tindakan atribut secara manual atau dengan kelas data beranotasi.

Cuplikan berikut mendefinisikan konfigurasi enkripsi tabel DynamoDB menggunakan SDK AWS Enkripsi Database tingkat rendah untuk DynamoDB API dan mengizinkan atribut unsigned yang ditentukan oleh awalan yang berbeda.

```
let table_config = DynamoDbTableEncryptionConfig::builder()
    .logical_table_name(ddb_table_name)
    .partition_key_name("partition_key")
    .sort_key_name("sort_key")
    .attribute_actions_on_encrypt(attribute_actions_on_encrypt)
    .keyring(kms_keyring)
    .allowed_unsigned_attribute_prefix(UNSIGNED_ATTR_PREFIX)
    // Specifying an algorithm suite is optional
    .algorithm_suite_id(
        DbeAlgorithmSuiteId::AlgAes256GcmHkdfSha512CommitKeyEcdsaP384SymsigHmacSha384,
    )
    .build()?;

let table_configs = DynamoDbTablesEncryptionConfig::builder()
```

```
    .table_encryption_configs(HashMap::from([(ddb_table_name.to_string(),
table_config)]))
    .build()?;
```

Nama tabel logis

Sebuah nama tabel logis untuk tabel DynamoDB Anda.

Nama tabel logis terikat secara kriptografis ke semua data yang disimpan dalam tabel untuk menyederhanakan operasi pemulihan DynamoDB. Kami sangat menyarankan untuk menentukan nama tabel DynamoDB Anda sebagai nama tabel logis saat Anda pertama kali menentukan konfigurasi enkripsi Anda. Anda harus selalu menentukan nama tabel logis yang sama. Agar dekripsi berhasil, nama tabel logis harus sesuai dengan nama yang ditentukan pada enkripsi. Jika nama tabel DynamoDB Anda berubah setelah memulihkan tabel DynamoDB Anda dari cadangan, nama tabel logis memastikan bahwa operasi dekripsi masih mengenali tabel.

Atribut yang tidak ditandatangani yang diizinkan

Atribut yang ditandai `DO_NOTHING` dalam tindakan atribut Anda.

Atribut unsigned yang diizinkan memberi tahu klien atribut mana yang dikecualikan dari tanda tangan. Klien mengasumsikan bahwa semua atribut lainnya termasuk dalam tanda tangan. Kemudian, saat mendekripsi catatan, klien menentukan atribut mana yang perlu diverifikasi dan mana yang harus diabaikan dari atribut unsigned yang diizinkan yang Anda tentukan. Anda tidak dapat menghapus atribut dari atribut yang tidak ditandatangani yang diizinkan.

Anda dapat menentukan atribut unsigned yang diizinkan secara eksplisit dengan membuat array yang mencantumkan semua atribut Anda. `DO_NOTHING` Anda juga dapat menentukan awalan yang berbeda saat menamai `DO_NOTHING` atribut Anda dan menggunakan awalan untuk memberi tahu klien atribut mana yang tidak ditandatangani. Kami sangat menyarankan untuk menentukan awalan yang berbeda karena menyederhanakan proses penambahan `DO_NOTHING` atribut baru di masa depan. Untuk informasi selengkapnya, lihat Memperbarui model data Anda.

Jika Anda tidak menentukan awalan untuk semua `DO_NOTHING` atribut, Anda dapat mengonfigurasi `allowedUnsignedAttributes` array yang secara eksplisit mencantumkan semua atribut yang diharapkan klien tidak ditandatangani saat bertemu dengan mereka pada dekripsi. Anda hanya harus secara eksplisit mendefinisikan atribut unsigned yang diizinkan jika benar-benar diperlukan.

Konfigurasi Pencarian (Opsional)

`SearchConfig`Mendefinisikan versi beacon.

`SearchConfig`Harus ditentukan untuk menggunakan enkripsi yang dapat dicari atau suar yang ditandatangani.

Suite Algoritma (Opsional)

`algorithmSuiteId`Mendefinisikan algoritma mana yang sesuai dengan AWS Database Encryption SDK yang digunakan.

Kecuali Anda secara eksplisit menentukan rangkaian algoritme alternatif, SDK Enkripsi AWS Database menggunakan rangkaian algoritme default. Rangkaian algoritme default menggunakan algoritma AES-GCM dengan derivasi kunci, tanda tangan digital, dan komitmen kunci. Meskipun rangkaian algoritme default kemungkinan cocok untuk sebagian besar aplikasi, Anda dapat memilih rangkaian algoritme alternatif. Misalnya, beberapa model kepercayaan akan dipenuhi oleh rangkaian algoritma tanpa tanda tangan digital. Untuk informasi tentang rangkaian algoritme yang didukung SDK Enkripsi AWS Database, lihatRangkaian algoritme yang didukung di SDK Enkripsi AWS Database.

Untuk memilih rangkaian algoritma AES-GCM tanpa tanda tangan digital ECDSA, sertakan cuplikan berikut dalam konfigurasi enkripsi tabel Anda.

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

Memperbarui item dengan SDK Enkripsi AWS Database

SDK Enkripsi AWS Database tidak mendukung ddb: UpdateItem untuk item yang menyertakan atribut terenkripsi atau ditandatangani. Untuk memperbarui atribut terenkripsi atau ditandatangani, Anda harus menggunakan ddb:. PutItem Saat Anda menentukan kunci utama yang sama dengan item yang ada dalam `PutItem` permintaan Anda, item baru sepenuhnya menggantikan item yang ada.

Klien Enkripsi DynamoDB Legacy

Pada tanggal 9 Juni 2023, pustaka enkripsi sisi klien kami diubah namanya menjadi Database Encryption SDK. AWS AWS Database Encryption SDK terus mendukung versi Klien Enkripsi DynamoDB lama. Untuk informasi selengkapnya tentang berbagai bagian pustaka enkripsi sisi klien yang berubah dengan penggantian nama, lihat. Ganti nama Klien Enkripsi Amazon DynamoDB

Untuk bermigrasi ke versi terbaru pustaka enkripsi sisi klien Java untuk DynamoDB, lihat. Migrasi ke versi 3.x

Topik

- AWS SDK Enkripsi Database untuk dukungan versi DynamoDB
- Cara kerja DynamoDB Encryption Client
- Konsep Amazon DynamoDB Encryption Client
- Penyedia bahan kriptografi
- Bahasa pemrograman Amazon DynamoDB Encryption Client yang tersedia
- Mengubah model data Anda
- Memecahkan masalah dalam aplikasi DynamoDB Encryption Client Anda

AWS SDK Enkripsi Database untuk dukungan versi DynamoDB

Topik di Bab Legacy memberikan informasi tentang versi 1. x —2. x dari DynamoDB Encryption Client untuk Java dan versi 1. x —3. x dari Klien Enkripsi DynamoDB untuk Python.

Tabel berikut mencantumkan bahasa dan versi yang mendukung enkripsi sisi klien di Amazon DynamoDB.

Bahasa pemrograman	Versi	Fase siklus hidup versi utama SDK
Java	Versi 1. x	End-of-Support fase, efektif Juli 2022
Java	Versi 2. x	Ketersediaan Umum (GA)
Java	Versi 3. x	Ketersediaan Umum (GA)
Python	Versi 1. x	End-of-Support fase, efektif Juli 2022
Python	Versi 2. x	End-of-Support fase, efektif Juli 2022
Python	Versi 3. x	Ketersediaan Umum (GA)

Cara kerja DynamoDB Encryption Client

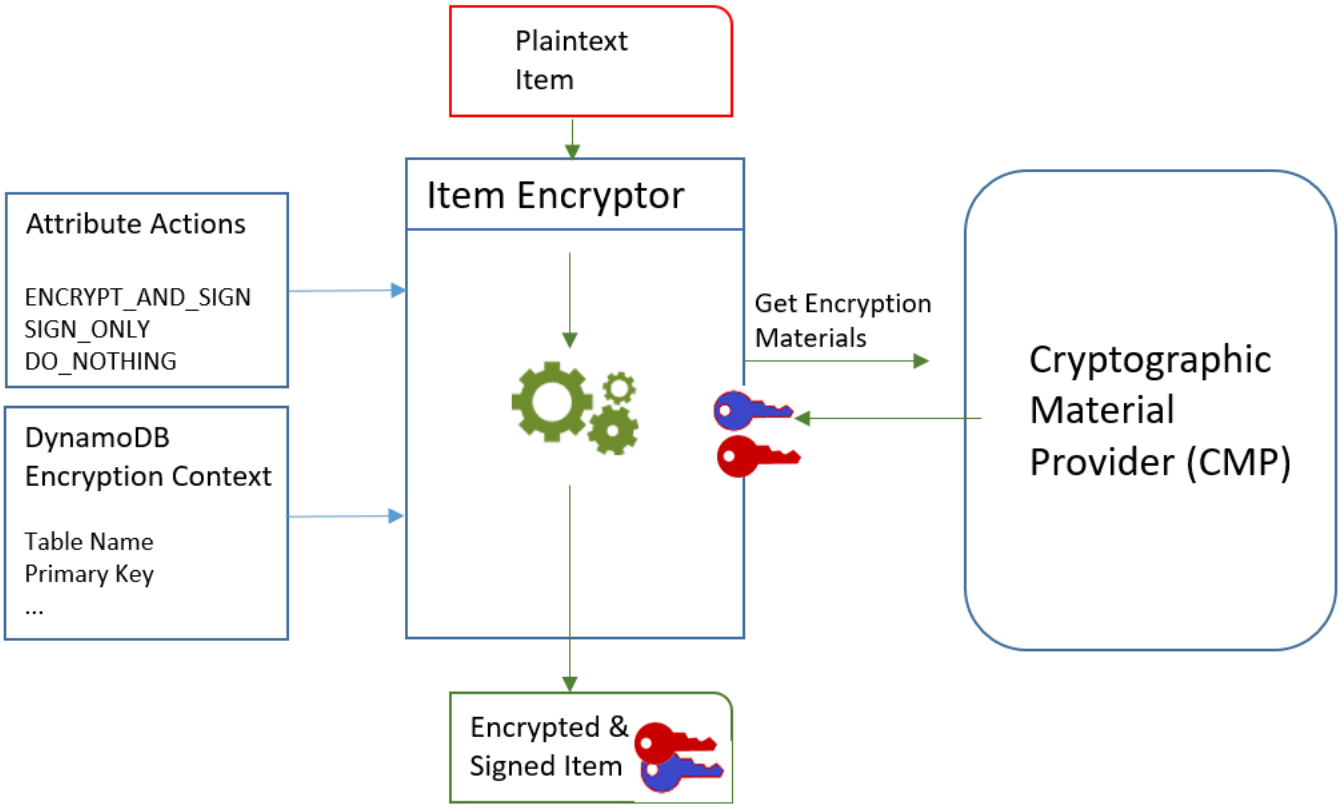
> **Note**
>
> Pustaka enkripsi sisi klien kami diubah namanya menjadi AWS Database Encryption SDK. Topik berikut memberikan informasi tentang versi 1. x —2. x dari DynamoDB Encryption Client untuk Java dan versi 1. x —3. x dari Klien Enkripsi DynamoDB untuk Python. Untuk informasi selengkapnya, lihat SDK Enkripsi AWS Database untuk dukungan versi DynamoDB.

DynamoDB Encryption Client dirancang khusus untuk melindungi data yang Anda simpan di DynamoDB. Pustakanya meliputi implementasi aman yang dapat Anda perluas atau gunakan tanpa perubahan. Dan, sebagian besar elemennya diwakili oleh elemen abstrak sehingga Anda dapat membuat dan menggunakan komponen kustom yang kompatibel.

Mengenkripsi dan menandatangani item tabel

Mesin utama dari DynamoDB Encryption Client adalah enkriptor item yang mengenkripsi, menandatangani, memverifikasi, dan mendekripsi item tabel. Ia menerima informasi tentang item tabel Anda dan memberi perintah tentang item mana yang perlu dienkripsi dan ditandatangani. Ia menerima materi enkripsi, dan memberi perintah tentang cara menggunakannya, dari penyedia materi kriptografis yang Anda pilih dan konfigurasikan.

Diagram berikut menunjukkan garis besar proses ini.



Untuk mengenkripsi dan menandatangani item tabel, DynamoDB Encryption Client memerlukan:

- Informasi tentang tabel. Ia mendapatkan informasi tentang tabel dari konteks enkripsi DynamoDB yang Anda berikan. Beberapa pembantu mendapatkan informasi yang diperlukan dari DynamoDB dan membuat konteks enkripsi DynamoDB untuk Anda.

  > **Note**
  >
  > Konteks enkripsi DynamoDB dalam Klien Enkripsi DynamoDB tidak terkait dengan konteks enkripsi di () dan. AWS Key Management Service AWS KMS AWS Encryption SDK

- Atribut mana yang akan dienkripsi dan ditandatangani. Ia mendapatkan informasi ini dari tindakan atribut yang Anda berikan.
- Materi enkripsi, termasuk enkripsi dan kunci penandatanganan. Ia mendapatkannya dari penyedia materi kriptografis (CMP) yang Anda pilih dan konfigurasikan.

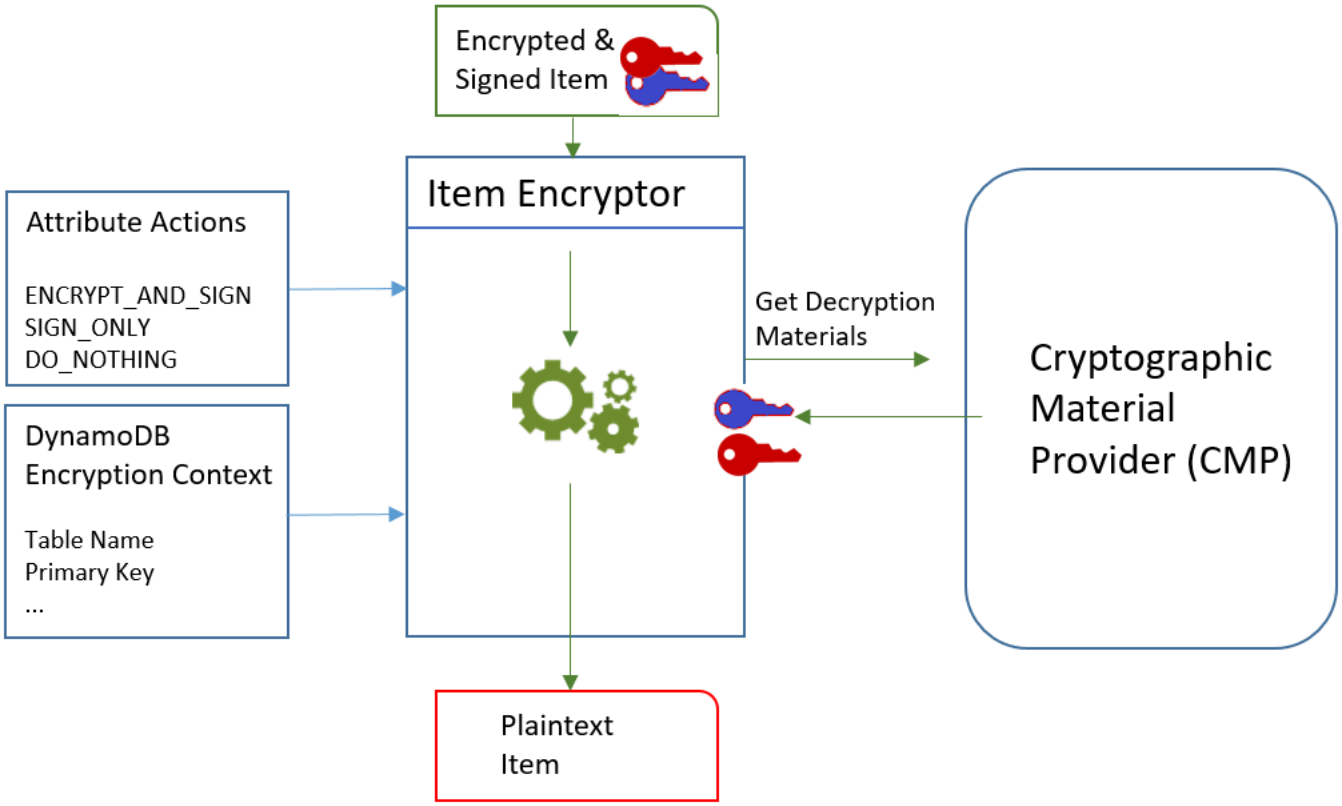
- Instruksi untuk mengenkripsi dan menandatangani item. CMP menambahkan instruksi untuk menggunakan materi enkripsi, termasuk algoritma enkripsi dan penandatanganan, ke deskripsi material aktual.

Enkriptor item menggunakan semua elemen ini untuk mengenkripsi dan menandatangani item. Enkriptor item juga menambahkan dua atribut untuk item: atribut deskripsi materi yang berisi instruksi enkripsi dan penandatanganan (deskripsi materi aktual), dan atribut yang berisi tanda tangan. Anda dapat berinteraksi dengan enkriptor item secara langsung, atau menggunakan fitur pembantu yang berinteraksi dengan enkriptor item untuk Anda untuk menerapkan perilaku default yang aman.

Hasilnya adalah item DynamoDB yang mengandung data yang dienkripsi dan ditandatangani.

Memverifikasi dan mendekripsi item tabel

Komponen ini juga bekerja sama untuk memverifikasi dan mendekripsi item Anda, seperti yang ditunjukkan dalam diagram berikut.



Untuk memverifikasi dan mendekripsi item, DynamoDB Encryption Client membutuhkan komponen yang sama, komponen dengan konfigurasi yang sama, atau komponen yang dirancang khusus untuk mendekripsi item, sebagai berikut:

- Informasi tentang tabel dari konteks enkripsi DynamoDB.
- Atribut mana yang harus diverifikasi dan didekripsi. Ia mendapatkannya dari tindakan atribut.
- Materi dekripsi, termasuk kunci verifikasi dan dekripsi, dari penyedia materi kriptografis (CMP) yang Anda pilih dan konfigurasikan.

  Item terenkripsi tidak termasuk catatan CMP yang digunakan untuk mengenkripsinya. Anda harus menyediakan CMP yang sama, CMP dengan konfigurasi yang sama, atau CMP yang dirancang untuk mendekripsi item.
- Informasi tentang cara item dienkripsi dan ditandatangani, termasuk algoritma enkripsi dan penandatanganan. Klien mendapatkannya dari atribut deskripsi materi dalam item.

Enkriptor item menggunakan semua elemen ini untuk memverifikasi dan mendekripsi item. Ia juga menghapus deskripsi materi dan atribut tanda tangan. Hasilnya adalah item DynamoDB plaintext.

Konsep Amazon DynamoDB Encryption Client

> **Note**
>
> Pustaka enkripsi sisi klien kami diubah namanya menjadi AWS Database Encryption SDK. Topik berikut memberikan informasi tentang versi 1. x —2. x dari DynamoDB Encryption Client untuk Java dan versi 1. x —3. x dari Klien Enkripsi DynamoDB untuk Python. Untuk informasi selengkapnya, lihat SDK Enkripsi AWS Database untuk dukungan versi DynamoDB.

Topik ini menjelaskan konsep dan terminologi yang digunakan dalam Amazon DynamoDB Encryption Client.

Untuk mempelajari bagaimana komponen DynamoDB Encryption Client berinteraksi, lihat Cara kerja DynamoDB Encryption Client.

Topik

- Penyedia materi kriptografis (CMP)

- Enkriptor item
- Tindakan atribut
- Deskripsi materi
- Konteks enkripsi DynamoDB
- Penyimpanan penyedia

Penyedia materi kriptografis (CMP)

Saat mengimplementasikan DynamoDB Encryption Client, salah satu tugas pertama Anda adalah memilih penyedia materi kriptografis (CMP) (juga dikenal sebagai penyedia materi enkripsi). Pilihan Anda menentukan sebagian besar aspek lain dalam implementasinya.

Penyedia materi kriptografis (CMP) mengumpulkan, menyusun, dan mengembalikan materi kriptografis yang digunakan enkriptor item untuk mengenkripsi dan menandatangani item tabel Anda. CMP menentukan algoritma enkripsi yang digunakan serta cara menghasilkan dan melindungi kunci enkripsi dan penandatanganan.

CMP berinteraksi dengan enkriptor item. Enkriptor item meminta materi enkripsi atau dekripsi dari CMP, dan CMP mengembalikannya ke enkriptor item. Kemudian, enkriptor item menggunakan materi kriptografis untuk mengenkripsi dan menandatangani, atau memverifikasi dan mendekripsi, item.

Anda menentukan CMP saat Anda mengonfigurasi klien. Anda dapat membuat CMP kustom yang kompatibel, atau menggunakan salah satu dari banyak CMPs di perpustakaan. Sebagian besar CMPs tersedia untuk beberapa bahasa pemrograman.

Enkriptor item

Enkriptor item adalah komponen dengan tingkat yang lebih rendah yang melakukan operasi kriptografis untuk DynamoDB Encryption Client. Ia meminta materi kriptografis dari penyedia materi kriptografis (CMP), kemudian menggunakan materi yang dikembalikan CMP untuk mengenkripsi dan menandatangani, atau memverifikasi dan mendekripsi, item tabel Anda.

Anda dapat berinteraksi dengan enkriptor item secara langsung atau menggunakan bantuan yang disediakan pustaka Anda. Sebagai contoh, DynamoDB Encryption Client untuk Java mencakup kelas bantuan `AttributeEncryptor` yang dapat Anda gunakan dengan `DynamoDBMapper`, alih-alih berinteraksi secara langsung dengan enkriptor item `DynamoDBEncryptor`. Pustaka Python mencakup kelas bantuan `EncryptedTable`, `EncryptedClient`, dan `EncryptedResource` yang berinteraksi dengan enkriptor item untuk Anda.

Tindakan atribut

Tindakan atribut memberitahukan kepada enkriptor item tentang tindakan mana yang perlu dilakukan pada setiap atribut item.

Nilai tindakan atribut dapat menjadi berupa salah satu dari yang berikut:

- Enkripsi dan tanda tangan – Enkripsi nilai atribut. Sertakan atribut (nama dan nilai) dalam tanda tangan item.
- Hanya tanda tangan – Sertakan atribut dalam tanda tangan item.
- Jangan lakukan apa pun – Jangan enkripsi atau tanda tangani atribut.

Untuk semua atribut yang dapat menyimpan data sensitif, gunakan Enkripsi dan tanda tangan. Untuk atribut kunci utama (kunci partisi dan kunci penyortiran), gunakan Hanya tanda tangan. Atribut deskripsi materi dan atribut tanda tangan tidak ditandatangani atau dienkripsi. Anda tidak perlu menentukan tindakan atribut untuk atribut ini.

Pilih tindakan atribut Anda dengan hati-hati. Bila ragu, gunakan Enkripsi dan tanda tangan. Begitu Anda menggunakan DynamoDB Encryption Client untuk melindungi item tabel Anda, Anda tidak dapat mengubah tindakan untuk atribut tanpa menimbulkan risiko kesalahan validasi tanda tangan. Untuk detailnya, lihat Mengubah model data Anda.

> **Warning**
>
> Jangan mengenkripsi atribut kunci utama. Atribut tersebut harus tetap dalam plaintext sehingga DynamoDB dapat menemukan item tanpa memindai keseluruhan tabel.

Jika konteks enkripsi DynamoDB mengidentifikasi atribut kunci utama Anda, klien akan menyebabkan kesalahan jika Anda mencoba mengenkripsinya.

Teknik yang Anda gunakan untuk menentukan tindakan atribut berbeda untuk setiap bahasa pemrograman. Ini mungkin juga spesifik untuk kelas bantuan yang Anda gunakan.

Untuk informasi lebih lanjut, lihat dokumentasi untuk bahasa pemrograman Anda.

- Python
- Java

Deskripsi materi

Deskripsi materi untuk item tabel terenkripsi terdiri atas informasi, seperti algoritma enkripsi, tentang bagaimana item tabel dienkripsi dan ditandatangani. Penyedia materi kriptografis (CMP) mencatat deskripsi materi saat menyusun materi kriptografis untuk enkripsi dan penandatanganan. Kemudian, ketika perlu menyusun materi kriptografis untuk memverifikasi dan mendekripsi item, ia menggunakan deskripsi materi sebagai panduan.

Pada DynamoDB Encryption Client, deskripsi materi mengacu pada tiga elemen terkait:

Deskripsi materi yang diminta

Beberapa penyedia materi kriptografi (CMPs) memungkinkan Anda menentukan opsi lanjutan, seperti algoritma enkripsi. Untuk menunjukkan pilihan Anda, Anda menambahkan pasangan nama-nilai ke properti deskripsi materi dari konteks enkripsi DynamoDB dalam permintaan Anda untuk mengenkripsi item tabel. Elemen ini dikenal sebagai deskripsi materi yang diminta. Nilai yang valid dalam deskripsi materi yang diminta didefinisikan oleh CMP yang Anda pilih.

> **Note**
>
> Karena deskripsi materi dapat menimpa nilai default yang aman, kami sarankan Anda menghilangkan deskripsi materi yang diminta kecuali jika Anda memiliki alasan kuat untuk menggunakannya.

Deskripsi materi aktual

Deskripsi materi yang dikembalikan oleh penyedia bahan kriptografi (CMPs) dikenal sebagai deskripsi materi yang sebenarnya. Ini menggambarkan nilai-nilai aktual yang digunakan CMP ketika menyusun materi kriptografis. Biasanya terdiri atas deskripsi materi yang diminta, jika ada, dengan penambahan dan perubahan.

Atribut deskripsi materi

Klien menyimpan deskripsi materi aktual dalam atribut deskripsi materi item yang dienkripsi. Nama atribut deskripsi materi adalah `amzn-ddb-map-desc` dan nilainya adalah deskripsi materi aktual. Klien menggunakan nilai dalam atribut deskripsi materi untuk memverifikasi dan mendekripsi item.

Konteks enkripsi DynamoDB

Konteks enkripsi DynamoDB menyediakan informasi tentang tabel dan item untuk penyedia materi kriptografis (CMP). Dalam implementasi lanjutan, konteks enkripsi DynamoDB dapat mencakup deskripsi materi yang diminta.

Ketika Anda mengenkripsi item tabel, konteks enkripsi DynamoDB secara kriptografis terikat dengan nilai-nilai atribut terenkripsi. Ketika Anda mendekripsi, jika konteks enkripsi DynamoDB tidak sama persis dalam hal penulisan huruf besar-kecil dengan konteks enkripsi DynamoDB yang digunakan untuk mengenkripsi, operasi dekripsi gagal. Jika Anda berinteraksi dengan enkriptor item secara langsung, Anda harus menyediakan konteks enkripsi DynamoDB ketika Anda memanggil metode enkripsi atau dekripsi. Kebanyakan bantuan membuat konteks enkripsi DynamoDB untuk Anda.

> **Note**
>
> Konteks enkripsi DynamoDB dalam Klien Enkripsi DynamoDB tidak terkait dengan konteks enkripsi di () dan. AWS Key Management Service AWS KMS AWS Encryption SDK

Konteks enkripsi DynamoDB dapat mencakup bidang berikut. Semua bidang dan nilai bersifat opsional.

- Nama tabel
- Nama kunci partisi
- Nama kunci penyortiran
- Atribut pasangan nama-nilai
- Deskripsi materi yang diminta

Penyimpanan penyedia

Toko penyedia adalah komponen yang mengembalikan penyedia bahan kriptografi (CMPs). Toko penyedia dapat membuat CMPs atau mendapatkannya dari sumber lain, seperti toko penyedia lain. Toko penyedia menyimpan versi CMPs yang dibuatnya dalam penyimpanan persisten di mana setiap CMP yang disimpan diidentifikasi dengan nama material pemohon dan nomor versi.

Penyedia Terbaru di Klien Enkripsi DynamoDB mendapatkannya CMPs dari toko penyedia, tetapi Anda dapat menggunakan toko penyedia untuk CMPs memasok ke komponen apa pun. Setiap

Penyedia Terbaru dikaitkan dengan satu toko penyedia, tetapi toko penyedia dapat memasok CMPs ke banyak pemohon di beberapa host.

Toko penyedia membuat versi baru sesuai permintaan, dan mengembalikan versi baru dan yang sudah ada. CMPs Penyimpanan penyedia ini juga mengembalikan nomor versi terbaru untuk nama materi tertentu. Hal ini memungkinkan peminta tahu kapan penyimpanan penyedia memiliki versi baru dari CMP yang dapat diminta.

Klien Enkripsi DynamoDB mencakup MetaStore, yang merupakan toko penyedia yang membuat CMPs Dibungkus dengan kunci yang disimpan di DynamoDB dan dienkripsi dengan menggunakan Klien Enkripsi DynamoDB internal.

Pelajari lebih lanjut:

- Penyimpanan penyedia: Java, Python
- MetaStore: Jawa, Python

Penyedia bahan kriptografi

> **Note**
>
> Pustaka enkripsi sisi klien kami diubah namanya menjadi AWS Database Encryption SDK. Topik berikut memberikan informasi tentang versi 1. x —2. x dari DynamoDB Encryption Client untuk Java dan versi 1. x —3. x dari Klien Enkripsi DynamoDB untuk Python. Untuk informasi selengkapnya, lihat SDK Enkripsi AWS Database untuk dukungan versi DynamoDB.

Salah satu keputusan terpenting yang Anda buat saat menggunakan DynamoDB Encryption Client adalah memilih penyedia bahan kriptografi (CMP). CMP merakit dan mengembalikan bahan kriptografi untuk enkriptor item. Hal ini juga menentukan bagaimana kunci enkripsi dan penandatanganan dihasilkan, apakah bahan kunci baru dihasilkan untuk setiap item atau digunakan kembali, serta algoritma enkripsi dan penandatanganan yang digunakan.

Anda dapat memilih CMP dari implementasi yang disediakan di pustaka DynamoDB Encryption Client atau membangun CMP kustom yang kompatibel. Pilihan CMP Anda mungkin juga tergantung pada bahasa pemrograman yang Anda gunakan.

Topik ini menjelaskan yang paling umum CMPs dan menawarkan beberapa saran untuk membantu Anda memilih yang terbaik untuk aplikasi Anda.

Penyedia Bahan KMS Langsung

Penyedia Bahan KMS Langsung melindungi item tabel Anda di bawah item AWS KMS keyyang tidak pernah meninggalkan AWS Key Management Service(AWS KMS) tidak terenkripsi. Aplikasi Anda tidak harus menghasilkan atau mengelola bahan kriptografi apa pun. Karena menggunakan AWS KMS key untuk menghasilkan enkripsi unik dan kunci penandatanganan untuk setiap item, penyedia ini memanggil AWS KMS setiap kali mengenkripsi atau mendekripsi item.

Jika Anda menggunakan AWS KMS dan satu AWS KMS panggilan per transaksi praktis untuk aplikasi Anda, penyedia ini adalah pilihan yang baik.

Untuk detailnya, lihat Penyedia Bahan KMS Langsung.

Penyedia Bahan Terbungkus (CMP Terbungkus)

Penyedia Bahan Terbungkus (CMP Terbungkus) memungkinkan Anda menghasilkan dan mengelola kunci pembungkus dan penandatanganan di luar DynamoDB Encryption Client.

CMP Terbungkus menghasilkan kunci enkripsi yang unik untuk setiap item. Kemudian itu menggunakan kunci pembungkus (atau pembuka bungkus) dan penandatanganan yang Anda berikan. Dengan demikian, Anda menentukan bagaimana kunci pembungkus dan penandatanganan dihasilkan dan apakah mereka itu unik untuk setiap item atau digunakan kembali. Wrapped CMP adalah alternatif yang aman untuk Direct KMS Provider untuk aplikasi yang tidak menggunakan AWS KMS dan dapat mengelola materi kriptografi dengan aman.

Untuk detailnya, lihat Penyedia Materi Terbungkus.

Penyedia Terbaru

Penyedia Terbaru adalah penyedia bahan kriptografi (CMP) yang dirancang untuk bekerja dengan toko penyedia. Itu didapat CMPs dari toko penyedia, dan mendapatkan materi kriptografi yang dikembalikan dari toko. CMPs Penyedia Terbaru biasanya menggunakan setiap CMP untuk memenuhi beberapa permintaan untuk bahan kriptografi, tetapi Anda dapat menggunakan fitur dari toko penyedia untuk mengendalikan sejauh mana bahan akan digunakan kembali, menentukan seberapa sering CMP dirotasi, dan bahkan mengubah jenis CMP yang digunakan tanpa mengubah Penyedia Terbaru.

Anda dapat menggunakan Penyedia Terbaru dengan toko penyedia yang kompatibel. Klien Enkripsi DynamoDB mencakup, MetaStore yang merupakan toko penyedia yang mengembalikan Wrapped. CMPs

Penyedia Terbaru adalah pilihan yang baik untuk aplikasi yang perlu meminimalkan panggilan ke sumber kriptografi mereka, dan aplikasi yang dapat menggunakan kembali beberapa bahan kriptografi tanpa melanggar persyaratan keamanan mereka. Misalnya, ini memungkinkan Anda untuk melindungi materi kriptografi Anda di bawah AWS KMS keyin AWS Key Management Service(AWS KMS) tanpa menelepon AWS KMS setiap kali Anda mengenkripsi atau mendekripsi item.

Untuk detailnya, lihat Penyedia Terbaru.

Penyedia Bahan Statis

Penyedia Bahan Statis dirancang untuk pengujian, proof-of-concept demonstrasi, dan kompatibilitas lama. Penyedia ini tidak menghasilkan bahan kriptografi yang unik untuk setiap item. Ia mengembalikan enkripsi dan kunci penandatanganan yang sama yang Anda berikan, dan kunci-kunci tersebut digunakan langsung untuk mengenkripsi, mendekripsi, dan menandatangani item tabel Anda.

> **Note**
>
> Penyedia Statis Asimetris di pustaka Java bukan penyedia statis. Penyedia ini hanya memasok konstruktor alternatif untuk CMP Terbungkus. Hal ini aman untuk penggunaan produksi, tetapi Anda harus menggunakan CMP Terbungkus langsung bila memungkinkan.

Topik

- Penyedia Bahan KMS Langsung
- Penyedia Materi Terbungkus
- Penyedia Terbaru
- Penyedia Materi Statis

Penyedia Bahan KMS Langsung

> **Note**
>
> Pustaka enkripsi sisi klien kami diubah namanya menjadi AWS Database Encryption SDK. Topik berikut memberikan informasi tentang versi 1. x —2. x dari DynamoDB Encryption Client untuk Java dan versi 1. x —3. x dari Klien Enkripsi DynamoDB untuk Python. Untuk informasi selengkapnya, lihat SDK Enkripsi AWS Database untuk dukungan versi DynamoDB.

Penyedia Bahan KMS Langsung (Penyedia KMS Langsung) melindungi item tabel Anda di bawah item AWS KMS keyyang tidak pernah meninggalkan AWS Key Management Service(AWS KMS) tidak terenkripsi. Penyedia bahan kriptografi ini mengembalikan kunci enkripsi yang unik dan kunci penandatanganan untuk setiap item tabel. Untuk melakukannya, ia memanggil AWS KMS setiap kali Anda mengenkripsi atau mendekripsi item.

Jika Anda memproses item DynamoDB pada frekuensi tinggi dan skala besar, Anda mungkin melebihi batas, menyebabkan AWS KMS requests-per-secondpenundaan pemrosesan. Jika Anda perlu melebihi batas, buat kasing di AWS Dukungan Pusat. Anda juga dapat mempertimbangkan untuk menggunakan penyedia bahan kriptografi dengan penggunaan kembali kunci secara terbatas, seperti Penyedia Terbaru.

Untuk menggunakan Penyedia KMS Langsung, penelepon harus memiliki Akun AWS, setidaknya satu AWS KMS key, dan izin untuk memanggil GenerateDataKeydan mendekripsi operasi pada. AWS KMS key AWS KMS key Harus berupa kunci enkripsi simetris; Klien Enkripsi DynamoDB tidak mendukung enkripsi asimetris. Jika Anda menggunakan tabel global DynamoDB, Anda mungkin ingin menentukan kunci multi-Wilayah AWS KMS. Untuk detailnya, lihat Cara menggunakannya.

> **Note**
>
> Saat Anda menggunakan Penyedia KMS Langsung, nama dan nilai atribut kunci utama Anda muncul dalam teks biasa dalam konteks AWS KMS enkripsi dan AWS CloudTrail log operasi terkait. AWS KMS Namun, DynamoDB Encryption Client tidak pernah mengekspos plaintext dari nilai-nilai atribut terenkripsi.

Direct KMS Provider adalah salah satu dari beberapa penyedia materi kriptografi (CMPs) yang didukung oleh Klien Enkripsi DynamoDB. Untuk informasi tentang yang lain CMPs, lihatPenyedia bahan kriptografi.

Misalnya kode, lihat:

- Java: AwsKmsEncryptedItem
- Python:, aws-kms-encrypted-tableaws-kms-encrypted-item

Topik

- Cara menggunakannya
- Cara kerjanya

Cara menggunakannya

Untuk membuat Penyedia KMS Langsung, gunakan parameter ID kunci untuk menentukan kunci KMS enkripsi simetris di akun Anda. Nilai parameter ID kunci dapat berupa ID kunci, ARN kunci, nama alias, atau alias ARN dari. AWS KMS key Untuk detail tentang pengidentifikasi kunci, lihat Pengidentifikasi kunci dalam Panduan Developer AWS Key Management Service .

Penyedia KMS Langsung memerlukan kunci KMS enkripsi simetris. Anda tidak dapat menggunakan kunci KMS asimetris. Namun, Anda dapat menggunakan kunci KMS Multi-wilayah, kunci KMS dengan bahan kunci yang diimpor, atau kunci KMS di toko kunci khusus. Anda harus memiliki izin kms: GenerateDataKey dan kms:Decrypt pada kunci KMS. Dengan demikian, Anda harus menggunakan kunci yang dikelola pelanggan, bukan kunci KMS yang AWS dikelola atau AWS dimiliki.

Klien Enkripsi DynamoDB untuk Python menentukan Wilayah untuk AWS KMS memanggil dari Wilayah dalam nilai parameter ID kunci, jika termasuk satu. Jika tidak, ia menggunakan Wilayah di AWS KMS klien, jika Anda menentukan satu, atau Wilayah yang Anda konfigurasikan di AWS SDK untuk Python (Boto3). Untuk informasi tentang pemilihan Wilayah dengan Python, lihat Konfigurasi di AWS SDK for Python (Boto3) API Referensi.

Klien Enkripsi DynamoDB untuk Java menentukan Wilayah untuk AWS KMS memanggil dari Wilayah di AWS KMS klien, jika klien yang Anda tentukan menyertakan Wilayah. Jika tidak, ia akan menggunakan Wilayah yang Anda konfigurasi di AWS SDK untuk Java. Untuk informasi tentang pemilihan Wilayah di AWS SDK untuk Java, lihat Wilayah AWS seleksi di Panduan AWS SDK untuk Java Pengembang.

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

Contoh berikut menggunakan kunci ARN untuk menentukan. AWS KMS key Jika pengenal kunci Anda tidak menyertakan Wilayah AWS, Klien Enkripsi DynamoDB mendapatkan Wilayah dari sesi Botocore yang dikonfigurasi, jika ada, atau dari default 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)
```

Jika Anda menggunakan tabel global Amazon DynamoDB, sebaiknya enkripsi data Anda di bawah kunci Multi-wilayah. AWS KMS Kunci Multi-Region berbeda Wilayah AWS yang dapat digunakan secara bergantian karena memiliki ID kunci dan bahan kunci yang sama. AWS KMS keys Untuk detailnya, lihat Menggunakan kunci multi-Wilayah dalam Panduan Developer AWS Key Management Service .

> **Note**
>
> Jika Anda menggunakan tabel global versi 2017.11.29, Anda harus menetapkan tindakan atribut agar bidang replikasi yang dicadangkan tidak dienkripsi atau ditandatangani. Untuk detailnya, lihat Masalah dengan tabel global versi lama.

Untuk menggunakan kunci multi-Wilayah dengan DynamoDB Encryption Client, buat kunci multi-Wilayah dan replikasi ke Wilayah di mana aplikasi Anda berjalan. Kemudian konfigurasi Penyedia Langsung KMS untuk menggunakan kunci multi-Wilayah di wilayah di mana klien DynamoDB Encryption Client memanggil AWS KMS.

Contoh berikut mengonfigurasi DynamoDB Encryption Client untuk mengenkripsi data di Wilayah AS Timur (N. Virginia) (us-east-1) dan mendekripsi di Wilayah AS Barat (Oregon) (us-west-2) menggunakan kunci multi-Wilayah.

Java

Dalam contoh ini, Klien Enkripsi DynamoDB mendapatkan Wilayah untuk AWS KMS memanggil dari Wilayah di klien. AWS KMS Nilai `keyArn` mengidentifikasi kunci multi-Wilayah dalam Wilayah yang sama.

```
// 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

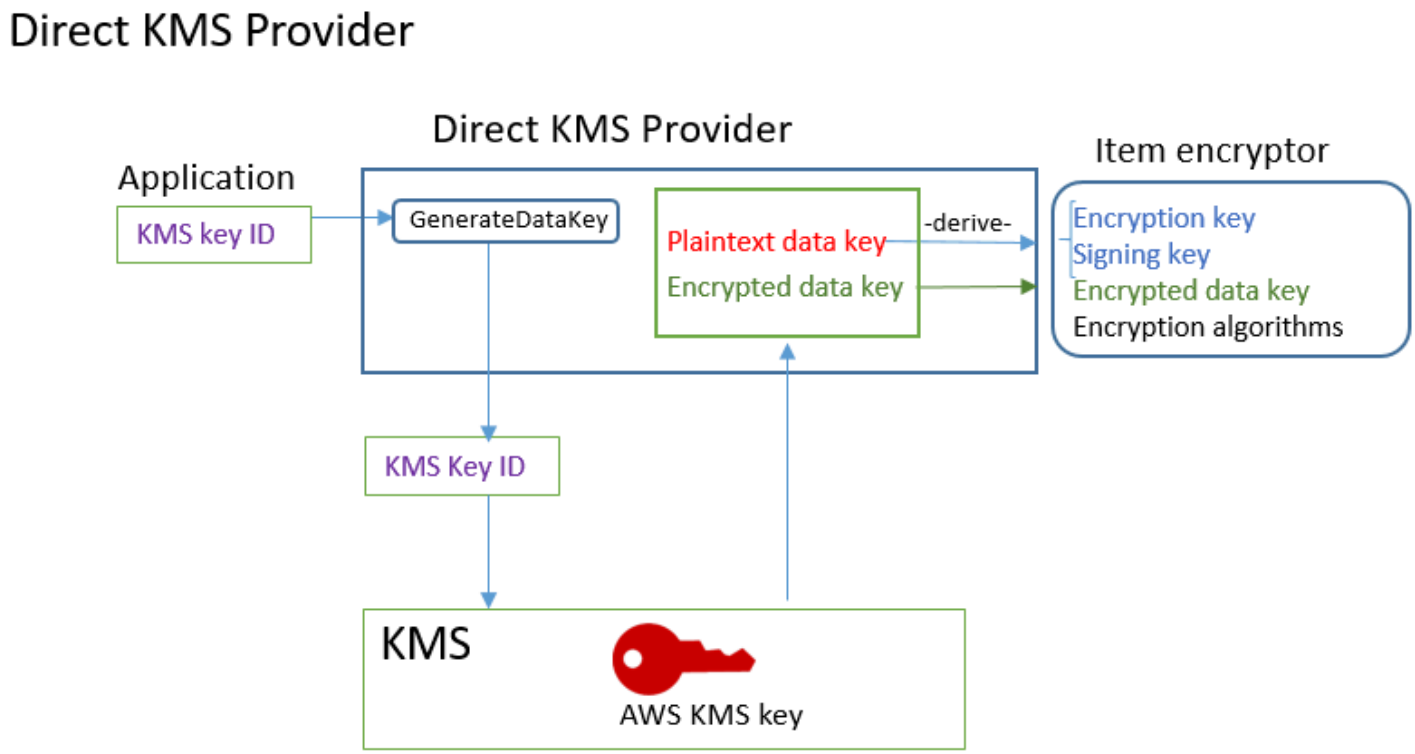
Dalam contoh ini, Klien Enkripsi DynamoDB mendapatkan Wilayah untuk AWS KMS memanggil dari Wilayah di ARN kunci.

```
# 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)
```

Cara kerjanya

Penyedia KMS Langsung mengembalikan kunci enkripsi dan penandatanganan yang dilindungi oleh AWS KMS key yang Anda tentukan, seperti yang ditunjukkan pada diagram berikut.



- Untuk menghasilkan materi enkripsi, Penyedia KMS Langsung meminta AWS KMS untuk membuat kunci data unik untuk setiap item menggunakan AWS KMS key yang Anda tentukan. Penyedia ini mendapatkan kunci enkripsi dan penandatanganan untuk item dari salinan plaintext dari kunci data, kemudian mengembalikan kunci enkripsi dan penandatanganan, bersama dengan kunci data terenkripsi, yang disimpan dalam atribut deskripsi bahan item tersebut.

  Enkriptor item menggunakan kunci enkripsi dan penandatanganan serta membuangnya dari memori secepat mungkin. Hanya salinan terenkripsi dari kunci data dari lokasi mereka diambil yang disimpan dalam item yang dienkripsi.

- Untuk menghasilkan materi dekripsi, Penyedia KMS Langsung meminta AWS KMS untuk mendekripsi kunci data terenkripsi. Kemudian, ia mendapatkan verifikasi dan kunci penandatanganan dari kunci data plaintext, dan mengembalikan mereka ke enkriptor item.

  Enkriptor item memverifikasi item dan, jika verifikasi berhasil, mendekripsi nilai terenkripsi. Kemudian, ia menghapus kunci dari memori sesegera mungkin.

Dapatkan bahan enkripsi

Bagian ini menjelaskan secara detail input, output, dan pemrosesan Penyedia KMS Langsung ketika menerima permintaan bahan enkripsi dari enkriptor item.

Input (dari aplikasi)

- ID kunci dari sebuah AWS KMS key.

Input (dari enkriptor item)

- Konteks enkripsi DynamoDB

Output (untuk enkriptor item)

- Kunci enkripsi (plaintext)
- Kunci penandatanganan
- Dalam deskripsi bahan aktual: Nilai-nilai ini disimpan dalam atribut deskripsi bahan yang ditambahkan klien ke item.
  - amzn-ddb-env-key: Kunci data yang dikodekan Base64 dienkripsi oleh AWS KMS key
  - amzn-ddb-env-alg: Algoritma enkripsi, secara default AES/256
  - amzn-ddb-sig-alg: Algoritma penandatanganan, secara default, Hmac /256 SHA256
  - amzn-ddb-wrap-alg: km

Pengolahan

1. Penyedia KMS Langsung mengirimkan AWS KMS permintaan untuk menggunakan yang ditentukan AWS KMS key untuk menghasilkan kunci data unik untuk item tersebut. Operasi mengembalikan kunci plaintext dan salinan yang dienkripsi di bawah file. AWS KMS key Hal ini dikenal sebagai bahan kunci awal.

Permintaan tersebut mencakup nilai-nilai berikut dalam plaintext pada konteks enkripsi AWS KMS. Nilai-nilai non-rahasia ini secara kriptografi terikat pada objek terenkripsi, sehingga konteks enkripsi yang sama diperlukan pada dekripsi. Anda dapat menggunakan nilai-nilai ini untuk mengidentifikasi panggilan ke AWS KMS dalam AWS CloudTrail log.

- amzn-ddb-env-alg — Algoritma enkripsi, secara default AES/256
- amzn-ddb-sig-alg — Algoritma penandatanganan, secara default Hmac /256 SHA256
- (Opsional) aws-kms-table - *`table name`*
- (Opsional) *`partition key name`* - *`partition key value`* (nilai biner adalah Base64-dikodekan)
- (Opsional) *`sort key name`* - *`sort key value`* (nilai biner adalah Base64-dikodekan)

Direct KMS Provider mendapatkan nilai untuk konteks AWS KMS enkripsi dari konteks enkripsi DynamoDB untuk item tersebut. Jika konteks enkripsi DynamoDB tidak menyertakan nilai, seperti nama tabel, pasangan nama-nilai itu dihilangkan dari konteks enkripsi. AWS KMS

2. Penyedia KMS Langsung mendapat kunci enkripsi simetris dan kunci penandatanganan dari kunci data. Secara default, ia menggunakan Secure Hash Algorithm (SHA) 256 dan Fungsi Derivasi Kunci RFC5869 berbasis HMAC untuk mendapatkan kunci enkripsi simetris AES 256-bit dan kunci penandatanganan HMAC-SHA-256 256-bit.
3. Penyedia KMS Langsung mengembalikan output ke enkriptor item.
4. Enkriptor item menggunakan kunci enkripsi untuk mengenkripsi atribut yang ditentukan dan kunci penandatanganan untuk menandatanganinya, menggunakan algoritma yang ditentukan dalam deskripsi bahan yang aktual. Ia akan menghapus kunci plaintext dari memori sesegera mungkin.

Dapatkan materi dekripsi

Bagian ini menjelaskan secara detail input, output, dan pemrosesan Penyedia KMS Langsung ketika menerima permintaan bahan dekripsi dari enkriptor item.

Input (dari aplikasi)

- ID kunci dari sebuah AWS KMS key.

  Nilai ID kunci dapat berupa ID kunci, kunci ARN, nama alias atau alias ARN dari. AWS KMS key Nilai apa pun yang tidak disertakan dalam ID kunci, seperti Wilayah, harus tersedia di profil AWS bernama. Kunci ARN menyediakan semua nilai yang AWS KMS dibutuhkan.

Input (dari enkriptor item)

- Salinan konteks enkripsi DynamoDB yang berisi konten atribut deskripsi materi.

Output (untuk enkriptor item)

- Kunci enkripsi (plaintext)
- Kunci penandatanganan

Pengolahan

1. Penyedia KMS Langsung mendapatkan kunci data terenkripsi dari atribut deskripsi material dalam item yang dienkripsi.
2. Ia meminta AWS KMS untuk menggunakan yang ditentukan AWS KMS key untuk mendekripsi kunci data terenkripsi. Operasi mengembalikan kunci plaintext.

   Permintaan ini harus menggunakan konteks enkripsi AWS KMS yang digunakan untuk membuat dan mengenkripsi kunci data.

   - aws-kms-table – *`table name`*
   - *`partition key name`*— *`partition key value`* (nilai biner adalah Base64-dikodekan)
   - (Opsional) *`sort key name`* - *`sort key value`* (nilai biner adalah Base64-dikodekan)
   - amzn-ddb-env-alg — Algoritma enkripsi, secara default AES/256
   - amzn-ddb-sig-alg — Algoritma penandatanganan, secara default Hmac /256 SHA256
3. Penyedia KMS Langsung menggunakan Secure Hash Algorithm (SHA) 256 dan Fungsi Derivasi Kunci RFC5869 berbasis HMAC untuk mendapatkan kunci enkripsi simetris AES 256-bit dan kunci penandatanganan HMAC-SHA-256 256-bit dari kunci data.
4. Penyedia KMS Langsung mengembalikan output ke enkriptor item.
5. Enkriptor item menggunakan kunci penandatanganan untuk memverifikasi item. Jika berhasil, ia menggunakan kunci enkripsi simetris untuk mendekripsi nilai atribut terenkripsi. Operasi ini menggunakan algoritma enkripsi dan penandatanganan yang ditentukan dalam deskripsi bahan aktual. Enkriptor item akan menghapus kunci plaintext dari memori sesegera mungkin.

Penyedia Materi Terbungkus

> **Note**
>
> Pustaka enkripsi sisi klien kami diubah namanya menjadi AWS Database Encryption SDK. Topik berikut memberikan informasi tentang versi 1. x —2. x dari DynamoDB Encryption Client untuk Java dan versi 1. x —3. x dari Klien Enkripsi DynamoDB untuk Python. Untuk informasi selengkapnya, lihat SDK Enkripsi AWS Database untuk dukungan versi DynamoDB.

Penyedia Bahan Terbungkus (Wrapped CMP) memungkinkan Anda menggunakan kunci pembungkus dan penandatanganan dari sumber mana pun dengan DynamoDB Encryption Client. CMP yang Dibungkus tidak bergantung pada AWS layanan apa pun. Namun, Anda harus menghasilkan dan mengelola kunci pembungkus dan penandatanganan di luar klien, termasuk menyediakan kunci yang benar untuk memverifikasi dan mendekripsi item.

Wrapped CMP menghasilkan kunci enkripsi item yang unik untuk setiap item. CMP ini membungkus kunci enkripsi item dengan kunci pembungkus yang Anda masukkan dan menyimpan kunci enkripsi item terbungkus dalam atribut deskripsi materi item. Karena Anda memasukkan kunci pembungkus dan penandatangan, Anda menentukan bagaimana kunci pembungkus dan penandatanganan dihasilkan dan apakah kunci itu unik untuk setiap item atau digunakan kembali.

Wrapped CMP adalah implementasi yang aman dan pilihan yang baik untuk aplikasi yang dapat mengelola materi kriptografis.

Wrapped CMP adalah salah satu dari beberapa penyedia bahan kriptografi (CMPs) yang didukung oleh Klien Enkripsi DynamoDB. Untuk informasi tentang yang lain CMPs, lihatPenyedia bahan kriptografi.

Misalnya kode, lihat:

- Java: AsymmetricEncryptedItem
- Python:, wrapped-rsa-encrypted-tablewrapped-symmetric-encrypted-table

Topik

- Cara menggunakannya

- Cara kerjanya

Cara menggunakannya

Untuk membuat Wrapped CMP, tentukan kunci pembungkus (diperlukan saat enkripsi), kunci pembuka pembungkus (diperlukan saat dekripsi), dan kunci penandatanganan. Anda harus menyediakan kunci ketika Anda mengenkripsi dan mendekripsi item.

Kunci pembungkus, pembuka pembungkus, dan penandatanganan dapat berupa kunci simetris atau pasangan kunci asimetris.

Java

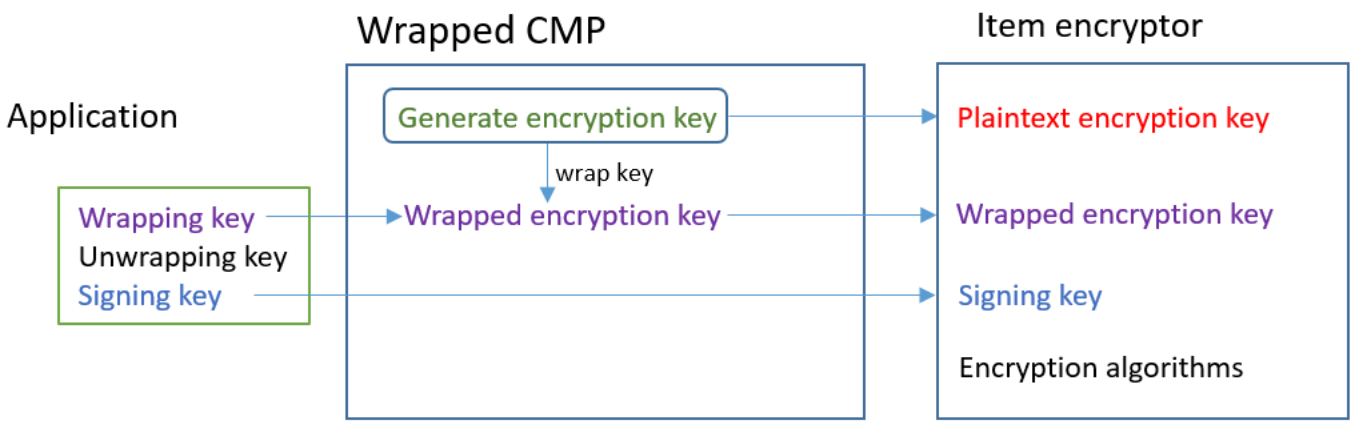
```
// This example uses asymmetric wrapping and signing key pairs
final KeyPair wrappingKeys = ...
final KeyPair signingKeys = ...

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
)
```

Cara kerjanya

Wrapped CMP menghasilkan kunci enkripsi item baru untuk setiap item. CMP ini menggunakan kunci pembungkus, pembuka pembungkus, dan penandatanganan yang Anda berikan, seperti yang ditunjukkan dalam diagram berikut.



Dapatkan materi enkripsi

Bagian ini menjelaskan secara terperinci input, output, dan pengolahan Penyedia Materi Terbungkus (Wrapped CMP) ketika menerima permintaan materi enkripsi.

Input (dari aplikasi)

- Kunci pembungkus: Sebuah kunci simetris Standar Enkripsi Lanjutan (AES), atau kunci publik RSA. Diperlukan jika ada nilai atribut yang dienkripsi. Jika tidak, itu bersifat opsional dan diabaikan.
- Kunci pembuka pembungkus: Opsional dan diabaikan.
- Kunci penandatanganan

Input (dari enkriptor item)

- Konteks enkripsi DynamoDB

Output (untuk enkriptor item):

- Kunci enkripsi item plaintext
- Kunci penandatanganan (tidak berubah)
- Deskripsi materi aktual: Nilai-nilai ini disimpan dalam atribut deskripsi materi yang ditambahkan klien ke item.
  - `amzn-ddb-env-key`: Kunci enkripsi item terbungkus yang didekodekan Base64
  - `amzn-ddb-env-alg`: Algoritma enkripsi yang digunakan untuk mengenkripsi item. Defaultnya adalah AES-256-CBC.

- `amzn-ddb-wrap-alg`: Algoritma pembungkus yang digunakan Wrapped CMP untuk membungkus kunci enkripsi item. Jika kunci pembungkus adalah kunci AES, kuncinya dibungkus menggunakan `AES-Keywrap` tanpa pad sebagaimana didefinisikan dalam RFC 3394. Jika kunci pembungkus adalah kunci RSA, kunci dienkripsi dengan menggunakan RSA OAEP dengan padding. MGF1

Pengolahan

Saat mengenkripsi item, Anda memasukkan kunci pembungkus dan kunci penandatanganan. Kunci pembuka pembungkus bersifat opsional dan diabaikan.

1. Wrapped CMP menghasilkan kunci enkripsi item simetris yang unik untuk setiap item tabel.
2. CMP ini menggunakan kunci pembungkus yang Anda tentukan untuk membungkus kunci enkripsi item. Kemudian, ia menghapusnya dari memori sesegera mungkin.
3. Ia mengembalikan kunci enkripsi item plaintext, kunci penandatanganan yang Anda berikan, dan deskripsi materi aktual yang mencakup kunci enkripsi item terbungkus, serta algoritma enkripsi dan pembungkus.
4. Enkriptor item menggunakan kunci enkripsi plaintext untuk mengenkripsi item. Ia menggunakan kunci penandatanganan yang Anda masukkan untuk menandatangani item. Kemudian, ia menghapus kunci plaintext dari memori sesegera mungkin. Ia menyalin bidang dalam deskripsi materi aktual, termasuk kunci enkripsi terbungkus (`amzn-ddb-env-key`), ke atribut deskripsi materi item.

Dapatkan materi dekripsi

Bagian ini menjelaskan secara terperinci input, output, dan pengolahan Penyedia Materi Terbungkus (Wrapped CMP) ketika menerima permintaan materi dekripsi.

Input (dari aplikasi)

- Kunci pembungkus: Opsional dan diabaikan.
- Kunci pembuka pembungkus: Kunci simetris Standar Enkripsi Lanjutan (AES) yang sama atau kunci privat RSA yang sesuai dengan kunci publik RSA yang digunakan untuk mengenkripsi. Diperlukan jika ada nilai atribut yang dienkripsi. Jika tidak, itu bersifat opsional dan diabaikan.
- Kunci penandatanganan

Input (dari enkriptor item)

- Salinan konteks enkripsi DynamoDB yang berisi konten atribut deskripsi materi.

Output (untuk enkriptor item)

- Kunci enkripsi item plaintext
- Kunci penandatanganan (tidak berubah)

Pengolahan

Ketika Anda mendekripsi item, Anda memasukkan kunci pembuka pembungkus dan kunci penandatanganan. Kunci pembuka pembungkus bersifat opsional dan diabaikan.

1. Wrapped CMP mendapatkan kunci enkripsi item terbungkus dari atribut deskripsi materi dari item.
2. Ia menggunakan kunci pembuka pembungkus dan algoritma untuk membuka bungkus kunci enkripsi item.
3. Ia mengembalikan kunci enkripsi item plaintext, kunci penandatanganan, serta algoritma enkripsi dan penandatanganan ke enkriptor item.
4. Enkriptor item menggunakan kunci penandatanganan untuk memverifikasi item. Jika berhasil, ia menggunakan kunci enkripsi item untuk mendekripsi item. Kemudian, ia menghapus kunci plaintext dari memori sesegera mungkin.

Penyedia Terbaru

> **Note**
>
> Pustaka enkripsi sisi klien kami diubah namanya menjadi AWS Database Encryption SDK. Topik berikut memberikan informasi tentang versi 1. x —2. x dari DynamoDB Encryption Client untuk Java dan versi 1. x —3. x dari Klien Enkripsi DynamoDB untuk Python. Untuk informasi selengkapnya, lihat SDK Enkripsi AWS Database untuk dukungan versi DynamoDB.

Penyedia Terbaru adalah penyedia bahan kriptografi (CMP) yang dirancang untuk bekerja dengan toko penyedia. Itu didapat CMPs dari toko penyedia, dan mendapatkan materi kriptografi yang

dikembalikan dari toko. CMPs Ia biasanya menggunakan setiap CMP untuk memenuhi beberapa permintaan untuk bahan kriptografi. Namun, Anda dapat menggunakan fitur dari toko penyedia untuk mengendalikan sejauh mana bahan akan digunakan kembali, menentukan seberapa sering CMP dirotasi, dan bahkan mengubah jenis CMP yang digunakan tanpa mengubah Penyedia Terbaru.

> **Note**
>
> Kode yang terkait dengan simbol `MostRecentProvider` untuk Penyedia Terbaru mungkin menyimpan bahan kriptografi dalam memori untuk seumur hidup proses. Kode ini mungkin membuat pemanggil dapat menggunakan tombol yang tidak lagi diotorisasi untuk digunakan. Simbol `MostRecentProvider` tidak lagi digunakan di versi terdukung yang lebih lama dari DynamoDB Encryption Client dan dihapus dari versi 2.0.0. Simbol ini digantikan oleh simbol `CachingMostRecentProvider`. Untuk detail selengkapnya, lihat Pembaruan untuk Penyedia Terbaru.

Penyedia Terbaru adalah pilihan yang baik untuk aplikasi yang perlu meminimalkan panggilan ke toko penyedia dan sumber kriptografi mereka, serta aplikasi yang dapat menggunakan kembali beberapa bahan kriptografi tanpa melanggar persyaratan keamanan mereka. Misalnya, ini memungkinkan Anda untuk melindungi materi kriptografi Anda di bawah AWS KMS keyin AWS Key Management Service(AWS KMS) tanpa menelepon AWS KMS setiap kali Anda mengenkripsi atau mendekripsi item.

Toko penyedia yang Anda pilih menentukan jenis CMPs yang digunakan Penyedia Terbaru dan seberapa sering mendapatkan CMP baru. Anda dapat menggunakan toko penyedia yang kompatibel dengan Penyedia Terbaru, termasuk toko penyedia kustom yang Anda desain.

Klien Enkripsi DynamoDB menyertakan MetaStoreyang membuat dan mengembalikan Penyedia Bahan Terbungkus (Dibungkus). CMPs MetaStore Menyimpan beberapa versi Wrapped CMPs yang dihasilkannya dalam tabel DynamoDB internal dan melindunginya dengan enkripsi sisi klien oleh instance internal Klien Enkripsi DynamoDB.

Anda dapat mengonfigurasi MetaStore untuk menggunakan semua jenis CMP internal untuk melindungi materi dalam tabel, termasuk Penyedia KMS Langsung yang menghasilkan materi kriptografi yang dilindungi oleh Anda AWS KMS key, CMP Wrapped yang menggunakan kunci pembungkus dan penandatanganan yang Anda berikan, atau CMP kustom yang kompatibel yang Anda desain.

Misalnya kode, lihat:

- Java: MostRecentEncryptedItem
- Python: most_recent_provider_encrypted_table

Topik

- Cara menggunakannya
- Cara kerjanya
- Pembaruan untuk Penyedia Terbaru

Cara menggunakannya

Untuk membuat Penyedia Terbaru, Anda perlu membuat dan mengonfigurasi toko penyedia, kemudian membuat Penyedia Terbaru yang menggunakan toko penyedia.

Contoh berikut menunjukkan cara membuat Penyedia Terbaru yang menggunakan MetaStore dan melindungi versi dalam tabel DynamoDB internalnya dengan materi kriptografi dari Penyedia KMS Langsung. Contoh-contoh ini menggunakan simbol `CachingMostRecentProvider`.

Setiap Penyedia Terbaru memiliki nama yang mengidentifikasinya CMPs dalam MetaStore tabel, pengaturan time-to-live(TTL), dan pengaturan ukuran cache yang menentukan berapa banyak entri yang dapat disimpan cache. Contoh ini mengatur ukuran cache hingga 1000 entri dan TTL selama 60 detik.

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
)
```

Cara kerjanya

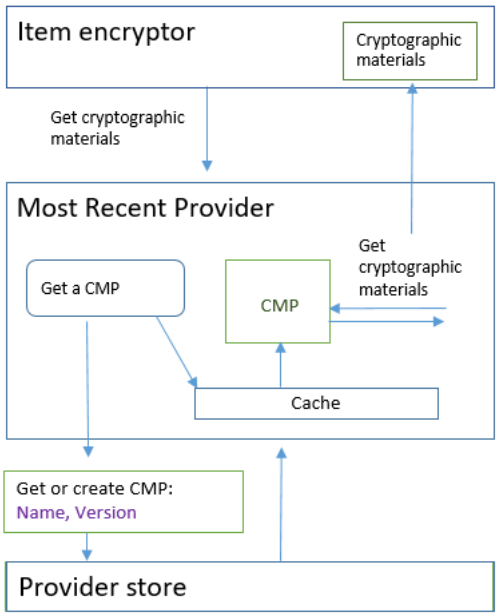
Penyedia Terbaru didapat CMPs dari toko penyedia. Kemudian, penyedia ini menggunakan CMP untuk menghasilkan bahan kriptografi yang dikembalikan ke enkriptor item.

Tentang Penyedia Terbaru

Penyedia Terbaru mendapatkan penyedia bahan kriptografi (CMP) dari penyimpanan penyedia. Kemudian, penyedia ini menggunakan CMP untuk menghasilkan bahan kriptografi yang akan dikembalikan. Setiap Penyedia Terbaru dikaitkan dengan satu toko penyedia, tetapi toko penyedia dapat memasok CMPs ke beberapa penyedia di beberapa host.

Penyedia Terbaru dapat digunakan dengan CMP yang kompatibel dari toko penyedia. Ia meminta bahan enkripsi atau dekripsi dari CMP dan mengembalikan output ke enkriptor item. Ia tidak melakukan operasi kriptografis apa pun.

Untuk meminta CMP dari toko penyedia, Penyedia Terbaru memasok nama bahan dan versi CMP yang ada yang hendak digunakan. Untuk bahan enkripsi, Penyedia Terbaru selalu meminta versi maksimum (“terbaru”). Untuk bahan dekripsi, ia meminta versi CMP yang digunakan untuk membuat bahan enkripsi, seperti yang ditunjukkan dalam diagram berikut.



Penyedia Terbaru menyimpan versi CMPs yang dikembalikan oleh toko penyedia dalam cache Last Recent Used (LRU) lokal di memori. Cache memungkinkan Penyedia Terbaru untuk mendapatkan CMPs yang dibutuhkan tanpa memanggil toko penyedia untuk setiap item. Anda dapat menghapus cache sesuai permintaan.

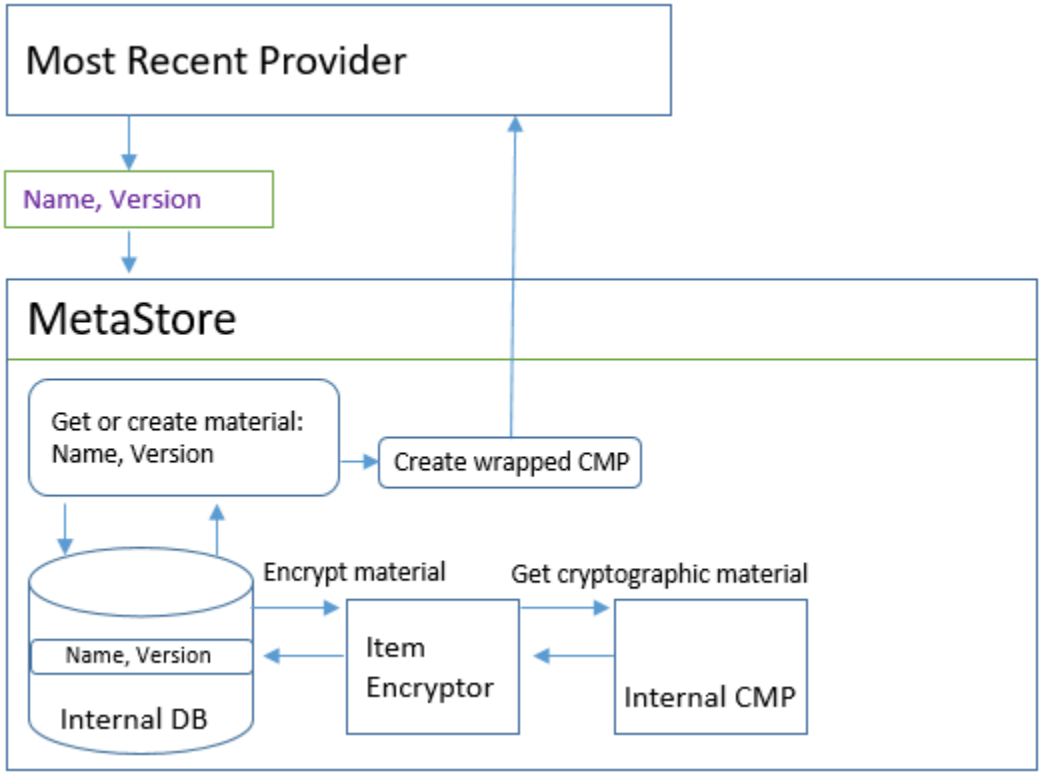
Penyedia Terbaru menggunakan time-to-livenilai yang dapat dikonfigurasi yang dapat Anda sesuaikan berdasarkan karakteristik aplikasi Anda.

Tentang MetaStore

Anda dapat menggunakan Penyedia Terbaru dengan toko penyedia apa pun, termasuk toko penyedia kustom yang kompatibel. Klien Enkripsi DynamoDB mencakup MetaStore, implementasi aman yang dapat Anda konfigurasi dan sesuaikan.

A MetaStoreadalah toko penyedia yang membuat dan mengembalikan Wrapped CMPs yang dikonfigurasi dengan kunci pembungkus, membuka kunci, dan kunci penandatanganan yang diperlukan Wrapped. CMPs A MetaStore adalah opsi aman untuk Penyedia Terbaru karena Wrapped CMPs selalu menghasilkan kunci enkripsi item unik untuk setiap item. Hanya kunci pembungkus yang melindungi kunci enkripsi item dan tombol penandatanganan digunakan kembali.

Diagram berikut menunjukkan komponen MetaStore dan bagaimana berinteraksi dengan Penyedia Terbaru.



MetaStore Menghasilkan Wrapped CMPs, dan kemudian menyimpannya (dalam bentuk terenkripsi) dalam tabel DynamoDB internal. Kunci partisi adalah nama dari bahan Penyedia Terbaru; kunci pengurutan nomor versinya. Bahan-bahan dalam tabel dilindungi oleh DynamoDB Encryption Client internal, termasuk enkriptor item dan penyedia bahan kriptografi (CMP) internal.

Anda dapat menggunakan semua jenis CMP internal di Anda MetaStore, termasuk Penyedia KMS Langsung, CMP Terbungkus dengan materi kriptografi yang Anda berikan, atau CMP kustom yang kompatibel. Jika CMP internal Anda MetaStore adalah Penyedia KMS Langsung, kunci pembungkus dan penandatanganan yang dapat digunakan kembali dilindungi di bawah in (). AWS KMS keyAWS Key Management ServiceAWS KMS MetaStore Panggilan AWS KMS setiap kali menambahkan versi CMP baru ke tabel internalnya atau mendapatkan versi CMP dari tabel internalnya.

Menetapkan time-to-live nilai

Anda dapat menetapkan nilai time-to-live (TTL) untuk setiap Penyedia Terbaru yang Anda buat. Secara umum, gunakan nilai TTL terendah yang praktis untuk aplikasi Anda.

Penggunaan nilai TTL berubah dalam simbol `CachingMostRecentProvider` untuk Penyedia Terbaru.

> **Note**
>
> Simbol `MostRecentProvider` untuk Penyedia Terbaru tidak lagi digunakan di versi terdukung yang lebih lama dari DynamoDB Encryption Client dan dihapus dari versi 2.0.0. Simbol ini digantikan oleh simbol `CachingMostRecentProvider`. Kami menyarankan agar Anda memperbarui kode sesegera mungkin. Untuk detail selengkapnya, lihat Pembaruan untuk Penyedia Terbaru.

`CachingMostRecentProvider`

Parameter `CachingMostRecentProvider` menggunakan nilai TTL dalam dua cara yang berbeda.

- TTL menentukan seberapa sering Penyedia Terbaru memeriksa toko penyedia untuk versi baru CMP. Jika versi baru tersedia, Penyedia Terbaru menggantikan CMP dan menyegarkan bahan kriptografinya. Jika tidak, ia akan terus menggunakan CMP dan bahan kriptografi saat ini.
- TTL menentukan berapa lama CMPs cache dapat digunakan. Sebelum ia menggunakan CMP cache untuk enkripsi, Penyedia Terbaru mengevaluasi waktu dalam cache. Jika waktu cache CMP melebihi TTL, CMP dikosongkan dari cache dan Penyedia Terbaru mendapat CMP versi paling baru dari toko penyedianya.

`MostRecentProvider`

Di `MostRecentProvider`, TTL menentukan seberapa sering Penyedia Terbaru memeriksa toko penyedia untuk versi baru CMP. Jika versi baru tersedia, Penyedia Terbaru menggantikan CMP dan menyegarkan bahan kriptografinya. Jika tidak, ia akan terus menggunakan CMP dan bahan kriptografi saat ini.

TTL tidak menentukan seberapa sering versi CMP baru dibuat. Anda membuat versi CMP baru dengan memutar bahan kriptografi.

Nilai TTL yang ideal bervariasi dengan aplikasi dan tujuan latensi dan ketersediaannya. TTL yang lebih rendah meningkatkan profil keamanan Anda dengan mengurangi waktu untuk menyimpan bahan kriptografi di dalam memori. Selain itu, TTL lebih rendah menyegarkan informasi penting lebih sering. Misalnya, jika CMP internal Anda adalah Penyedia KMS Langsung, itu memverifikasi lebih sering bahwa penelepon masih berwenang untuk menggunakan. AWS KMS key

Jika TTL terlalu singkat, panggilan berulang ke toko penyedia dapat meningkatkan biaya dan menyebabkan toko penyedia Anda untuk membatasi permintaan dari aplikasi Anda dan aplikasi lain yang juga menggunakan akun layanan Anda. Anda juga dapat memperoleh manfaat dari koordinasi TTL dengan tingkat di mana Anda memutar bahan kriptografi.

Selama pengujian, variasikan ukuran TTL dan cache di bawah beban kerja yang berbeda sampai Anda menemukan konfigurasi yang pas untuk aplikasi Anda serta standar keamanan dan performa Anda.

Memutar bahan kriptografi

Ketika Penyedia Terbaru membutuhkan bahan enkripsi, ia selalu menggunakan versi terbaru dari CMP yang dikenalnya. Frekuensi yang diperiksa untuk versi yang lebih baru ditentukan oleh nilai time-to-live(TTL) yang Anda tetapkan saat Anda mengonfigurasi Penyedia Terbaru.

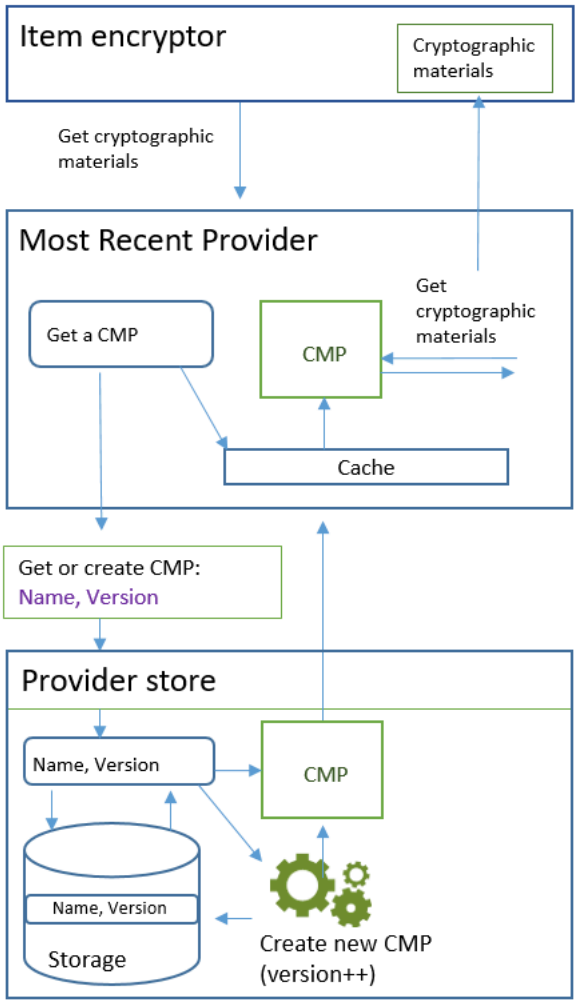
Ketika TTL kedaluwarsa, Penyedia Terbaru memeriksa toko penyedia untuk versi CMP lebih baru. Jika ada yang tersedia, Provider Terbaru mendapatkannya dan menggantikan CMP dalam cache-nya. Ia menggunakan CMP ini dan bahan kriptografi sampai menemukan bahwa toko penyedia memiliki versi yang lebih baru.

Untuk memberi tahu toko penyedia untuk membuat versi baru dari CMP untuk Penyedia Terbaru, panggil operasi Buat Penyedia Baru toko penyedia dengan nama bahan Penyedia Terbaru. Toko penyedia membuat CMP baru dan menyimpan salinan terenkripsi dalam penyimpanan internal dengan nomor versi yang lebih besar. (Hal ini juga mengembalikan CMP, tetapi Anda dapat membuangnya.) Akibatnya, lain kali Penyedia Terbaru menanyakan toko penyedia untuk nomor versi maksimumnya CMPs, ia mendapatkan nomor versi baru yang lebih besar, dan menggunakannya dalam permintaan berikutnya ke toko untuk melihat apakah versi baru CMP telah dibuat.

Anda dapat menjadwalkan panggilan Buat Penyedia Baru berdasarkan waktu, jumlah item atau atribut yang diproses, atau metrik lain yang masuk akal untuk aplikasi Anda.

Dapatkan bahan enkripsi

Penyedia Terbaru menggunakan proses berikut, ditunjukkan dalam diagram ini, untuk mendapatkan bahan enkripsi yang kembali ke enkriptor item. Output bergantung pada jenis CMP yang dikembalikan toko penyedia. Penyedia Terbaru dapat menggunakan toko penyedia yang kompatibel, termasuk MetaStore yang disertakan dalam Klien Enkripsi DynamoDB.



Saat Anda membuat Penyedia Terbaru menggunakan `CachingMostRecentProvider`simbol, Anda menentukan toko penyedia, nama untuk Penyedia Terbaru, dan nilai time-to-live(TTL). Anda juga dapat secara opsional menentukan ukuran cache, yang menentukan jumlah maksimum bahan kriptografi yang dapat ditempatkan di cache.

Ketika enkriptor item meminta Penyedia Terbaru untuk bahan enkripsi, Penyedia Terbaru dimulai dengan mencari cache untuk versi terbaru dari CMP.

- Jika ia menemukan versi terbaru CMP dalam cache dan CMP tidak melebihi nilai TTL, Penyedia Terbaru menggunakan CMP untuk menghasilkan bahan enkripsi. Kemudian, ia mengembalikan bahan enkripsi ke enkriptor item. Operasi ini tidak memerlukan panggilan ke toko penyedia.
- Jika versi terbaru dari CMP tidak berada dalam cache, atau jika berada dalam cache tetapi telah melebihi nilai TTL, Penyedia Terbaru meminta CMP dari toko penyedia. Permintaan tersebut mencakup nama bahan Penyedia Terbaru dan nomor versi maksimum yang diketahui.

1. Toko penyedia mengembalikan CMP dari penyimpanan tetap. Jika toko penyedia adalah a MetaStore, ia mendapatkan CMP Wrapped terenkripsi dari tabel DynamoDB internalnya dengan menggunakan nama materi Penyedia Terbaru sebagai kunci partisi dan nomor versi sebagai kunci pengurutan. MetaStore Menggunakan enkripsi item internal dan CMP internal untuk mendekripsi CMP Wrapped. Kemudian, ia mengembalikan CMP plaintext ke Penyedia Terbaru. Jika CMP internal adalah Penyedia KMS Langsung, langkah ini mencakup panggilan ke AWS Key Management Service (AWS KMS).
2. CMP menambahkan bidang `amzn-ddb-meta-id` ke deskripsi bahan aktual. Nilainya adalah nama bahan dan versi CMP dalam tabel internalnya. Toko penyedia mengembalikan CMP ke Penyedia Terbaru.
3. Penyedia Terbaru menyimpan cache CMP dalam memori.
4. Penyedia Terbaru menggunakan CMP untuk menghasilkan bahan enkripsi. Kemudian, ia mengembalikan bahan enkripsi ke enkriptor item.

Dapatkan bahan dekripsi

Ketika enkriptor item meminta Penyedia Terbaru untuk bahan dekripsi, Penyedia Terbaru menggunakan proses berikut untuk mendapatkan dan mengembalikannya.

1. Penyedia Terbaru meminta toko penyedia untuk nomor versi bahan kriptografi yang digunakan untuk mengenkripsi item. Ia meneruskan deskripsi bahan aktual dari atribut deskripsi bahan dari item.
2. Toko penyedia mendapatkan nomor versi CMP pengenkripsi dari bidang `amzn-ddb-meta-id` di deskripsi bahan aktual dan mengembalikannya ke Penyedia Terbaru.
3. Penyedia Terbaru mencari cache untuk versi CMP yang digunakan untuk mengenkripsi dan menandatangani item.

- Jika menemukan versi CMP yang cocok ada di cache dan CMP belum melebihi nilai time-to-live (TTL), Penyedia Terbaru menggunakan CMP untuk menghasilkan bahan dekripsi. Kemudian, ia mengembalikan bahan dekripsi ke enkriptor item. Operasi ini tidak memerlukan panggilan ke toko penyedia atau CMP lainnya.
- Jika versi CMP yang cocok tidak ada dalam cache-nya, atau jika cache AWS KMS key telah melebihi nilai TTL-nya, Penyedia Terbaru meminta CMP dari toko penyedianya. Ia mengirimkan nama bahan dan nomor versi CMP pengenkripsi dalam permintaan.

1. Toko penyedia mencari penyimpanan tetap untuk CMP dengan menggunakan nama Penyedia Terbaru sebagai kunci partisi dan nomor versi sebagai kunci pengurutan.
   - Jika nama dan nomor versi tidak berada dalam penyimpanan tetap, toko penyedia akan memunculkan pengecualian. Jika toko penyedia digunakan untuk menghasilkan CMP, CMP harus disimpan dalam penyimpanan tetap, kecuali jika itu sengaja dihapus.
   - Jika CMP dengan nama yang cocok dan nomor versi berada dalam penyimpanan tetap toko penyedia, toko penyedia mengembalikan CMP yang ditentukan untuk Penyedia Terbaru.

     Jika toko penyedia adalah a MetaStore, ia mendapatkan CMP terenkripsi dari tabel DynamoDB-nya. Kemudian, ia menggunakan bahan kriptografi dari CMP internal untuk mendekripsi CMP terenkripsi sebelum mengembalikan CMP ke Penyedia Terbaru. Jika CMP internal adalah Penyedia KMS Langsung, langkah ini mencakup panggilan ke AWS Key Management Service (AWS KMS).
2. Penyedia Terbaru menyimpan cache CMP dalam memori.
3. Penyedia Terbaru menggunakan CMP untuk menghasilkan bahan dekripsi. Kemudian, ia mengembalikan bahan dekripsi ke enkriptor item.

Pembaruan untuk Penyedia Terbaru

Simbol untuk Penyedia Terbaru diubah dari `MostRecentProvider` ke `CachingMostRecentProvider`.

> **Note**
>
> Simbol `MostRecentProvider`, yang mewakili Penyedia Terbaru, tidak lagi digunakan dalam versi 1.15 DynamoDB Encryption Client untuk Java dan versi 1.3 DynamoDB DynamoDB Encryption Client untuk Python dan dihapus dari versi 2.0.0 DynamoDB Encryption Client di kedua implementasi bahasa. Sebagai gantinya, gunakan `CachingMostRecentProvider`.

`CachingMostRecentProvider` menerapkan perubahan berikut:

- `CachingMostRecentProvider`Secara berkala menghapus materi kriptografi dari memori ketika waktu mereka dalam memori melebihi nilai yang dikonfigurasi time-to-live (TTL).

`MostRecentProvider` mungkin menyimpan bahan kriptografi dalam memori selama masa proses. Akibatnya, Penyedia Terbaru mungkin tidak menyadari perubahan otorisasi. Ia mungkin menggunakan kunci enkripsi setelah izin pemanggil untuk menggunakannya dicabut.

Jika Anda tidak dapat memperbarui ke versi baru ini, Anda bisa mendapatkan efek yang sama dengan secara berkala memanggil metode `clear()` pada cache. Metode ini secara manual membuang isi cache dan membutuhkan Penyedia Terbaru untuk meminta CMP baru dan bahan kriptografi baru.

- `CachingMostRecentProvider` juga mencakup pengaturan ukuran cache yang memberi Anda lebih banyak kontrol atas cache.

Untuk memperbarui ke `CachingMostRecentProvider`, Anda harus mengubah nama simbol dalam kode Anda. Dalam semua hal lainnya, `CachingMostRecentProvider` kompatibel mundur sepenuhnya dengan `MostRecentProvider`. Anda tidak perlu mengenkripsi ulang item tabel.

Namun, `CachingMostRecentProvider` menghasilkan lebih banyak panggilan ke infrastruktur kunci yang mendasarinya. Ini memanggil toko penyedia setidaknya sekali dalam setiap interval time-to-live (TTL). Aplikasi dengan banyak aktif CMPs (karena rotasi yang sering) atau aplikasi dengan armada besar kemungkinan besar sensitif terhadap perubahan ini.

Sebelum merilis kode yang diperbarui, uji secara menyeluruh untuk memastikan bahwa panggilan yang lebih sering tidak mengganggu aplikasi Anda atau menyebabkan pembatasan oleh layanan tempat penyedia Anda bergantung, seperti AWS Key Management Service () atau AWS KMS Amazon DynamoDB. Untuk mengurangi masalah kinerja apa pun, sesuaikan ukuran cache dan ukuran `CachingMostRecentProvider` berdasarkan karakteristik kinerja yang Anda amati. time-to-live Untuk panduan, lihat Menetapkan time-to-live nilai.

Penyedia Materi Statis

> **Note**
>
> Pustaka enkripsi sisi klien kami diubah namanya menjadi AWS Database Encryption SDK. Topik berikut memberikan informasi tentang versi 1. x —2. x dari DynamoDB Encryption Client untuk Java dan versi 1. x —3. x dari Klien Enkripsi DynamoDB untuk Python. Untuk informasi selengkapnya, lihat SDK Enkripsi AWS Database untuk dukungan versi DynamoDB.

Penyedia Bahan Statis (CMP Statis) adalah penyedia bahan kriptografi (CMP) yang sangat sederhana yang ditujukan untuk pengujian, proof-of-concept demonstrasi, dan kompatibilitas warisan.

Untuk menggunakan Static CMP guna mengenkripsi item tabel, Anda menyediakan kunci enkripsi simetris Standar Enkripsi Lanjutan (AES) dan kunci penandatanganan atau pasangan kunci. Anda harus menyediakan kunci yang sama untuk mendekripsi item yang dienkripsi. Static CMP tidak melakukan operasi kriptografis apa pun. Sebaliknya, Static CMP meneruskan kunci enkripsi yang Anda sediakan ke enkriptor item tanpa perubahan. Enkriptor item mengenkripsi item secara langsung dengan kunci enkripsi. Kemudian, ia menggunakan kunci penandatanganan secara langsung untuk menandatanganinya.

Karena Static CMP tidak menghasilkan materi kriptografis yang unik, semua item tabel yang Anda proses dienkripsi dengan kunci enkripsi yang sama dan ditandatangani dengan kunci penandatanganan yang sama. Ketika Anda menggunakan kunci yang sama untuk mengenkripsi nilai atribut pada banyak item atau menggunakan kunci yang sama atau pasangan kunci untuk menandatangani semua item, ada risiko Anda melebihi batas kriptografis kunci.

> **Note**
>
> Penyedia Statis Asimetris di pustaka Java bukan penyedia statis. Itu hanya memasok konstruktor alternatif untuk Wrapped CMP. Penyedia ini aman untuk penggunaan produksi, tetapi Anda harus menggunakan Wrapped CMP secara langsung bila memungkinkan.

CMP Statis adalah salah satu dari beberapa penyedia bahan kriptografi (CMPs) yang didukung oleh Klien Enkripsi DynamoDB. Untuk informasi tentang yang lain CMPs, lihatPenyedia bahan kriptografi.

Misalnya kode, lihat:

- Java: SymmetricEncryptedItem

Topik

- Cara menggunakannya
- Cara kerjanya

Cara menggunakannya

Untuk membuat penyedia statis, masukkan kunci enkripsi atau pasangan kunci dan kunci penandatanganan atau pasangan kunci. Anda perlu menyediakan materi kunci untuk mengenkripsi dan mendekripsi item tabel.

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);
```

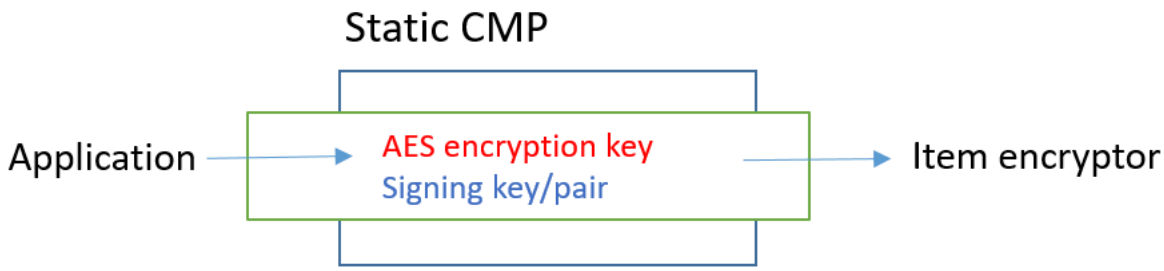
Python

```
# 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
)
```

Cara kerjanya

Provider Statis meneruskan enkripsi dan kunci penandatanganan yang Anda masukkan ke enkriptor item, di mana kunci itu digunakan secara langsung untuk mengenkripsi dan menandatangani item

tabel Anda. Kecuali Anda menyediakan kunci yang berbeda untuk setiap item, kunci yang sama digunakan untuk setiap item.



Dapatkan materi enkripsi

Bagian ini menjelaskan secara terperinci input, output, dan pengolahan Penyedia Materi Statis (Static CMP) ketika menerima permintaan materi enkripsi.

Input (dari aplikasi)

- Kunci enkripsi – Harus berupa kunci simetris, seperti kunci Standar Enkripsi Lanjutan (AES).
- Kunci penandatanganan – Bisa berupa kunci simetris atau pasangan kunci asimetris.

Input (dari enkriptor item)

- Konteks enkripsi DynamoDB

Output (untuk enkriptor item)

- Kunci enkripsi diteruskan sebagai input.
- Kunci penandatanganan diteruskan sebagai input.
- Deskripsi materi aktual: Deskripsi materi yang diminta, jika ada, tanpa perubahan.

Dapatkan materi dekripsi

Bagian ini menjelaskan secara terperinci input, output, dan pengolahan Penyedia Materi Statis (Static CMP) ketika menerima permintaan materi dekripsi.

Meskipun mencakup metode berbeda untuk mendapatkan materi enkripsi dan mendapatkan materi dekripsi, perilakunya sama.

Input (dari aplikasi)

- Kunci enkripsi – Harus berupa kunci simetris, seperti kunci Standar Enkripsi Lanjutan (AES).
- Kunci penandatanganan – Bisa berupa kunci simetris atau pasangan kunci asimetris.

Input (dari enkriptor item)

- Konteks enkripsi DynamoDB (tidak digunakan)

Output (untuk enkriptor item)

- Kunci enkripsi diteruskan sebagai input.
- Kunci penandatanganan diteruskan sebagai input.

Bahasa pemrograman Amazon DynamoDB Encryption Client yang tersedia

> **Note**
>
> Pustaka enkripsi sisi klien kami diubah namanya menjadi AWS Database Encryption SDK. Topik berikut memberikan informasi tentang versi 1. x —2. x dari DynamoDB Encryption Client untuk Java dan versi 1. x —3. x dari Klien Enkripsi DynamoDB untuk Python. Untuk informasi selengkapnya, lihat SDK Enkripsi AWS Database untuk dukungan versi DynamoDB.

Amazon DynamoDB Encryption Client tersedia dalam bahasa pemrograman berikut. Pustaka spesifik-bahasa bervariasi, tetapi implementasi yang dihasilkan dapat dioperasikan. Misalnya, Anda dapat mengenkripsi (dan menandatangani) item dengan klien Java dan mendekripsi item dengan klien Python.

Untuk informasi lebih lanjut, lihat topik terkait.

Topik

- Amazon DynamoDB Encryption Client untuk Java
- DynamoDB Encryption Client untuk Python

Amazon DynamoDB Encryption Client untuk Java

> **Note**
>
> Pustaka enkripsi sisi klien kami diubah namanya menjadi AWS Database Encryption SDK. Topik berikut memberikan informasi tentang versi 1. x —2. x dari DynamoDB Encryption Client untuk Java dan versi 1. x —3. x dari Klien Enkripsi DynamoDB untuk Python. Untuk informasi selengkapnya, lihat SDK Enkripsi AWS Database untuk dukungan versi DynamoDB.

Topik ini menjelaskan cara menginstal dan menggunakan Amazon DynamoDB Encryption Client untuk Java. Untuk detail tentang pemrograman dengan Klien Enkripsi DynamoDB, lihat contoh Java, contoh di repositori GitHubaktif, dan aws-dynamodb-encryption-java Javadoc untuk Klien Enkripsi DynamoDB.

> **Note**
>
> Versi 1. x. x dari Klien Enkripsi DynamoDB untuk Java sedang end-of-support dalam fase efektif Juli 2022. Tingkatkan ke versi yang lebih baru sesegera mungkin.

Topik

- Prasyarat
- Penginstalan
- Menggunakan Amazon DynamoDB Encryption Client untuk Java
- Contoh kode untuk DynamoDB Encryption Client untuk Java

Prasyarat

Sebelum Anda menginstal Amazon DynamoDB Encryption Client untuk Java, pastikan Anda memiliki prasyarat berikut.

Lingkungan pengembangan Java

Anda akan membutuhkan Java 8 atau yang lebih baru. Di situs web Oracle, buka Unduhan Java SE, kemudian unduh dan instal Java SE Development Kit (JDK).

Jika Anda menggunakan Oracle JDK, Anda juga harus mengunduh dan menginstal File Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy.

AWS SDK untuk Java

Klien Enkripsi DynamoDB memerlukan modul DynamoDB bahkan jika aplikasi Anda tidak berinteraksi dengan DynamoDB. AWS SDK untuk Java Anda dapat menginstal seluruh SDK atau modul ini saja. Jika Anda menggunakan Maven, tambahkan `aws-java-sdk-dynamodb` ke file `pom.xml` Anda.

Untuk informasi selengkapnya tentang menginstal dan mengonfigurasi AWS SDK untuk Java, lihat AWS SDK untuk Java.

Penginstalan

Anda dapat menginstal Amazon DynamoDB Encryption Client untuk Java dengan cara berikut.

Secara manual

Untuk menginstal Klien Enkripsi Amazon DynamoDB untuk Java, kloning atau unduh repositori. aws-dynamodb-encryption-java GitHub

Menggunakan Apache Maven

Amazon DynamoDB Encryption Client untuk Java tersedia melalui Apache Maven dengan definisi dependensi berikut.

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-dynamodb-encryption-java</artifactId>
  <version>version-number</version>
</dependency>
```

Setelah Anda menginstal SDK, mulailah dengan melihat kode contoh dalam panduan ini dan DynamoDB Encryption Client Javadoc aktif. GitHub

Menggunakan Amazon DynamoDB Encryption Client untuk Java

> **Note**
>
> Pustaka enkripsi sisi klien kami diubah namanya menjadi AWS Database Encryption SDK. Topik berikut memberikan informasi tentang versi 1. x —2. x dari DynamoDB Encryption

Client untuk Java dan versi 1. x —3. x dari Klien Enkripsi DynamoDB untuk Python. Untuk informasi selengkapnya, lihat SDK Enkripsi AWS Database untuk dukungan versi DynamoDB.

Topik ini menjelaskan beberapa fitur DynamoDB Encryption Client di Java yang mungkin tidak ditemukan dalam implementasi bahasa pemrograman lainnya.

Untuk detail tentang pemrograman dengan Klien Enkripsi DynamoDB, lihat contoh Java, contoh di `aws-dynamodb-encryption-java repository` aktif, dan Javadoc untuk Klien Enkripsi DynamoDB. GitHub

Topik

- Enkripsi item: AttributeEncryptor dan Dynamo DBEncryptor
- Mengonfigurasi perilaku penyimpanan
- Tindakan atribut di Java
- Menimpa nama tabel

Enkripsi item: AttributeEncryptor dan Dynamo DBEncryptor

Klien Enkripsi DynamoDB di Java memiliki dua enkripsi item: Dynamo tingkat rendah dan. DBEncryptor AttributeEncryptor

`AttributeEncryptor`Ini adalah kelas pembantu yang membantu Anda menggunakan Dynamo DBMapper di AWS SDK untuk Java dengan di Klien Enkripsi DynamoDB. `DynamoDB Encryptor` Saat Anda menggunakan `AttributeEncryptor` dengan `DynamoDBMapper`, item tersebut secara transparan mengenkripsi dan menandatangani item Anda saat Anda menyimpannya. Item ini juga secara transparan memverifikasi dan mendekripsi item Anda ketika Anda memuatnya.

Mengonfigurasi perilaku penyimpanan

Anda dapat menggunakan `AttributeEncryptor` dan `DynamoDBMapper` untuk menambah atau mengganti item tabel dengan atribut yang ditandatangani saja atau dienkripsi dan ditandatangani. Untuk tugas-tugas ini, kami sarankan Anda mengonfigurasinya untuk menggunakan perilaku penyimpanan PUT, seperti yang ditunjukkan dalam contoh berikut. Jika tidak, Anda mungkin tidak dapat mendekripsi data.

```
DynamoDBMapperConfig mapperConfig =
 DynamoDBMapperConfig.builder().withSaveBehavior(SaveBehavior.PUT).build();
DynamoDBMapper mapper = new DynamoDBMapper(ddb, mapperConfig, new
 AttributeEncryptor(encryptor));
```

Jika Anda menggunakan perilaku penyimpanan default, yang hanya memperbarui atribut yang dimodelkan dalam item tabel, atribut yang tidak dimodelkan tidak disertakan dalam tanda tangan, dan tidak diubah oleh penulisan tabel. Akibatnya, pada pembacaan semua atribut nanti, tanda tangan tidak akan memvalidasi, karena tidak menyertakan atribut yang tidak dimodelkan.

Anda juga dapat menggunakan perilaku penyimpanan `CLOBBER`. Perilaku ini identik dengan perilaku penyimpanan PUT kecuali pada bagian perilaku ini menonaktifkan penguncian optimis dan menimpa item dalam tabel.

Untuk mencegah kesalahan tanda tangan, Klien Enkripsi DynamoDB melempar pengecualian runtime jika `AttributeEncryptor` digunakan dengan `DynamoDBMapper` yang tidak dikonfigurasi dengan perilaku penyimpanan atau. `CLOBBER` PUT

Untuk melihat kode ini digunakan dalam contoh, lihat Menggunakan Dynamo DBMapper dan AwsKmsEncryptedObjectcontoh.java di `aws-dynamodb-encryption-java` repositori di. GitHub

Tindakan atribut di Java

Tindakan atribut menentukan mana nilai atribut yang dienkripsi dan ditandatangani, mana yang hanya ditandatangani, dan mana yang diabaikan. Metode yang Anda gunakan untuk menentukan tindakan atribut tergantung pada apakah Anda menggunakan `DynamoDBMapper` dan`AttributeEncryptor`, atau Dynamo tingkat yang lebih rendah. DBEncryptor

> **Important**
>
> Setelah Anda menggunakan tindakan atribut untuk mengenkripsi item tabel Anda, menambahkan atau menghapus atribut dari model data Anda dapat menyebabkan kesalahan validasi tanda tangan yang mencegah Anda mendekripsi data Anda. Untuk penjelasan detail, lihat Mengubah model data Anda.

Tindakan atribut untuk Dynamo DBMapper

Saat Anda menggunakan `DynamoDBMapper` dan `AttributeEncryptor`, Anda menggunakan anotasi untuk menentukan tindakan atribut. DynamoDB Encryption Client menggunakan anotasi

atribut DynamoDB standar yang menentukan jenis atribut untuk menentukan cara melindungi atribut. Secara default, semua atribut dienkripsi dan ditandatangani kecuali kunci utama, yang ditandatangani tetapi tidak dienkripsi.

> **Note**
>
> Jangan mengenkripsi nilai atribut dengan anotasi DBVersion Atribut @Dynamo, meskipun Anda dapat (dan harus) menandatanganinya. Jika dilakukan, syarat yang menggunakan nilai tersebut akan memiliki efek yang tidak diinginkan.

```
// 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")
```

Untuk menentukan pengecualian, gunakan anotasi enkripsi yang didefinisikan dalam DynamoDB Encryption Client untuk Java. Jika Anda menentukannya di tingkat kelas, pengecualian itu menjadi nilai default untuk kelas tersebut.

```
// Sign only
@DoNotEncrypt

// Do nothing; not encrypted or signed
@DoNotTouch
```

Sebagai contoh, anotasi ini menandatangani tetapi tidak mengenkripsi atribut `PublicationYear`, dan tidak mengenkripsi atau menandatangani nilai atribut `ISBN`.

```
// Sign only (override the default)
@DoNotEncrypt
@DynamoDBAttribute(attributeName="PublicationYear")

// Do nothing (override the default)
@DoNotTouch
@DynamoDBAttribute(attributeName="ISBN")
```

Tindakan atribut untuk Dynamo DBEncryptor

Untuk menentukan tindakan atribut saat Anda menggunakan Dynamo DBEncryptor secara langsung, buat `HashMap` objek di mana pasangan nama-nilai mewakili nama atribut dan tindakan yang ditentukan.

Nilai-nilai yang valid adalah untuk tindakan atribut yang didefinisikan dalam jenis `EncryptionFlags` yang disebutkan. Anda dapat menggunakan `ENCRYPT` dan `SIGN` secara bersamaan, menggunakan `SIGN` saja, atau tidak menggunakan keduanya. Namun, jika Anda menggunakan `ENCRYPT` saja, DynamoDB Encryption Client menyebabkan kesalahan. Anda tidak dapat mengenkripsi atribut yang tidak Anda tanda tangani.

```
ENCRYPT
SIGN
```

> **Warning**
>
> Jangan mengenkripsi atribut kunci utama. Atribut tersebut harus tetap dalam plaintext sehingga DynamoDB dapat menemukan item tanpa memindai keseluruhan tabel.

Jika Anda menentukan kunci utama dalam konteks enkripsi dan kemudian menetapkan `ENCRYPT` dalam tindakan atribut untuk atribut kunci utama, DynamoDB Encryption Client membuat pengecualian.

Misalnya, kode Java berikut membuat sebuah `actions` HashMap yang mengenkripsi dan menandatangani semua atribut dalam item. `record` Pengecualiannya adalah atribut kunci partisi dan kunci penyortiran, yang ditandatangani tetapi tidak dienkripsi, dan atribut `test`, yang tidak ditandatangani atau dienkripsi.

```
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;
  }
}
```

Kemudian, ketika Anda memanggil metode encryptRecord `DynamoDBEncryptor`, tentukan peta sebagai nilai parameter `attributeFlags`. Sebagai contoh, panggilan untuk `encryptRecord` ini menggunakan peta `actions`.

```
// Encrypt the plaintext record
final Map<String, AttributeValue> encrypted_record = encryptor.encryptRecord(record,
 actions, encryptionContext);
```

Menimpa nama tabel

Pada DynamoDB Encryption Client, nama tabel DynamoDB adalah elemen konteks enkripsi DynamoDB yang diteruskan ke metode enkripsi dan dekripsi. Ketika Anda mengenkripsi atau menandatangani item tabel, konteks enkripsi DynamoDB, termasuk nama tabel, secara kriptografis terikat pada ciphertext. Jika konteks enkripsi DynamoDB yang diteruskan ke metode dekripsi tidak cocok dengan konteks enkripsi DynamoDB yang diteruskan ke metode enkripsi, operasi dekripsi gagal.

Kadang-kadang, nama tabel berubah, seperti ketika Anda membuat cadangan tabel atau melakukan point-in-time pemulihan. Ketika Anda mendekripsi atau memverifikasi tanda tangan item ini, Anda harus menggunakan konteks enkripsi DynamoDB yang sama yang digunakan untuk mengenkripsi dan menandatangani item, termasuk nama tabel asli. Nama tabel saat ini tidak diperlukan.

Saat Anda menggunakan `DynamoDBEncryptor`, Anda menyusun konteks enkripsi DynamoDB secara manual. Namun, jika Anda menggunakan `DynamoDBMapper`, `AttributeEncryptor` membuat konteks enkripsi DynamoDB untuk Anda, termasuk nama tabel saat ini. Untuk memerintahkan `AttributeEncryptor` agar membuat konteks enkripsi dengan nama tabel yang berbeda, gunakan `EncryptionContextOverrideOperator`.

Sebagai contoh, kode berikut menciptakan instans penyedia materi kriptografis (CMP) dan `DynamoDBEncryptor`. Kemudian kode itu memanggil metode `setEncryptionContextOverrideOperator` pada `DynamoDBEncryptor`. Kode itu menggunakan operator `overrideEncryptionContextTableName`, yang menimpa satu nama tabel. Ketika dikonfigurasi dengan cara ini, `AttributeEncryptor` membuat konteks enkripsi DynamoDB yang mencakup `newTableName` menggantikan `oldTableName`. Untuk contoh lengkap, lihat EncryptionContextOverridesWithDynamoDBMapper.java.

```
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, keyArn);
final DynamoDBEncryptor encryptor = DynamoDBEncryptor.getInstance(cmp);

encryptor.setEncryptionContextOverrideOperator(EncryptionContextOperators.overrideEncryptionCon
                oldTableName, newTableName));
```

Ketika Anda memanggil metode beban pada `DynamoDBMapper`, yang mendekripsi dan memverifikasi item, Anda menentukan nama tabel asli.

```
mapper.load(itemClass, DynamoDBMapperConfig.builder()

 .withTableNameOverride(DynamoDBMapperConfig.TableNameOverride.withTableNameReplacement(oldTabl
                .build());
```

Anda juga dapat menggunakan operator `overrideEncryptionContextTableNameUsingMap`, yang menimpa beberapa nama tabel.

Operator penimpa nama tabel biasanya digunakan ketika mendekripsi data dan memverifikasi tanda tangan. Namun, Anda dapat menggunakannya untuk menetapkan nama tabel dalam konteks enkripsi DynamoDB dengan nilai yang berbeda saat mengenkripsi dan menandatangani.

Jangan gunakan operator penimpa nama tabel jika Anda menggunakan `DynamoDBEncryptor`. Sebaliknya, buatlah konteks enkripsi dengan nama tabel asli dan kirimkan ke metode dekripsi.

Contoh kode untuk DynamoDB Encryption Client untuk Java

> **Note**
>
> Pustaka enkripsi sisi klien kami diubah namanya menjadi AWS Database Encryption SDK. Topik berikut memberikan informasi tentang versi 1. x —2. x dari DynamoDB Encryption Client untuk Java dan versi 1. x —3. x dari Klien Enkripsi DynamoDB untuk Python.

Untuk informasi selengkapnya, lihat SDK Enkripsi AWS Database untuk dukungan versi DynamoDB.

Contoh berikut menunjukkan cara menggunakan DynamoDB Encryption Client untuk Java guna melindungi item tabel DynamoDB dalam aplikasi Anda. Anda dapat menemukan lebih banyak contoh (dan berkontribusi sendiri) di direktori contoh aws-dynamodb-encryption-javarepositori di. GitHub

Topik

- Menggunakan Dynamo DBEncryptor
- Menggunakan Dynamo DBMapper

Menggunakan Dynamo DBEncryptor

Contoh ini menunjukkan cara menggunakan Dynamo tingkat rendah DBEncryptor dengan Direct KMS Provider. Penyedia KMS Langsung menghasilkan dan melindungi materi kriptografinya di bawah AWS KMS keyin AWS Key Management Service (AWS KMS) yang Anda tentukan.

Anda dapat menggunakan penyedia materi kriptografi (CMP) yang kompatibel dengan`DynamoDBEncryptor`, dan Anda dapat menggunakan Penyedia KMS Langsung dengan dan. `DynamoDBMapper` AttributeEncryptor

Lihat contoh kode lengkapnya: AwsKmsEncryptedItem.java

Langkah 1: Buat Penyedia KMS Langsung

Buat instance AWS KMS klien dengan wilayah yang ditentukan. Kemudian, gunakan instance klien untuk membuat instance Penyedia KMS Langsung dengan pilihan AWS KMS key Anda.

Contoh ini menggunakan Amazon Resource Name (ARN) untuk mengidentifikasi AWS KMS key, tetapi Anda dapat menggunakan pengidentifikasi kunci yang valid.

```
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);
```

Langkah 2: Buat item

Contoh ini mendefinisikan `record` HashMap yang mewakili item tabel sampel.

```
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"));
```

Langkah 3: Buat Dynamo DBEncryptor

Buat instans `DynamoDBEncryptor` dengan Penyedia KMS Langsung.

```
final DynamoDBEncryptor encryptor = DynamoDBEncryptor.getInstance(cmp);
```

Langkah 4: Buat konteks enkripsi DynamoDB

Konteks enkripsi DynamoDB berisi informasi tentang struktur tabel dan bagaimana tabel itu dienkripsi dan ditandatangani. Jika Anda menggunakan `DynamoDBMapper`, `AttributeEncryptor` membuat konteks enkripsi untuk Anda.

```
final String tableName = "testTable";

final EncryptionContext encryptionContext = new EncryptionContext.Builder()
    .withTableName(tableName)
    .withHashKeyName(partitionKeyName)
    .withRangeKeyName(sortKeyName)
    .build();
```

Langkah 5: Buat objek tindakan atribut

Tindakan atribut menentukan mana atribut item yang dienkripsi dan ditandatangani, mana yang hanya ditandatangani, dan mana yang tidak dienkripsi dan ditandatangani.

Di Java, untuk menentukan tindakan atribut, Anda membuat HashMap nama atribut dan pasangan `EncryptionFlags` nilai.

Misalnya, kode Java berikut membuat `actions` HashMap yang mengenkripsi dan menandatangani semua atribut dalam `record` item, kecuali untuk kunci partisi dan atribut kunci sortir, yang ditandatangani, tetapi tidak dienkripsi, dan `test` atribut, yang tidak ditandatangani atau dienkripsi.

```
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;
  }
}
```

Langkah 6: Enkripsi dan tanda tangani item

Untuk mengenkripsi dan menandatangani item tabel, panggil metode `encryptRecord` pada instans `DynamoDBEncryptor`. Tentukan item tabel (`record`), tindakan atribut (`actions`), dan konteks enkripsi (`encryptionContext`).

```
final Map<String, AttributeValue> encrypted_record = encryptor.encryptRecord(record,
 actions, encryptionContext);
```

Langkah 7: Masukkan item ke dalam tabel DynamoDB

Akhirnya, letakkan item yang dienkripsi dan ditandatangani ke dalam tabel DynamoDB.

```
final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();
ddb.putItem(tableName, encrypted_record);
```

Menggunakan Dynamo DBMapper

Contoh berikut menunjukkan kepada Anda cara menggunakan kelas bantuan DynamoDB Mapper dengan Penyedia KMS Langsung. Penyedia KMS Langsung menghasilkan dan melindungi materi kriptografinya di bawah AWS KMS keyin AWS Key Management Service (AWS KMS) yang Anda tentukan.

Anda dapat menggunakan penyedia materi kriptografis (CMP) yang kompatibel dengan `DynamoDBMapper`, dan Anda dapat menggunakan Penyedia KMS Langsung dengan `DynamoDBEncryptor` yang tingkatnya lebih rendah.

Lihat contoh kode lengkapnya: AwsKmsEncryptedObject.java

Langkah 1: Buat Penyedia KMS Langsung

Buat instance AWS KMS klien dengan wilayah yang ditentukan. Kemudian, gunakan instance klien untuk membuat instance Penyedia KMS Langsung dengan pilihan AWS KMS key Anda.

Contoh ini menggunakan Amazon Resource Name (ARN) untuk mengidentifikasi AWS KMS key, tetapi Anda dapat menggunakan pengidentifikasi kunci yang valid.

```
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);
```

Langkah 2: Buat DynamoDB Encryptor dan Dynamo DBMapper

Gunakan Penyedia KMS Langsung yang Anda buat di langkah sebelumnya untuk membuat instans DynamoDB Encryptor. Anda perlu membuat instans bagi DynamoDB Encryptor yang tingkatnya lebih rendah untuk menggunakan DynamoDB Mapper.

Berikutnya, membuat sebuah instans dari database DynamoDB Anda dan konfigurasi mapper, dan menggunakannya untuk membuat sebuah instance dari DynamoDB Mapper.

> **Important**
>
> Saat menggunakan `DynamoDBMapper` untuk menambah atau mengedit item yang ditandatangani (atau dienkripsi dan ditandatangani), atur konfigurasinya untuk menggunakan perilaku penympanan, seperti PUT, yang mencakup semua atribut, seperti

yang ditunjukkan dalam contoh berikut. Jika tidak, Anda mungkin tidak dapat mendekripsi data.

```
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));
```

Langkah 3: Tentukan tabel DynamoDB Anda

Selanjutnya, tentukan tabel DynamoDB Anda. Gunakan anotasi untuk menentukan tindakan atribut. Contoh ini membuat tabel DynamoDB, `ExampleTable`, dan kelas `DataPoJo` yang mewakili item tabel.

Dalam tabel contoh ini, atribut kunci utama akan ditandatangani tetapi tidak dienkripsi. Hal ini berlaku untuk `partition_attribute`, yang dianotasikan dengan `@DynamoDBHashKey`, dan `sort_attribute`, yang dianotasikan dengan `@DynamoDBRangeKey`.

Atribut yang dianotasikan dengan `@DynamoDBAttribute`, seperti `some numbers`, akan dienkripsi dan ditandatangani. Pengecualiannya mencakup atribut yang menggunakan anotasi enkripsi `@DoNotEncrypt` (hanya ditandatangani) atau `@DoNotTouch` (jangan dienkripsi atau ditandatangani) yang didefinisikan oleh DynamoDB Encryption Client. Sebagai contoh, karena atribut `leave me` memiliki anotasi `@DoNotTouch`, atribut ini tidak akan dienkripsi atau ditandatangani.

```
@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 +
 "]";
  }
}
```

Langkah 4: Enkripsi dan simpan item tabel

Sekarang, ketika Anda membuat item tabel dan menggunakan DynamoDB Mapper untuk menyimpannya, item secara otomatis dienkripsi dan ditandatangani sebelum ditambahkan ke tabel.

Contoh ini mendefinisikan item tabel yang disebut `record`. Sebelum disimpan dalam tabel, atributnya dienkripsi dan ditandatangani berdasarkan anotasi di kelas `DataPoJo`. Dalam hal ini, semua atribut kecuali `PartitionAttribute`, `SortAttribute`, dan `LeaveMe` dienkripsi dan ditandatangani. `PartitionAttribute` dan `SortAttributes` hanya ditandatangani. Atribut `LeaveMe` tidak dienkripsi atau ditandatangani.

Untuk mengenkripsi dan menandatangani item `record`, dan kemudian menambahkannya ke `ExampleTable`, panggil metode `save` kelas `DynamoDBMapper`. Karena DynamoDB Mapper Anda dikonfigurasi untuk menggunakan perilaku penyimpanan PUT, item tersebut menggantikan semua item dengan kunci utama yang sama, bukan memperbaruinya. Hal ini memastikan tanda tangan cocok dan Anda dapat mendekripsi item ketika Anda mendapatkannya dari tabel.

```
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);
```

DynamoDB Encryption Client untuk Python

> **Note**
>
> Pustaka enkripsi sisi klien kami diubah namanya menjadi AWS Database Encryption SDK. Topik berikut memberikan informasi tentang versi 1. x —2. x dari DynamoDB Encryption Client untuk Java dan versi 1. x —3. x dari Klien Enkripsi DynamoDB untuk Python. Untuk informasi selengkapnya, lihat SDK Enkripsi AWS Database untuk dukungan versi DynamoDB.

Topik ini menjelaskan cara menginstal dan menggunakan DynamoDB Encryption Client untuk Python. Anda dapat menemukan kode di aws-dynamodb-encryption-pythonrepositori GitHub, termasuk kode sampel lengkap dan teruji untuk membantu Anda memulai.

> **Note**
>
> Versi 1. x. x dan 2. x. x dari Klien Enkripsi DynamoDB untuk Python sedang end-of-support dalam fase efektif Juli 2022. Tingkatkan ke versi yang lebih baru sesegera mungkin.

Topik

- Prasyarat
- Penginstalan
- Menggunakan DynamoDB Encryption Client untuk Python
- Contoh kode untuk DynamoDB Encryption Client untuk Python

Prasyarat

Sebelum Anda menginstal Amazon DynamoDB Encryption Client untuk Python, pastikan Anda memiliki prasyarat berikut.

Versi Python yang didukung

Python 3.8 atau yang lebih baru diperlukan oleh Klien Enkripsi Amazon DynamoDB untuk Python versi 3.3.0 dan yang lebih baru. Untuk mengunduh Python, lihat Unduh Python.

Versi sebelumnya dari Amazon DynamoDB Encryption Client untuk Python mendukung Python 2.7 dan Python 3.4 dan yang lebih baru, tetapi kami menyarankan Anda menggunakan versi terbaru dari DynamoDB Encryption Client.

Alat instalasi pip untuk Python

Python 3.6 dan yang lebih baru menyertakan pip, meskipun Anda mungkin ingin memutakhirkannya. Untuk informasi selengkapnya tentang meningkatkan atau menginstal pip, lihat Instalasi dalam dokumentasi pip.

Penginstalan

Gunakan pip untuk menginstal Amazon DynamoDB Encryption Client, seperti yang ditunjukkan dalam contoh berikut.

Pasang versi terbaru

```
pip install dynamodb-encryption-sdk
```

Untuk detail lebih lanjut tentang menggunakan pip untuk menginstal dan meng-upgrade paket, lihat Menginstal Paket.

DynamoDB Encryption Client memerlukan pustaka kriptografi di semua platform. Semua versi pip menginstal dan membangun kriptografi pada pip. Windows 8.1 dan versi lebih baru menginstal dan membangun kriptografi di Linux. Jika Anda menggunakan versi sebelumnya dari pip dan lingkungan Linux Anda tidak memiliki alat yang diperlukan untuk membangun pustaka kriptografi, Anda perlu menginstalnya. Untuk informasi selengkapnya, lihat Membangun kriptografi di Linux.

Anda bisa mendapatkan versi pengembangan terbaru dari DynamoDB Encryption Client dari aws-dynamodb-encryption-pythonrepositori. GitHub

Setelah Anda menginstal DynamoDB Encryption Client, mulailah dengan melihat kode contoh Python dalam panduan ini.

Menggunakan DynamoDB Encryption Client untuk Python

> **Note**
>
> Pustaka enkripsi sisi klien kami diubah namanya menjadi AWS Database Encryption SDK. Topik berikut memberikan informasi tentang versi 1. x —2. x dari DynamoDB Encryption Client untuk Java dan versi 1. x —3. x dari Klien Enkripsi DynamoDB untuk Python. Untuk informasi selengkapnya, lihat SDK Enkripsi AWS Database untuk dukungan versi DynamoDB.

Topik ini menjelaskan beberapa fitur DynamoDB Encryption Client untuk Python yang mungkin tidak ditemukan dalam implementasi bahasa pemrograman lainnya. Fitur-fitur ini dirancang untuk membuatnya lebih mudah untuk menggunakan DynamoDB Encryption Client dengan cara yang paling aman. Jika Anda tidak memiliki kasus penggunaan yang tidak biasa, kami sarankan Anda menggunakannya.

Untuk detail tentang pemrograman dengan Klien Enkripsi DynamoDB, lihat contoh Python dalam panduan ini, contoh di repositori GitHub aktif, dan dokumentasi Python aws-dynamodb-encryption-python untuk Klien Enkripsi DynamoDB.

Topik

- Kelas pembantu klien
- TableInfo kelas
- Tindakan atribut di Python

Kelas pembantu klien

DynamoDB Encryption Client untuk Python termasuk beberapa kelas pembantu klien yang mencerminkan kelas-kelas Boto 3 untuk DynamoDB. Kelas-kelas pembantu ini dirancang untuk membuatnya lebih mudah untuk menambahkan enkripsi dan penandatanganan ke aplikasi DynamoDB yang ada dan menghindari masalah yang paling umum, sebagai berikut:

- Mencegah Anda mengenkripsi kunci utama dalam item Anda, baik dengan menambahkan tindakan penggantian untuk kunci utama ke AttributeActionsobjek, atau dengan melempar pengecualian jika `AttributeActions` objek Anda secara eksplisit memberi tahu klien untuk mengenkripsi kunci utama. Jika tindakan default di objek `AttributeActions` Anda adalah `DO_NOTHING`,

kelas pembantu klien menggunakan tindakan tersebut untuk kunci primer. Jika tidak, mereka menggunakan `SIGN_ONLY`.

- Buat TableInfo objek dan isi konteks enkripsi DynamoDB berdasarkan panggilan ke DynamoDB. Ini membantu untuk memastikan bahwa konteks enkripsi DynamoDB Anda akurat dan klien dapat mengidentifikasi kunci utama.
- Metode Support, seperti `put_item` dan `get_item`, yang secara transparan mengenkripsi dan mendekripsi tabel item ketika Anda menulis atau membaca dari tabel DynamoDB. Hanya metode `update_item` yang tidak didukung.

Anda dapat menggunakan kelas pembantu klien,alih-alih berinteraksi langsung dengan enkriptor item yang tingkatnya lebih rendah. Gunakan kelas-kelas ini kecuali Anda perlu mengatur opsi lanjutan dalam enkriptor item.

Kelas pembantu klien meliputi:

- EncryptedTableuntuk aplikasi yang menggunakan sumber daya Tabel di DynamoDB untuk memproses satu tabel pada satu waktu.
- EncryptedResourceuntuk aplikasi yang menggunakan kelas Service Resource di DynamoDB untuk pemrosesan batch.
- EncryptedClientuntuk aplikasi yang menggunakan klien tingkat rendah di DynamoDB.

Untuk menggunakan kelas pembantu klien, pemanggil harus memiliki izin untuk memanggil operasi DynamoDB pada tabel target DescribeTable.

TableInfo kelas

TableInfoKelas adalah kelas pembantu yang mewakili tabel DynamoDB, lengkap dengan bidang untuk kunci utama dan indeks sekunder. Ini membantu Anda untuk mendapatkan informasi akurat dan real-time tentang tabel.

Jika Anda menggunakankelas pembantu klien, itu menciptakan dan menggunakan objek `TableInfo` untuk Anda. Jika tidak, Anda dapat membuat satu secara eksplisit. Sebagai contoh, lihat Gunakan enkriptor item.

Ketika Anda memanggil `refresh_indexed_attributes` metode pada `TableInfo` objek, itu mengisi nilai properti objek dengan memanggil operasi DynamoDB DescribeTable. Kueri tabel jauh lebih dapat diandalkan daripada nama indeks hard-coding. Kelas `TableInfo` juga mencakup

sebuah properti `encryption_context_values` yang menyediakan nilai-nilai yang diperlukan untuk konteks enkripsi DynamoDB.

Untuk menggunakan `refresh_indexed_attributes` metode ini, pemanggil harus memiliki izin untuk memanggil operasi DescribeTableDynamoDB pada tabel target.

Tindakan atribut di Python

Tindakan atribut memberitahukan kepada enkriptor item tentang tindakan yang dilakukan pada setiap atribut item. Untuk menentukan tindakan atribut di Python, buat objek `AttributeActions` dengan tindakan default dan pengecualian untuk atribut tertentu. Nilai-nilai yang valid adalah untuk tindakan atribut yang didefinisikan dalam jenis `CryptoAction` yang disebutkan.

> **Important**
>
> Setelah Anda menggunakan tindakan atribut untuk mengenkripsi item tabel Anda, menambahkan atau menghapus atribut dari model data Anda dapat menyebabkan kesalahan validasi tanda tangan yang mencegah Anda mendekripsi data Anda. Untuk penjelasan detail, lihat Mengubah model data Anda.

```
DO_NOTHING = 0
SIGN_ONLY = 1
ENCRYPT_AND_SIGN = 2
```

Misalnya, objek `AttributeActions` ini menetapkan `ENCRYPT_AND_SIGN` sebagai default untuk semua atribut, dan menentukan pengecualian untuk atribut `ISBN` dan `PublicationYear`.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={
        'ISBN': CryptoAction.DO_NOTHING,
        'PublicationYear': CryptoAction.SIGN_ONLY
    }
)
```

Jika Anda menggunakan sebuah kelas pembantu klien, Anda tidak perlu menentukan tindakan atribut untuk atribut kunci primer. Kelas pembantu klien mencegah Anda mengenkripsi kunci utama Anda.

Jika Anda tidak menggunakan kelas pembantu klien dan tindakan default adalah `ENCRYPT_AND_SIGN`, Anda harus menentukan tindakan untuk kunci primer. Tindakan yang direkomendasikan untuk kunci primer adalah `SIGN_ONLY`. Untuk mempermudahnya, gunakan metode `set_index_keys`, yang menggunakan SIGN_ONLY untuk kunci primer, atau DO_NOTHING, ketika itu adalah tindakan default.

> **Warning**
>
> Jangan mengenkripsi atribut kunci utama. Atribut tersebut harus tetap dalam plaintext sehingga DynamoDB dapat menemukan item tanpa memindai keseluruhan tabel.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
)
actions.set_index_keys(*table_info.protected_index_keys())
```

Contoh kode untuk DynamoDB Encryption Client untuk Python

> **Note**
>
> Pustaka enkripsi sisi klien kami diubah namanya menjadi AWS Database Encryption SDK. Topik berikut memberikan informasi tentang versi 1. x —2. x dari DynamoDB Encryption Client untuk Java dan versi 1. x —3. x dari Klien Enkripsi DynamoDB untuk Python. Untuk informasi selengkapnya, lihat SDK Enkripsi AWS Database untuk dukungan versi DynamoDB.

Contoh berikut menunjukkan cara menggunakan DynamoDB Encryption Client untuk Python guna melindungi data DynamoDB dalam aplikasi Anda. Anda dapat menemukan lebih banyak contoh (dan berkontribusi sendiri) di direktori contoh aws-dynamodb-encryption-pythonrepositori di. GitHub

Topik

- Gunakan kelas pembantu EncryptedTable klien
- Gunakan enkriptor item

Gunakan kelas pembantu EncryptedTable klien

Contoh berikut menunjukkan cara menggunakan Provider KMS Langsungdengan kelas pembantu klien `EncryptedTable`. Contoh ini menggunakan penyedia bahan kriptografi yang sama seperti contoh Gunakan enkriptor item berikut. Namun, contoh tersebut menggunakan kelas `EncryptedTable` alih-alih berinteraksi langsung dengan enkriptor item yang tingkatnya lebih rendah.

Dengan membandingkan contoh-contoh ini, Anda dapat melihat pekerjaan yang kelas pembantu klien untuk Anda. Hal ini mencakup pembuatan konteks enkripsi DynamoDB dan memastikan atribut kunci primer selalu ditandatangani, tetapi tidak pernah dienkripsi. Untuk membuat konteks enkripsi dan menemukan kunci utama, kelas pembantu klien memanggil operasi DynamoDB DescribeTable. Untuk menjalankan kode ini, Anda harus memiliki izin untuk memanggil operasi ini.

Lihat contoh kode lengkap: aws_kms_encrypted_table.py

Langkah 1: Buat Tabel

Mulailah dengan membuat sebuah instans dari tabel DynamoDB standar dengan nama tabel.

```
table_name='test-table'
table = boto3.resource('dynamodb').Table(table_name)
```

Langkah 2: Buat penyedia bahan kriptografi

Buat instans penyedia bahan kriptografi (CMP) yang Anda pilih.

Contoh ini menggunakan Penyedia KMS Langsung, tetapi Anda dapat menggunakan CMP mana pun yang kompatibel. Untuk membuat Penyedia KMS Langsung, tentukan. AWS KMS key Contoh ini menggunakan Amazon Resource Name (ARN) dari AWS KMS key, tetapi Anda dapat menggunakan pengidentifikasi kunci yang valid.

```
kms_key_id='arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key_id)
```

Langkah 3: Buat objek tindakan atribut

Tindakan atribut memberitahukan kepada enkriptor item tentang tindakan yang dilakukan pada setiap atribut item. Objek `AttributeActions` dalam contoh ini mengenkripsi dan menandatangani semua item kecuali untuk atribut `test`, yang diabaikan.

Jangan tentukan tindakan atribut untuk atribut kunci primer saat Anda menggunakan kelas pembantu klien. Kelas `EncryptedTable` menandatangani, tetapi tidak pernah mengenkripsi, atribut kunci utama.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={'test': CryptoAction.DO_NOTHING}
)
```

Langkah 4: Buat tabel terenkripsi

Membuat tabel dienkripsi menggunakan tabel standar, Penyedia KMS Langsung, dan tindakan atribut. Langkah ini melengkapi konfigurasi.

```
encrypted_table = EncryptedTable(
    table=table,
    materials_provider=kms_cmp,
    attribute_actions=actions
)
```

Langkah 5: Masukkan item plaintext ke dalam tabel

Ketika Anda memanggil metode `put_item` di `encrypted_table`, item tabel Anda secara transparan dienkripsi, ditandatangani, dan ditambahkan ke tabel DynamoDB Anda.

Pertama, tentukan item tabel.

```
plaintext_item = {
    'partition_attribute': 'value1',
    'sort_attribute': 55
    'example': 'data',
    'numbers': 99,
    'binary': Binary(b'\x00\x01\x02'),
    'test': 'test-value'
}
```

Lalu, taruh di tabel.

```
encrypted_table.put_item(Item=plaintext_item)
```

Untuk mendapatkan item dari daftar tabel DynamoDB dalam bentuk terenkripsi, panggil metode `get_item` di objek `table`. Untuk mendapatkan item didekripsi, panggil metode `get_item` di objek `encrypted_table`.

Gunakan enkriptor item

Contoh ini menunjukkan kepada Anda bagaimana untuk berinteraksi langsung dengan enkriptor item di DynamoDB Encryption Client ketika mengenkripsi item tabel, alih-alih menggunakan kelas pembantu klien yang berinteraksi dengan enkriptor item untuk Anda.

Bila Anda menggunakan teknik ini, Anda membuat konteks enkripsi DynamoDB dan konfigurasi objek (`CryptoConfig`) secara manual. Anda juga mengenkripsi item dalam satu panggilan dan menempatkannya dalam tabel DynamoDB Anda dalam panggilan terpisah. Hal ini memungkinkan Anda untuk menyesuaikan panggilan `put_item` Anda dan menggunakan DynamoDB Encryption Client untuk mengenkripsi dan menandatangani data terstruktur yang tidak pernah dikirim ke DynamoDB.

Contoh ini menggunakan Penyedia KMS Langsung, tetapi Anda dapat menggunakan CMP kompatibel.

Lihat contoh kode lengkap: aws_kms_encrypted_table.py

Langkah 1: Buat Tabel

Mulailah dengan membuat sebuah instans dari sumber daya tabel DynamoDB standar dengan nama tabel.

```
table_name='test-table'
table = boto3.resource('dynamodb').Table(table_name)
```

Langkah 2: Buat penyedia bahan kriptografi

Buat instans penyedia bahan kriptografi (CMP) yang Anda pilih.

Contoh ini menggunakan Penyedia KMS Langsung, tetapi Anda dapat menggunakan CMP mana pun yang kompatibel. Untuk membuat Penyedia KMS Langsung, tentukan. AWS KMS key Contoh ini menggunakan Amazon Resource Name (ARN) dari AWS KMS key, tetapi Anda dapat menggunakan pengidentifikasi kunci yang valid.

```
kms_key_id='arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
```

```
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key_id)
```

Langkah 3: Gunakan TableInfo kelas pembantu

Untuk mendapatkan informasi tentang tabel dari DynamoDB, buat instance dari TableInfokelas helper. Apabila Anda bekerja secara langsung dengan enkriptor item, Anda perlu membuat instans `TableInfo` dan memanggil metodenya. Kelas pembantu klien melakukan ini untuk Anda.

`refresh_indexed_attributes`Metode `TableInfo` menggunakan operasi DescribeTableDynamoDB untuk mendapatkan informasi real-time dan akurat tentang tabel. Ini termasuk kunci primer dan indeks sekunder lokal dan global. Pemanggil perlu memiliki izin untuk memanggil `DescribeTable`.

```
table_info = TableInfo(name=table_name)
table_info.refresh_indexed_attributes(table.meta.client)
```

Langkah 4: Buat konteks enkripsi DynamoDB

Konteks enkripsi DynamoDB berisi informasi tentang struktur tabel dan bagaimana tabel itu dienkripsi dan ditandatangani. Contoh ini menciptakan konteks enkripsi DynamoDB secara eksplisit, karena berinteraksi dengan enkriptor item. Kelas pembantu klien membuat konteks enkripsi DynamoDB untuk Anda.

Untuk mendapatkan kunci partisi dan kunci sortir, Anda dapat menggunakan properti kelas TableInfopembantu.

```
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)
)
```

Langkah 5: Buat objek tindakan atribut

Tindakan atribut memberitahukan kepada enkriptor item tentang tindakan yang dilakukan pada setiap atribut item. Objek `AttributeActions` dalam contoh ini mengenkripsi dan

menandatangani semua item kecuali atribut kunci utama, yang ditandatangani, tetapi tidak dienkripsi, dan atribut `test`, yang diabaikan.

Ketika Anda berinteraksi langsung dengan item enkripsi dan tindakan default Anda adalah `ENCRYPT_AND_SIGN`, Anda harus menentukan tindakan alternatif untuk kunci primer. Anda dapat menggunakan metode `set_index_keys`, yang menggunakan `SIGN_ONLY` untuk kunci primer, atau menggunakan `DO_NOTHING` jika itu adalah tindakan default.

Untuk menentukan kunci utama, contoh ini menggunakan kunci indeks dalam TableInfoobjek, yang diisi oleh panggilan ke DynamoDB. Teknik ini lebih aman daripada nama kunci primer hard-coding.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={'test': CryptoAction.DO_NOTHING}
)
actions.set_index_keys(*table_info.protected_index_keys())
```

Langkah 6: Buat konfigurasi untuk item

Untuk mengonfigurasi Klien Enkripsi DynamoDB, gunakan objek yang baru saja Anda buat dalam CryptoConfigkonfigurasi untuk item tabel. Kelas pembantu klien membuat CryptoConfig untuk Anda.

```
crypto_config = CryptoConfig(
    materials_provider=kms_cmp,
    encryption_context=encryption_context,
    attribute_actions=actions
)
```

Langkah 7: Enkripsi item

Langkah ini mengenkripsi dan menandai item, tetapi tidak memasukkannya ke dalam tabel DynamoDB.

Ketika Anda menggunakan kelas pembantu klien, item Anda secara transparan dienkripsi dan ditandatangani, dan kemudian ditambahkan ke tabel DynamoDB Anda ketika Anda memanggil metode `put_item` kelas pembantu. Ketika Anda menggunakan item enkripsi secara langsung, enkripsi dan tindakan independen.

Pertama, buat item plaintext.

```
plaintext_item = {
    'partition_attribute': 'value1',
    'sort_key': 55,
    'example': 'data',
    'numbers': 99,
    'binary': Binary(b'\x00\x01\x02'),
    'test': 'test-value'
}
```

Kemudian, enkripsi dan tanda tangani. Metode `encrypt_python_item` memerlukan objek konfigurasi `CryptoConfig`.

```
encrypted_item = encrypt_python_item(plaintext_item, crypto_config)
```

Langkah 8: Masukkan item ke dalam tabel DynamoDB

Akhirnya, letakkan item yang dienkripsi dan ditandatangani ke dalam tabel DynamoDB.

```
table.put_item(Item=encrypted_item)
```

Untuk melihat item yang dienkripsi, panggil metode `get_item` pada objek `table` asli, bukan dari objek `encrypted_table`. Item didapatkan dari tabel DynamoDB tanpa memverifikasi dan mendekripsinya.

```
encrypted_item = table.get_item(Key=partition_key)['Item']
```

Gambar berikut menunjukkan bagian dari contoh item tabel yang dienkripsi dan ditandatangani.

Nilai atribut terenkripsi adalah data biner. Nama dan nilai atribut kunci primer (`partition_attribute` dan `sort_attribute`) dan atribut `test` tetap dalam plaintext. Output juga menunjukkan atribut yang berisi tanda tangan (`*amzn-ddb-map-sig*`) dan atribut deskripsi materi (`*amzn-ddb-map-desc*`).

```
{
    '*amzn-ddb-map-desc*': Binary(b'\x00\x00\x00\x00\x00\x00\x10amzn-ddb-env-alg\
\x00\x00\x00\xe0AQEBAHhA84wnXjEJdBbBBylRUFcZZK2j7xwh6UyLoL28nQ
+0FAAAAH4wfAYJKoZIhvcNAQcGoG8wbQIBADBoBgkqhkiG9w0BBwEwHgYJYIZIAWUDBAEuMBEEDPeFBydmoJD
izYl0R0C4M7wAK6E1/N/bgTmHI=\x00\x00\x00\x17amzn-ddb-map-signingAlg\x00\x00\x00\nHmacS
\x00\x00\x00\x11/CBC/PKCS5Padding\x00\x00\x00\x10amzn-ddb-sig-alg\x00\x00\x00\x0eHmac
\x00\x00\x00\x0faws-kms-ec-attr\x00\x00\x00\x06*keys*'),
    '*amzn-ddb-map-sig*': Binary(b"\xd3\xc6\xc7\n\xb7#\x13\xd1Y\xea\xe4.|^\xbd\xdf\xe
    'binary': Binary(b'!"\xc5\x92\xd7\x13\x1d\xe8Bs\x9b\x7f\xa8\x8e\x9c\xcf\x10\x1e\x
    'example': Binary(b"'b\x933\x9a+s\xf1\xd6a\xc5\xd5\x1aZ\xed\xd6\xce\xe9X\xf0T\xcb
    'numbers': Binary(b'\xd5\xa0\\d\xcc\x85\xf5\x1e\xb9-f!\xb9\xb8\x8a\x1aT\xbaq\xf7\
    'partition_attribute': 'value1',
    'sort_attribute': 55,
    'test': 'test-value'
}
```

Mengubah model data Anda

> **Note**
>
> Pustaka enkripsi sisi klien kami diubah namanya menjadi AWS Database Encryption SDK. Topik berikut memberikan informasi tentang versi 1. x —2. x dari DynamoDB Encryption Client untuk Java dan versi 1. x —3. x dari Klien Enkripsi DynamoDB untuk Python. Untuk informasi selengkapnya, lihat SDK Enkripsi AWS Database untuk dukungan versi DynamoDB.

Setiap kali mengenkripsi atau mendekripsi item, Anda perlu memberikan tindakan atribut yang memberi tahu DynamoDB Encryption Client atribut mana yang perlu dienkripsi dan ditandatangani, atribut mana yang perlu ditanandatangani (tapi tidak dienkripsi), dan mana yang diabaikan. Atribut tindakan tidak disimpan dalam item yang dienkripsi dan DynamoDB Encryption Client tidak memperbarui tindakan atribut Anda secara otomatis.

> **Important**
>
> Klien Enkripsi DynamoDB tidak mendukung enkripsi data tabel DynamoDB yang sudah ada dan tidak terenkripsi.

Setiap kali Anda mengubah model data Anda, yaitu, ketika Anda menambahkan atau menghapus atribut dari item tabel Anda, Anda berisiko mendapatkan masalah. Jika tindakan atribut yang Anda

tentukan tidak mencakup semua atribut dalam item, item tersebut mungkin tidak dienkripsi dan ditandatangani sesuai keinginan Anda. Lebih penting lagi, jika tindakan atribut yang Anda berikan saat mendekripsi item berbeda dari tindakan atribut yang Anda berikan saat mengenkripsi item, verifikasi tanda tangan mungkin gagal.

Sebagai contoh, jika tindakan atribut yang digunakan untuk mengenkripsi item memberitahu untuk menandatangani atribut `test`, tanda tangan dalam item akan mencakup atribut `test`. Namun, jika tindakan atribut yang digunakan untuk mendekripsi item tidak memperhitungkan atribut `test`, verifikasi akan gagal karena klien akan mencoba untuk memverifikasi tanda tangan yang tidak termasuk atribut `test`.

Ini adalah masalah ketika beberapa aplikasi membaca dan menulis item DynamoDB sama karena DynamoDB Encryption Client harus menghitung tanda tangan yang sama untuk item di semua aplikasi. Ini juga masalah untuk aplikasi terdistribusi karena perubahan dalam tindakan atribut harus menyebar ke semua host. Bahkan jika tabel DynamoDB Anda diakses oleh satu host dalam satu proses, membangun proses praktek terbaik akan membantu mencegah kesalahan jika proyek pernah menjadi lebih kompleks.

Untuk menghindari kesalahan validasi tanda tangan yang mencegah Anda membaca item tabel Anda, gunakan panduan berikut.

- Menambahkan atribut — Jika atribut baru mengubah tindakan atribut Anda, sepenuhnya men-deploy perubahan tindakan atribut sebelum menyertakan atribut baru dalam item.
- Menghapus atribut — Jika Anda berhenti menggunakan atribut dalam item Anda, jangan mengubah tindakan atribut Anda.
- Mengubah tindakan - Setelah Anda telah menggunakan konfigurasi tindakan atribut untuk mengenkripsi item tabel Anda, Anda tidak dapat dengan aman mengubah tindakan default atau tindakan untuk atribut yang ada tanpa mereenkripsi setiap item dalam tabel Anda.

Kesalahan validasi tanda tangan bisa sangat sulit diatasi, jadi pendekatan terbaik adalah mencegahnya.

Topik

- Menambahkan atribut
- Menghapus atribut

Menambahkan atribut

Saat menambahkan atribut baru ke item tabel, Anda mungkin perlu mengubah tindakan atribut Anda. Untuk mencegah kesalahan validasi tanda tangan, sebaiknya Anda menerapkan perubahan ini dalam proses dua tahap. Verifikasi bahwa tahap pertama selesai sebelum memulai tahap kedua.

1. Ubah tindakan atribut di semua aplikasi yang membaca atau menulis ke tabel. Deploy perubahan ini dan konfirmasi bahwa pembaruan telah disebarkan ke semua host tujuan.
2. Tulis nilai ke atribut baru dalam item tabel Anda.

Pendekatan dua tahap ini memastikan bahwa semua aplikasi dan host memiliki tindakan atribut yang sama, dan akan menghitung tanda tangan yang sama, sebelum menemui atribut baru. Hal ini penting bahkan ketika tindakan untuk atribut adalah Jangan lakukan apa-apa (jangan mengenkripsi atau menandatangani), karena default untuk beberapa enkriptor adalah mengenkripsi dan menandatangani.

Contoh berikut menunjukkan kode untuk tahap pertama dalam proses ini. Atribut item baru ditambahkan, `link`, yang menyimpan link ke item tabel lain. Karena link ini harus tetap dalam teks biasa, contoh menetapkan itu tindakan tanda-saja. Setelah sepenuhnya men-deploy perubahan ini dan kemudian memverifikasi bahwa semua aplikasi dan host memiliki tindakan atribut baru, Anda dapat mulai menggunakan atribut `link` dalam item tabel Anda.

Java DynamoDB Mapper

Saat menggunakan `DynamoDB Mapper` dan `AttributeEncryptor` secara default, semua atribut dienkripsi dan ditandatangani kecuali kunci utama, yang ditandatangani tetapi tidak dienkripsi. Untuk menentukan tindakan tanda-saja, gunakan anotasi `@DoNotEncrypt`.

Contoh ini menggunakan anotasi `@DoNotEncrypt` untuk atribut `link` baru.

```
@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

Dalam enkriptor DynamoDB di level yang lebih rendah, Anda mesti menetapkan tindakan untuk setiap atribut. Contoh ini menggunakan pernyataan switch yang mana default adalah `encryptAndSign` dan pengecualian dispesifikasikan untuk kunci partisi, kunci penyortiran, dan atribut `link` baru. Dalam contoh ini, jika kode atribut tautan tidak sepenuhnya di-deploy sebelum digunakan, atribut tautan akan dienkripsi dan ditandatangani oleh beberapa aplikasi, tetapi hanya ditandatangani oleh orang lain.

```
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

Dalam DynamoDB Encryption Client untuk Python, Anda dapat menentukan tindakan default untuk semua atribut dan kemudian menentukan pengecualian.

Jika Anda menggunakan sebuah kelas pembantu klien Python, Anda tidak perlu menentukan tindakan atribut untuk atribut kunci primer. Kelas pembantu klien mencegah Anda mengenkripsi kunci utama Anda. Namun, jika Anda tidak menggunakan kelas pembantu klien, Anda harus mengatur tindakan SIGN_ONLY pada kunci partisi dan kunci sortir Anda. Jika Anda secara tidak sengaja mengenkripsi kunci partisi atau sortir Anda, Anda tidak akan dapat memulihkan data tanpa pemindaian tabel lengkap.

Contoh ini menentukan pengecualian untuk atribut `link` baru, yang mendapat tindakan `SIGN_ONLY`.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={
      'example': CryptoAction.DO_NOTHING,
      'link': CryptoAction.SIGN_ONLY
    }
)
```

Menghapus atribut

Jika Anda tidak lagi membutuhkan atribut dalam item yang telah dienkripsi dengan DynamoDB Encryption Client, Anda dapat berhenti menggunakan atribut. Namun, jangan menghapus atau mengubah tindakan untuk atribut tersebut. Jika Anda melakukannya, dan kemudian menemukan item dengan atribut itu, tanda tangan yang dihitung untuk item tidak akan cocok dengan tanda tangan asli, dan validasi tanda tangan akan gagal.

Meskipun Anda mungkin tergoda untuk menghapus semua jejak atribut dari kode Anda, tambahkan komentar bahwa item tidak lagi digunakan, alih-alih menghapusnya. Bahkan jika Anda melakukan pemindaian tabel secara penuh untuk menghapus semua instans atribut, item yang dienkripsi dengan atribut yang mungkin disembunyikan atau dalam proses di suatu tempat di konfigurasi Anda.

Memecahkan masalah dalam aplikasi DynamoDB Encryption Client Anda

> **Note**
>
> Pustaka enkripsi sisi klien kami diubah namanya menjadi AWS Database Encryption SDK. Topik berikut memberikan informasi tentang versi 1. x —2. x dari DynamoDB Encryption Client untuk Java dan versi 1. x —3. x dari Klien Enkripsi DynamoDB untuk Python. Untuk informasi selengkapnya, lihat SDK Enkripsi AWS Database untuk dukungan versi DynamoDB.

Bagian ini menjelaskan masalah yang mungkin Anda alami saat menggunakan DynamoDB Encryption Client dan menawarkan saran-saran untuk menyelesaikannya.

Untuk memberikan umpan balik tentang Klien Enkripsi DynamoDB, ajukan masalah di aws-dynamodb-encryption-javaatau repositori. aws-dynamodb-encryption-python GitHub

Untuk memberikan umpan balik tentang dokumentasi ini, gunakan tautan umpan balik pada halaman mana pun.

Topik

- Akses ditolak
- Verifikasi tanda tangan gagal
- Masalah dengan tabel global versi lama
- Kinerja yang buruk dari Penyedia Terbaru

Akses ditolak

Masalah: Aplikasi Anda ditolak aksesnya ke sumber daya yang dibutuhkan.

Saranan: Pelajari tentang izin yang diperlukan dan tambahkan izin tersebut ke konteks keamanan di mana aplikasi Anda dijalankan.

Detail

Untuk menjalankan aplikasi yang menggunakan pustaka DynamoDB Encryption Client, pemanggil harus memiliki izin untuk menggunakan komponennya. Jika tidak, aksesnya ke elemen yang dibutuhkan akan ditolak.

- DynamoDB Encryption Client tidak memerlukan akun Amazon Web Services (AWS) atau tergantung pada layanan AWS mana pun. Namun, jika aplikasi Anda menggunakan AWS, Anda memerlukan Akun AWS dan pengguna yang memiliki izin untuk menggunakan akun tersebut.
- DynamoDB Encryption Client tidak memerlukan Amazon DynamoDB. Namun, Jika aplikasi yang menggunakan klien membuat tabel DynamoDB, menempatkan item ke dalam tabel, atau mendapatkan item dari tabel, pemanggil harus memiliki izin untuk menggunakan operasi DynamoDB yang diperlukan dalam Anda. Akun AWS Untuk detailnya, lihat topik kontrol akses di Panduan Developer Amazon DynamoDB.
- Jika aplikasi Anda menggunakan class client helper di DynamoDB Encryption Client untuk Python, pemanggil harus memiliki izin untuk memanggil operasi DynamoDB. DescribeTable
- Klien Enkripsi DynamoDB tidak AWS Key Management Service memerlukan ().AWS KMS Namun, jika aplikasi Anda menggunakan Penyedia Materi KMS Langsung, atau menggunakan Penyedia Terbaru dengan toko penyedia yang menggunakan AWS KMS, penelepon harus memiliki izin untuk menggunakan AWS KMS GenerateDataKeydan Mendekripsi operasi.

Verifikasi tanda tangan gagal

Masalah: Item tidak dapat didekripsi karena verifikasi tanda tangan gagal. Item juga mungkin tidak dienkripsi dan ditandatangani sesuai keinginan Anda.

Saranan: Pastikan bahwa tindakan atribut yang Anda berikan mencakup semua atribut dalam item. Saat mendekripsi item, pastikan untuk memberikan tindakan atribut yang sesuai dengan tindakan yang digunakan untuk mengenkripsi item.

Detail

Tindakan atribut yang Anda berikan memberi tahu DynamoDB Encryption Client atribut mana yang perlu dienkripsi dan ditandatangani, atribut mana yang perlu ditanandatangani (tapi tidak dienkripsi), dan mana yang diabaikan.

Jika tindakan atribut yang Anda tentukan tidak mencakup semua atribut dalam item, item mungkin tidak dienkripsi dan ditandatangani sesuai keinginan Anda. Jika tindakan atribut yang Anda berikan saat mendekripsi item berbeda dari tindakan atribut yang Anda berikan saat mengenkripsi item, verifikasi tanda tangan mungkin gagal. Ini adalah masalah khusus untuk aplikasi terdistribusi di mana tindakan atribut baru mungkin tidak disebarkan ke semua host.

Kesalahan validasi tanda tangan sulit untuk diselesaikan. Untuk membantu mencegahnya, lakukan tindakan pencegahan ekstra saat mengubah model data Anda. Untuk detailnya, lihat Mengubah model data Anda.

Masalah dengan tabel global versi lama

Masalah: Item dalam tabel global Amazon DynamoDB versi lama tidak dapat didekripsi karena verifikasi tanda tangan gagal.

Saran: Tetapkan tindakan atribut agar bidang replikasi yang dicadangkan tidak dienkripsi atau ditandatangani.

Detail

Anda dapat menggunakan Klien Enkripsi DynamoDB dengan tabel global DynamoDB. Kami menyarankan Anda menggunakan tabel global dengan kunci KMS Multi-wilayah dan mereplikasi kunci KMS ke semua Wilayah AWS tempat tabel global direplikasi.

Dimulai dengan tabel global versi 2019.11.21, Anda dapat menggunakan tabel global dengan Klien Enkripsi DynamoDB tanpa konfigurasi khusus. Namun, jika Anda menggunakan tabel global versi 2017.11.29, Anda harus memastikan bahwa bidang replikasi yang dicadangkan tidak dienkripsi atau ditandatangani.

Jika Anda menggunakan tabel global versi 2017.11.29, Anda harus mengatur tindakan atribut untuk atribut berikut `DO_NOTHING` di `@DoNotTouch`Java atau Python.

- `aws:rep:deleting`
- `aws:rep:updatetime`
- `aws:rep:updateregion`

Jika Anda menggunakan versi lain dari tabel global, tidak ada tindakan yang diperlukan.

Kinerja yang buruk dari Penyedia Terbaru

Masalah: Aplikasi Anda kurang responsif, terutama setelah diperbarui ke versi DynamoDB Encryption Client yang lebih baru.

Saran: Sesuaikan time-to-live nilai dan ukuran cache.

Detail

Penyedia Terbaru dirancang untuk meningkatkan kinerja aplikasi yang menggunakan DynamoDB Encryption Client dengan mengizinkan penggunaan kembali secara terbatas materi kriptografis. Bila Anda mengonfigurasi Penyedia Terbaru untuk aplikasi Anda, Anda harus menyeimbangkan peningkatan kinerja dengan masalah keamanan yang timbul dari caching dan penggunaan kembali.

Dalam versi yang lebih baru dari DynamoDB Encryption Client, time-to-live nilai (TTL) menentukan berapa lama penyedia materi kriptografi cache () dapat digunakan. CMPs TTL juga menentukan seberapa sering Penyedia Terbaru memeriksa versi baru dari CMP.

Jika TTL terlalu lama, aplikasi Anda mungkin melanggar aturan bisnis atau standar keamanan Anda. Jika TTL terlalu singkat, panggilan berulang ke toko penyedia dapat menyebabkan toko penyedia Anda untuk membatasi permintaan dari aplikasi Anda dan aplikasi lain yang juga menggunakan akun layanan Anda. Untuk mengatasi masalah ini, sesuaikan TTL dan ukuran cache ke nilai yang memenuhi sasaran latensi dan ketersediaan Anda serta sesuai dengan standar keamanan Anda. Untuk detailnya, lihat Menetapkan time-to-live nilai.

Ganti nama Klien Enkripsi Amazon DynamoDB

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Pada tanggal 9 Juni 2023, pustaka enkripsi sisi klien kami diubah namanya menjadi Database Encryption SDK. AWS SDK Enkripsi AWS Database kompatibel dengan Amazon DynamoDB. Itu dapat mendekripsi dan membaca item yang dienkripsi oleh Klien Enkripsi DynamoDB lama. Untuk informasi selengkapnya tentang versi Klien Enkripsi DynamoDB lama, lihat. AWS SDK Enkripsi Database untuk dukungan versi DynamoDB

AWS Database Encryption SDK menyediakan versi 3. x dari pustaka enkripsi sisi klien Java untuk DynamoDB, yang merupakan penulisan ulang utama Klien Enkripsi DynamoDB untuk Java. Ini mencakup banyak pembaruan, seperti format data terstruktur baru, dukungan multitenancy yang ditingkatkan, perubahan skema yang mulus, dan dukungan enkripsi yang dapat dicari.

Untuk mempelajari lebih lanjut tentang fitur baru yang diperkenalkan dengan AWS Database Encryption SDK, lihat topik berikut.

Enkripsi yang dapat dicari

Anda dapat merancang database yang dapat mencari catatan terenkripsi tanpa mendekripsi seluruh database. Bergantung pada model ancaman dan persyaratan kueri, Anda dapat menggunakan enkripsi yang dapat dicari untuk melakukan penelusuran yang sama persis atau kueri kompleks yang lebih disesuaikan pada catatan terenkripsi Anda.

Gantungan kunci

AWS Database Encryption SDK menggunakan keyrings untuk melakukan enkripsi amplop. Keyrings menghasilkan, mengenkripsi, dan mendekripsi kunci data yang melindungi catatan Anda. AWS Database Encryption SDK mendukung AWS KMS keyrings yang menggunakan enkripsi simetris atau RSA asimetris AWS KMS keysuntuk melindungi kunci data Anda, dan gantungan kunci AWS KMS hirarkis yang memungkinkan Anda melindungi materi kriptografi Anda di bawah kunci KMS enkripsi simetris tanpa menelepon setiap kali Anda mengenkripsi atau mendekripsi catatan. AWS KMS Anda juga dapat menentukan bahan kunci Anda sendiri dengan gantungan kunci Raw AES dan gantungan kunci Raw RSA.

Perubahan skema yang mulus

Saat Anda mengonfigurasi SDK Enkripsi AWS Database, Anda memberikan tindakan kriptografi yang memberi tahu klien bidang mana yang akan dienkripsi dan ditandatangani, bidang mana yang akan ditandatangani (tetapi tidak dienkripsi), dan mana yang harus diabaikan. Setelah Anda menggunakan AWS Database Encryption SDK untuk melindungi catatan Anda, Anda masih dapat membuat perubahan pada model data Anda. Anda dapat memperbarui tindakan kriptografi Anda, seperti menambahkan atau menghapus bidang terenkripsi, dalam satu penerapan.

Konfigurasikan tabel DynamoDB yang ada untuk enkripsi sisi klien

Versi lama dari Klien Enkripsi DynamoDB dirancang untuk diimplementasikan dalam tabel baru yang tidak terisi. Dengan SDK Enkripsi AWS Database untuk DynamoDB, Anda dapat memigrasikan tabel Amazon DynamoDB yang ada ke versi 3. x dari pustaka enkripsi sisi klien Java untuk DynamoDB.

Referensi

Pustaka enkripsi sisi klien kami diubah namanya menjadi SDK Enkripsi AWS Database. Panduan pengembang ini masih memberikan informasi tentang Klien Enkripsi DynamoDB.

Topik berikut memberikan rincian teknis untuk AWS Database Encryption SDK.

Format deskripsi bahan

Deskripsi materi berfungsi sebagai header untuk catatan terenkripsi. Saat Anda mengenkripsi dan menandatangani bidang dengan SDK Enkripsi AWS Database, enkripsi mencatat deskripsi materi saat merakit materi kriptografi dan menyimpan deskripsi materi di bidang baru (`aws_dbe_head`) yang ditambahkan enkripsi ke catatan Anda. Deskripsi materi adalah struktur data berformat portabel yang berisi kunci data terenkripsi dan informasi tentang bagaimana catatan dienkripsi dan ditandatangani. Tabel berikut menjelaskan nilai-nilai yang membentuk deskripsi material. Byte ditambahkan dalam urutan yang ditunjukkan.

Nilai	Panjangnya dalam byte
Version	1
Signatures Enabled	1
Record ID	32
Encrypt Legend	Variabel
Encryption Context Length	2
???	Variabel
Encrypted Data Key Count	1
Encrypted Data Keys	Variabel
Record Commitment	1

Versi

Versi format `aws_dbe_head` bidang ini.

Tanda Tangan Diaktifkan

Mengkodekan apakah tanda tangan digital ECDSA diaktifkan untuk catatan ini.

Nilai byte	Arti
`0x01`	Tanda tangan digital ECDSA diaktifkan (default)
`0x00`	Tanda tangan digital ECDSA dinonaktifkan

Rekam ID

Nilai 256-bit yang dihasilkan secara acak yang mengidentifikasi catatan. ID Rekaman:

- Mengidentifikasi catatan terenkripsi secara unik.
- Mengikat deskripsi material ke catatan terenkripsi.

Enkripsi Legenda

Deskripsi serial bidang yang diautentikasi dienkripsi. Encrypt Legend digunakan untuk menentukan bidang apa metode dekripsi harus mencoba untuk mendekripsi.

Nilai byte	Arti
`0x65`	`ENCRYPT_AND_SIGN`
`0x73`	`SIGN_ONLY`

Encrypt Legend diserialisasikan sebagai berikut:

1. Secara leksikografis dengan urutan byte yang mewakili jalur kanonik mereka.
2. Untuk setiap bidang, secara berurutan, tambahkan salah satu nilai byte yang ditentukan di atas untuk menunjukkan apakah bidang itu harus dienkripsi.

Panjang Konteks Enkripsi

Panjang konteks enkripsi. Ini adalah nilai 2-byte yang ditafsirkan sebagai integer unsigned 16-bit. Panjang maksimum adalah 65.535 byte.

Konteks Enkripsi

Satu set pasangan nama-nilai yang berisi data otentikasi tambahan yang arbitrer dan non-rahasia.

Ketika tanda tangan digital ECDSA diaktifkan, konteks enkripsi berisi pasangan kunci-nilai. `{"aws-crypto-footer-ecdsa-key": Qtxt}` `Qtxt`mewakili titik kurva elips yang `Q` dikompresi menurut SEC 1 versi 2.0 dan kemudian dikodekan base64.

Hitungan Kunci Data Terenkripsi

Jumlah kunci data terenkripsi. Ini adalah nilai 1-byte ditafsirkan sebagai 8-bit unsigned integer yang menentukan jumlah kunci data terenkripsi. Jumlah maksimum kunci data terenkripsi di setiap catatan adalah 255.

Kunci Data Terenkripsi

Urutan kunci data terenkripsi. Panjang urutan ditentukan oleh jumlah kunci data terenkripsi dan panjang masing-masing. Urutan berisi setidaknya satu kunci data terenkripsi.

Tabel berikut menjelaskan bidang yang membentuk setiap kunci data terenkripsi. Byte ditambahkan dalam urutan yang ditunjukkan.

Struktur Kunci Data Terenkripsi

Bidang	Panjangnya dalam byte
Key Provider ID Length	2
Key Provider ID	Variabel. Sama dengan nilai yang ditentuka n dalam 2 byte sebelumnya (Panjang ID Penyedia Kunci).
Key Provider Information Length	2
Key Provider Information	Variabel. Sama dengan nilai yang ditentukan dalam 2 byte sebelumnya (Panjang Informasi Penyedia Kunci).

Bidang	Panjangnya dalam byte
Encrypted Data Key Length	2
Encrypted Data Key	Variabel. Sama dengan nilai yang ditentukan dalam 2 byte sebelumnya (Panjang Kunci Data Terenkripsi).

Panjang ID Penyedia Kunci

Panjang pengenal penyedia kunci. Ini adalah nilai 2-byte yang ditafsirkan sebagai integer unsigned 16-bit yang menentukan jumlah byte yang berisi ID penyedia kunci.

ID Penyedia Kunci

Pengidentifikasi penyedia kunci. Ini digunakan untuk menunjukkan penyedia kunci data terenkripsi dan dimaksudkan untuk dapat diperluas.

Panjang Informasi Penyedia Kunci

Panjang informasi penyedia kunci. Ini adalah nilai 2-byte yang ditafsirkan sebagai integer unsigned 16-bit yang menentukan jumlah byte yang berisi informasi penyedia kunci.

Informasi Penyedia Utama

Informasi penyedia utama. Itu ditentukan oleh penyedia kunci.

Bila Anda menggunakan AWS KMS keyring, nilai ini berisi Amazon Resource Name (ARN) dari. AWS KMS key

Panjang Kunci Data Terenkripsi

Panjang kunci data terenkripsi. Ini adalah nilai 2-byte ditafsirkan sebagai 16-bit unsigned integer yang menentukan jumlah byte yang berisi kunci data terenkripsi.

Kunci Data Terenkripsi

Kunci data terenkripsi. Ini adalah kunci data yang dienkripsi oleh penyedia kunci.

Rekam Komitmen

Hash 256-bit Hash Based Message Authentication Code (HMAC) yang berbeda dihitung atas semua byte deskripsi material sebelumnya menggunakan kunci komit.

AWS KMS Rincian teknis keyring hierarkis

Keyring AWS KMS Hierarkis menggunakan kunci data unqiue untuk mengenkripsi setiap bidang dan mengenkripsi setiap kunci data dengan kunci pembungkus unik yang berasal dari kunci cabang aktif. Ini menggunakan derivasi kunci dalam mode counter dengan fungsi pseudorandom dengan HMAC SHA-256 untuk menurunkan kunci pembungkus 32 byte dengan input berikut.

- Garam acak 16 byte
- Kunci cabang aktif
- Nilai yang dikodekan UTF-8 untuk pengenal penyedia kunci "" aws-kms-hierarchy

Keyring Hierarkis menggunakan kunci pembungkus turunan untuk mengenkripsi salinan kunci data teks biasa menggunakan AES-GCM-256 dengan tag otentikasi 16 byte dan input berikut.

- Kunci pembungkus turunan digunakan sebagai kunci sandi AES-GCM
- Kunci data digunakan sebagai pesan AES-GCM
- Vektor inisialisasi acak 12 byte (IV) digunakan sebagai AES-GCM IV
- Data otentikasi tambahan (AAD) yang berisi nilai serial berikut.

Nilai	Panjangnya dalam byte	Ditafsirkan sebagai
"aws-kms-hierarchy"	17	UTF-8 dikodekan
Pengidentifikasi kunci cabang	Variabel	UTF-8 dikodekan
Versi kunci cabang	16	UTF-8 dikodekan
Konteks enkripsi	Variabel	Pasangan nilai kunci yang dikodekan UTF-8

Riwayat dokumen untuk Panduan Pengembang SDK Enkripsi AWS Database

Tabel berikut menjelaskan perubahan signifikan pada dokumentasi ini. Selain perubahan besar ini, kami juga sering memperbarui dokumentasi untuk memperbaiki deskripsi dan contoh, serta membahas umpan balik yang Anda kirimkan kepada kami. Untuk diberitahu tentang perubahan signifikan, berlangganan umpan RSS.

Perubahan	Deskripsi	Tanggal
Fitur baru	Menambahkan dokumentasi untuk keyring AWS KMS ECDH dan keyring ECDH mentah.	Juni 17, 2024
Rilis Ketersediaan Umum (GA)	Memperkenalkan dukungan untuk pustaka enkripsi sisi klien .NET untuk DynamoDB.	Januari 17, 2024
Rilis Ketersediaan Umum (GA)	Diperbarui dokumentasi untuk rilis GA versi 3. x dari pustaka enkripsi sisi klien Java untuk DynamoDB. **Warning** Kunci cabang yang dibuat selama rilis pratinjau pengembang tidak lagi didukung.	Juli 24, 2023
Rebrand dari DynamoDB Encryption Client	Pustaka enkripsi sisi klien diubah namanya menjadi AWS Database Encryption SDK.	9 Juni 2023

Rilis pratinjau	Ditambahkan dan diperbaru i dokumentasi untuk versi 3. x dari pustaka enkripsi sisi klien Java untuk DynamoDB, yang mencakup format data terstruktur baru, dukungan multitenancy yang ditingkat kan, perubahan skema yang mulus, dan dukungan enkripsi yang dapat dicari.	9 Juni 2023
Perubahan dokumentasi	Ganti AWS Key Managemen t Service istilah customer master key (CMK) dengan AWS KMS keydan kunci KMS.	Agustus 30, 2021
Fitur baru	Menambahkan dukungan untuk AWS Key Managemen t Service (AWS KMS) kunci Multi-wilayah. Tombol Multi-Region adalah AWS KMS kunci Wilayah AWS yang berbeda yang dapat digunakan secara bergantian karena memiliki ID kunci dan bahan kunci yang sama.	8 Juni 2021
Contoh baru	Ditambahkan contoh menggunakan Dynamo DBMapper di Java.	6 September 2018
Dukungan Python	Dukungan tambahan untuk Python, selain Java.	2 Mei 2018
Rilis awal	Rilis awal dokumentasi ini.	2 Mei 2018

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.