



Guide du développeur

AWS SDK de chiffrement de base de données



AWS SDK de chiffrement de base de données: Guide du développeur

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

Les marques et la présentation commerciale d'Amazon ne peuvent être utilisées en relation avec un produit ou un service qui n'est pas d'Amazon, d'une manière susceptible de créer une confusion parmi les clients, ou d'une manière qui dénigre ou discrédite Amazon. Toutes les autres marques commerciales qui ne sont pas la propriété d'Amazon appartiennent à leurs propriétaires respectifs, qui peuvent ou non être affiliés ou connectés à Amazon, ou sponsorisés par Amazon.

Table of Contents

Qu'est-ce que le SDK AWS de chiffrement des bases de données ?	1
Développé dans des référentiels open source	3
Support et maintenance	3
Envoyer un commentaire	4
Concepts	4
Chiffrement d'enveloppe	5
Clé de données	7
Clé d'emballage	8
Porte-clés	8
Actions cryptographiques	9
Description du matériau	10
Contexte de chiffrement	11
Gestionnaire de matériaux de chiffrement	11
Chiffrement symétrique et asymétrique	12
Engagement clé	12
Signatures numériques	13
Comment ça marche	15
Chiffrer et signer	15
Déchiffrer et vérifier	17
Suites d'algorithmes prises en charge	18
Suite d'algorithmes par défaut	21
AES-GCM sans signatures numériques ECDSA	21
Interaction avec AWS KMS	24
Configuration du kit SDK	26
Sélection d'un langage de programmation	26
Sélection des clés d'emballage	26
Création d'un filtre de découverte	28
Utilisation de bases de données mutualisées	29
Création de balises signées	30
Principaux magasins	38
Terminologie et concepts clés du magasin	38
Implémentation des autorisations avec le moindre privilégié	39
Créer un magasin de clés	40
Configurer les actions clés du magasin	42

Configurez les actions clés de votre boutique	43
Création de clés de branche	45
Faites pivoter votre clé de branche active	49
Porte-clés	52
Fonctionnement des porte-clés	53
AWS KMS porte-clés	54
Autorisations requises pour les AWS KMS porte-clés	55
Identification AWS KMS keys dans un AWS KMS porte-clés	56
Création d'un AWS KMS porte-clés	57
Utilisation de plusieurs régions AWS KMS keys	60
Utilisation d'un porte-clés AWS KMS Discovery	62
Utiliser un porte-clés de découverte AWS KMS régional	65
AWS KMS Porte-clés hiérarchiques	67
Comment ça marche	69
Conditions préalables	71
Autorisations requises	72
Choisissez un cache	72
Création d'un trousseau de clés hiérarchique	82
Utilisation du trousseau de clés hiérarchique pour un chiffrement consultable	88
AWS KMS Porte-clés ECDH	93
Autorisations requises pour les AWS KMS porte-clés ECDH	94
Création d'un AWS KMS porte-clés ECDH	94
Création d'un AWS KMS porte-clés de découverte ECDH	98
Porte-clés AES brut	101
Porte-clés RSA bruts	103
Porte-clés ECDH bruts	107
Création d'un porte-clés ECDH brut	108
Porte-clés multiples	117
Chiffrement consultable	122
Les balises sont-elles adaptées à mon ensemble de données ?	123
Scénario de chiffrement consultable	126
Balises	128
Balises standard	129
Balises composées	130
Balises de planification	131
Considérations relatives aux bases de données mutualisées	133

Choisir un type de balise	133
Choix de la longueur d'une balise	140
Choisir un nom de balise	147
Configuration des balises	148
Configuration des balises standard	149
Configuration de balises composées	158
Exemples de configuration	169
Utilisation de balises	173
Interrogation de balises	176
Chiffrement consultable pour les bases de données mutualisées	178
Interrogation de balises dans une base de données mutualisée	181
Amazon DynamoDB	183
Chiffrement côté client et côté serveur	184
Quels sont les champs chiffrés et signés ?	186
Chiffrement des valeurs d'attribut	187
Signature de l'élément	188
Chiffrement consultable dans DynamoDB	188
Configuration des index secondaires avec des balises	189
Tester les sorties des balises	190
Mettre à jour votre modèle de données	196
Ajouter ENCRYPT_AND_SIGN de SIGN_ONLY nouveaux SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT attributs et	198
Supprimer les attributs existants	199
Remplacer un ENCRYPT_AND_SIGN attribut existant par SIGN_ONLY ou SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT	199
Modifier un SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT attribut SIGN_ONLY ou un existant en ENCRYPT_AND_SIGN	200
Ajouter un nouvel DO_NOTHING attribut	201
Modifier un SIGN_ONLY attribut existant en SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT	202
Modifier un SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT attribut existant en SIGN_ONLY	202
Langages de programmation	203
Java	203
.NET	240
Rust	257

Héritée	263
AWS SDK de chiffrement de base de données pour la prise en charge des versions	
DynamoDB	264
Comment ça marche	264
Concepts	268
Fournisseur de matériel cryptographique	273
Langages de programmation	305
Modification de votre modèle de données	332
Résolution des problèmes	337
Renommer le client de chiffrement DynamoDB	342
Référence	344
Format de description du matériau	344
AWS KMS Détails techniques du porte-clés hiérarchique	348
Historique de la documentation	350
.....	cccliii

Qu'est-ce que le SDK AWS de chiffrement des bases de données ?

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Le SDK AWS Database Encryption est un ensemble de bibliothèques logicielles qui vous permettent d'inclure le chiffrement côté client dans la conception de votre base de données. Le SDK AWS Database Encryption fournit des solutions de chiffrement au niveau des enregistrements. Vous spécifiez quels champs sont cryptés et quels champs sont inclus dans les signatures afin de garantir l'authenticité de vos données. Le chiffrement de vos données sensibles en transit et au repos permet de garantir que vos données en texte brut ne sont pas accessibles à des tiers, y compris. AWS Le SDK AWS de chiffrement de base de données est fourni gratuitement sous la licence Apache 2.0.

Ce guide du développeur fournit une présentation conceptuelle du SDK AWS Database Encryption, notamment une [introduction à son architecture](#), des détails sur la [façon dont il protège vos données](#), en quoi il diffère du [chiffrement côté serveur](#) et des conseils sur la [sélection des composants essentiels de votre application](#) afin de vous aider à démarrer.

Le SDK AWS Database Encryption prend en charge Amazon DynamoDB avec un chiffrement au niveau des attributs.

Le SDK AWS de chiffrement de base de données présente les avantages suivants :

Conçu spécialement pour les applications de base de données

Il n'est pas nécessaire d'être un expert en cryptographie pour utiliser le SDK AWS Database Encryption. Les implémentations incluent des méthodes d'assistance conçues pour fonctionner avec vos applications existantes.

Après avoir créé et configuré les composants requis, le client de chiffrement chiffre et signe de manière transparente vos enregistrements lorsque vous les ajoutez à une base de données, puis les vérifie et les déchiffre lorsque vous les récupérez.

Inclut le chiffrement et la signature sécurisées

Le SDK AWS de chiffrement de base de données inclut des implémentations sécurisées qui chiffrent les valeurs des champs de chaque enregistrement à l'aide d'une clé de chiffrement des données unique, puis signent l'enregistrement pour le protéger contre les modifications non autorisées, telles que l'ajout ou la suppression de champs, ou l'échange de valeurs chiffrées.

Utilise les matériaux de chiffrement de n'importe quelle source

Le SDK AWS Database Encryption utilise des [trousseaux de clés](#) pour générer, chiffrer et déchiffrer la clé de chiffrement des données unique qui protège votre dossier. Les porte-clés déterminent les [clés d'encapsulation qui chiffrent cette clé](#) de données.

Vous pouvez utiliser des clés d'encapsulation provenant de n'importe quelle source, y compris des services de cryptographie tels que [AWS Key Management Service](#)(AWS KMS) ou [AWS CloudHSM](#). Le SDK AWS de chiffrement de base de données ne nécessite Compte AWS aucun AWS service.

Support pour la mise en cache du matériel cryptographique

Le trousseau de [clés AWS KMS hiérarchique](#) est une solution de mise en cache des matériaux cryptographiques qui réduit le nombre d' AWS KMS appels en utilisant des clés de branche AWS KMS protégées conservées dans une table Amazon DynamoDB, puis en mettant en cache localement les éléments clés de branche utilisés dans les opérations de chiffrement et de déchiffrement. Il vous permet de protéger votre matériel cryptographique sous une clé KMS de chiffrement symétrique sans avoir à appeler AWS KMS chaque fois que vous chiffrez ou déchiffrez un enregistrement. Le porte-clés AWS KMS hiérarchique est un bon choix pour les applications qui doivent minimiser les appels à AWS KMS.

Chiffrement consultable

Vous pouvez concevoir des bases de données capables de rechercher des enregistrements chiffrés sans déchiffrer l'intégralité de la base de données. En fonction de votre modèle de menace et de vos exigences en matière de requêtes, vous pouvez utiliser le [chiffrement consultable](#) pour effectuer des recherches de correspondance exacte ou des requêtes complexes plus personnalisées sur votre base de données cryptée.

Support pour les schémas de base de données mutualisés

Le SDK AWS Database Encryption vous permet de protéger les données stockées dans des bases de données avec un schéma partagé en isolant chaque client avec des matériaux de

chiffrement distincts. Si plusieurs utilisateurs effectuent des opérations de chiffrement dans votre base de données, utilisez l'un des trousseaux de AWS KMS clés pour fournir à chaque utilisateur une clé distincte à utiliser dans ses opérations cryptographiques. Pour de plus amples informations, veuillez consulter [Utilisation de bases de données mutualisées](#).

Support pour des mises à jour de schéma fluides

Lorsque vous configurez le SDK de chiffrement de AWS base de données, vous fournissez des [actions cryptographiques](#) qui indiquent au client les champs à chiffrer et à signer, les champs à signer (mais pas à chiffrer) et ceux à ignorer. Après avoir utilisé le SDK AWS de chiffrement de base de données pour protéger vos enregistrements, vous pouvez toujours [apporter des modifications à votre modèle de données](#). Vous pouvez mettre à jour vos actions cryptographiques, telles que l'ajout ou la suppression de champs chiffrés, en un seul déploiement.

Développé dans des référentiels open source

Le SDK AWS Database Encryption est développé dans des référentiels open source sur GitHub. Vous pouvez utiliser ces référentiels pour consulter le code, lire et signaler les problèmes, et trouver des informations spécifiques à votre implémentation.

Le SDK AWS de chiffrement de base de données pour DynamoDB

- Le référentiel [aws-database-encryption-sdk-dynamodb](#) GitHub prend en charge les dernières versions du SDK de chiffrement de AWS base de données pour DynamoDB en Java, .NET et Rust.

Le SDK AWS de chiffrement de base de données pour DynamoDB est un produit [de](#) Dafny, un langage compatible avec la vérification dans lequel vous rédigez les spécifications, le code pour les implémenter et les preuves pour les tester. Le résultat est une bibliothèque qui implémente les fonctionnalités du SDK de chiffrement de AWS base de données pour DynamoDB dans une structure garantissant l'exactitude fonctionnelle.

Support et maintenance

Le SDK AWS Database Encryption utilise la même [politique de maintenance](#) que le AWS SDK et les outils, y compris ses phases de versionnement et de cycle de vie. En tant que bonne pratique, nous vous recommandons d'utiliser la dernière version disponible du SDK de chiffrement de base de données pour l'implémentation de votre AWS base de données, et de procéder à une mise à niveau à mesure que de nouvelles versions sont publiées.

Pour plus d'informations, consultez la [politique de maintenance de AWS SDKs and Tools](#) dans le guide de référence AWS SDKs and Tools.

Envoyer un commentaire

Nous apprécions vos commentaires. Si vous avez une question ou un commentaire, ou un problème à signaler, veuillez utiliser les ressources suivantes.

Si vous découvrez une faille de sécurité potentielle dans le SDK de chiffrement des AWS bases de données, veuillez en [informer le service AWS de sécurité](#). Ne créez pas de GitHub problème public.

Pour émettre des commentaires sur cette documentation, utilisez le lien des commentaires sur n'importe quelle page.

AWS Concepts du SDK de chiffrement de base de données

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Cette rubrique explique les concepts et la terminologie utilisés dans le SDK AWS Database Encryption.

Pour savoir comment les composants du SDK de chiffrement AWS de base de données interagissent, consultez [Fonctionnement du SDK AWS de chiffrement de base de données](#).

Pour en savoir plus sur le SDK AWS de chiffrement des bases de données, consultez les rubriques suivantes.

- Découvrez comment le SDK AWS de chiffrement des bases de données utilise le [chiffrement d'enveloppe](#) pour protéger vos données.
- Découvrez les éléments du chiffrement des enveloppes : les [clés de données](#) qui protègent vos enregistrements et les [clés d'encapsulation](#) qui protègent vos clés de données.
- Découvrez les [porte-clés](#) qui déterminent les clés d'emballage que vous utilisez.
- Découvrez le [contexte de chiffrement](#) qui renforce l'intégrité de votre processus de chiffrement.

- Découvrez la [description matérielle](#) que les méthodes de chiffrement ajoutent à votre dossier.
- Découvrez les [actions cryptographiques](#) qui indiquent au SDK AWS Database Encryption les champs à chiffrer et à signer.

Rubriques

- [Chiffrement d'enveloppe](#)
- [Clé de données](#)
- [Clé d'emballage](#)
- [Porte-clés](#)
- [Actions cryptographiques](#)
- [Description du matériau](#)
- [Contexte de chiffrement](#)
- [Gestionnaire de matériaux de chiffrement](#)
- [Chiffrement symétrique et asymétrique](#)
- [Engagement clé](#)
- [Signatures numériques](#)

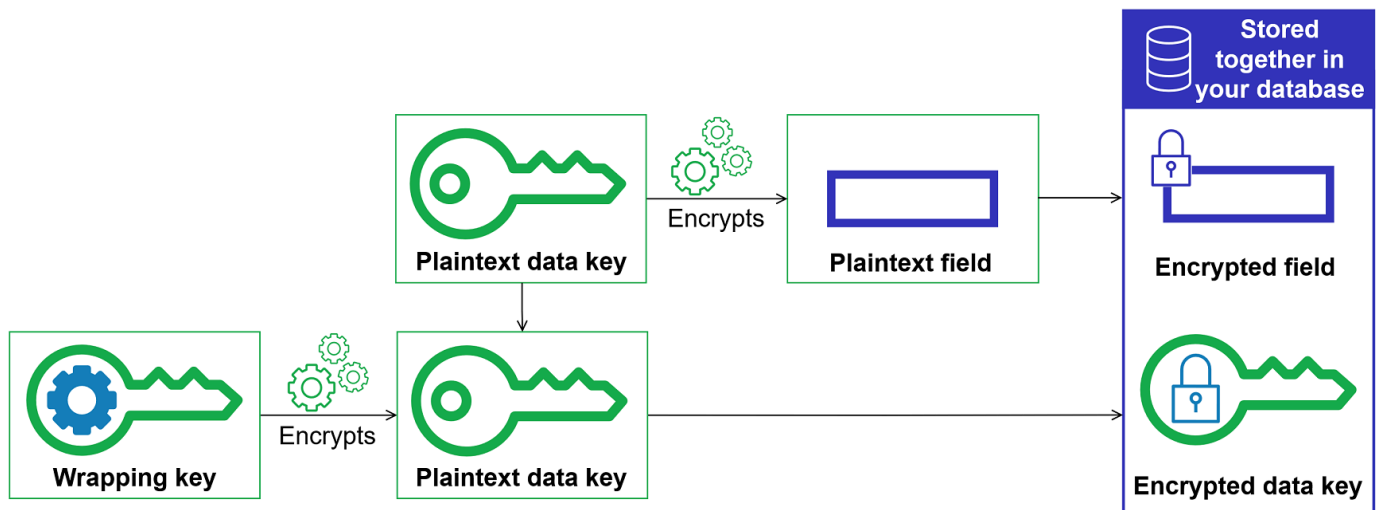
Chiffrement d'enveloppe

La sécurité de vos données chiffrées dépend partiellement de la protection de la clé de données capable de les déchiffrer. Le chiffrement de la clés de données en vue de sa protection est un bonne pratique reconnue. Pour ce faire, vous avez besoin d'une autre clé de chiffrement, connue sous le nom de clé de chiffrement ou clé [d'encapsulation](#). La pratique consistant à utiliser une clé d'encapsulation pour chiffrer des clés de données est connue sous le nom de chiffrement d'enveloppe.

Protection des clés de données

Le SDK AWS Database Encryption chiffre chaque champ à l'aide d'une clé de données unique. Il chiffre ensuite chaque clé de données sous la clé d'encapsulation que vous spécifiez. Il stocke les clés de données cryptées dans la [description du matériau](#).

Pour spécifier votre clé d'emballage, vous utilisez un [porte-clés](#).



Chiffrer les mêmes données sous plusieurs clés d'encapsulation

Vous pouvez chiffrer la clé de données à l'aide de plusieurs clés d'encapsulation. Vous souhaitez peut-être fournir des clés d'encapsulation différentes pour différents utilisateurs, ou des clés d'encapsulation de différents types ou à différents emplacements. Chacune des clés d'encapsulation chiffre la même clé de données. Le SDK AWS de chiffrement de base de données stocke toutes les clés de données cryptées à côté des champs cryptés de la [description du matériel](#).

Pour déchiffrer les données, vous devez fournir au moins une clé d'encapsulation capable de déchiffrer les clés de données chiffrées.

Combinaison des points forts de plusieurs algorithmes

Pour chiffrer vos données, le SDK de chiffrement de AWS base de données utilise par défaut une suite d'algorithmes avec chiffrement symétrique AES-GCM, fonction de dérivation de clés basée sur HMAC (HKDF) et signature ECDSA. Pour chiffrer la clé de données, vous pouvez spécifier un algorithme de chiffrement symétrique ou asymétrique adapté à votre clé d'encapsulation.

En règle générale, les algorithmes de chiffrement à clé symétrique sont plus rapides et produisent des textes chiffrés plus petits que le chiffrement asymétrique et le chiffrement de clé publique. Mais les algorithmes à clé publique assurent une séparation intrinsèque des rôles. Pour combiner les points forts de chacun, vous pouvez chiffrer la clé de données avec le chiffrement par clé publique.

Nous vous recommandons d'utiliser l'un des AWS KMS porte-clés dans la mesure du possible. Lorsque vous utilisez le [AWS KMS trousseau de clés](#), vous pouvez choisir de combiner les

forces de plusieurs algorithmes en spécifiant un RSA asymétrique AWS KMS key comme clé d'encapsulation. Vous pouvez également utiliser une clé KMS de chiffrement symétrique.

Clé de données

Une clé de données est une clé de chiffrement que le SDK de chiffrement de AWS base de données utilise pour chiffrer les champs d'un enregistrement marqués ENCRYPT_AND_SIGN dans les actions [cryptographiques](#). Chaque clé de données est un tableau d'octets qui respecte les exigences concernant les clés cryptographiques. Le SDK AWS de chiffrement de base de données utilise une clé de données unique pour chiffrer chaque attribut.

Il n'est pas nécessaire de spécifier, de générer, d'implémenter, d'étendre, de protéger ou d'utiliser des clés de données. Le SDK AWS de chiffrement de base de données fait cela pour vous lorsque vous appelez les opérations de chiffrement et de déchiffrement.

[Pour protéger vos clés de données, le SDK AWS de chiffrement de base de données les chiffre à l'aide d'une ou de plusieurs clés de chiffrement appelées clés d'encapsulation.](#) Une fois que le SDK AWS de chiffrement de base de données utilise vos clés de données en texte brut pour chiffrer vos données, il les supprime de la mémoire dès que possible. Stocke ensuite la clé de données cryptée dans la [description du matériau](#). Pour en savoir plus, consultez [Fonctionnement du SDK AWS de chiffrement de base de données](#).

Tip

Dans le SDK AWS Database Encryption, nous distinguons les clés de données des clés de chiffrement de données. Il est recommandé que toutes les [suites d'algorithmes](#) prises en charge utilisent une [fonction de dérivation de clés](#). La fonction de dérivation de clés prend une clé de données en entrée et renvoie les clés de chiffrement des données qui sont réellement utilisées pour chiffrer vos enregistrements. C'est pour cette raison que nous indiquons souvent que les données sont chiffrées « sous » une clé de données plutôt que « par » la clé de données.

Chaque clé de données chiffrée inclut des métadonnées, notamment l'identifiant de la clé d'encapsulation qui l'a chiffrée. Ces métadonnées permettent au SDK de chiffrement de AWS base de données d'identifier les clés d'encapsulation valides lors du déchiffrement.

Clé d'emballage

Une clé d'encapsulation est une clé de chiffrement utilisée par le SDK AWS de chiffrement de base de données pour chiffrer la [clé de données](#) qui chiffre vos enregistrements. Chaque clé de données peut être chiffrée sous une ou plusieurs clés d'encapsulation. Vous déterminez quelles clés d'encapsulation sont utilisées pour protéger vos données lorsque vous configurez un [trousseau de clés](#).



Le SDK AWS Database Encryption prend en charge plusieurs clés d'encapsulation couramment utilisées, telles que [AWS Key Management Service](#) (AWS KMS) les clés KMS de chiffrement symétriques (y compris les clés [multirégionales](#)) et [AWS KMS les clés RSA KMS asymétriques](#), les [clés AES-GCM](#) (Advanced Encryption Standard/Galois Counter Mode) brutes et les clés RSA brutes. Nous vous recommandons d'utiliser des clés KMS dans la mesure du possible. Pour choisir la clé d'encapsulation à utiliser, consultez la section [Sélection des clés d'encapsulation](#).

Lorsque vous utilisez le chiffrement des enveloppes, vous devez protéger vos clés d'emballage contre tout accès non autorisé. Pour ce faire, vous pouvez utiliser l'une des méthodes suivantes :

- Utilisez un service conçu à cet effet, tel que [AWS Key Management Service \(AWS KMS\)](#).
- Utilisez un [module de sécurité matériel \(HSM\)](#) tel que celui proposé par [AWS CloudHSM](#).
- Utilisez d'autres outils et services de gestion clés.

Si vous n'avez pas de système de gestion des clés, nous vous le recommandons AWS KMS. Le SDK AWS de chiffrement de base de données s'intègre AWS KMS pour vous aider à protéger et à utiliser vos clés d'encapsulation.

Porte-clés

Pour spécifier les clés d'encapsulation que vous utilisez pour le chiffrement et le déchiffrement, vous utilisez un trousseau de clés. Vous pouvez utiliser les trousseaux de clés fournis par le SDK AWS Database Encryption ou concevoir vos propres implémentations.

Un porte-clés génère, chiffre et déchiffre des clés de données. Il génère également les clés MAC utilisées pour calculer les codes d'authentification des messages basés sur le hachage (HMACs) contenus dans la signature. Lorsque vous définissez un trousseau de clés, vous pouvez spécifier les [clés d'encapsulation](#) qui chiffrent vos clés de données. La plupart des porte-clés contiennent au moins une clé d'emballage ou un service fournissant et protégeant les clés d'emballage. Lors du chiffrement, le SDK AWS de chiffrement de base de données utilise toutes les clés d'encapsulation spécifiées dans le jeu de clés pour chiffrer la clé de données. Pour obtenir de l'aide sur le choix et l'utilisation des trousseaux de clés définis par le SDK AWS de chiffrement de base de données, consultez la section [Utilisation](#) des trousseaux de clés.

Actions cryptographiques

Les actions cryptographiques indiquent au crypteur les actions à effectuer sur chaque champ d'un enregistrement.

Les valeurs de l'action cryptographique peuvent être l'une des suivantes :

- Chiffrer et signer — Chiffrez le champ. Incluez le champ crypté dans la signature.
- Signer uniquement — Incluez le champ dans la signature.
- Signer et inclure dans le contexte de chiffrement — Incluez le champ dans le [contexte de signature et de chiffrement](#).

Par défaut, les clés de partition et de tri sont les seuls attributs inclus dans le contexte de chiffrement. Vous pouvez envisager de définir des champs supplémentaires `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` afin que le fournisseur d'ID de clé de branche pour votre jeu de [clés AWS KMS hiérarchique](#) puisse identifier la clé de branche requise pour le déchiffrement à partir du contexte de chiffrement. Pour plus d'informations, consultez le [fournisseur d'ID de clé de branche](#).

Note

Pour utiliser l'action `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` cryptographique, vous devez utiliser la version 3.3 ou ultérieure du SDK AWS Database Encryption. Déployez la nouvelle version sur tous les lecteurs avant de [mettre à jour votre modèle de données](#) pour l'inclure `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

- Ne rien faire : ne chiffrez pas et n'incluez pas le champ dans la signature.

Pour tous les champs susceptibles de stocker des données sensibles, utilisez Chiffrer et signer. Pour les valeurs de clé primaire (par exemple, une clé de partition et une clé de tri dans une table DynamoDB), utilisez Signer uniquement ou Signer et inclure dans le contexte de chiffrement. Si vous spécifiez des attributs Sign et incluez dans le contexte de chiffrement, les attributs de partition et de tri doivent également être Signer et inclure dans le contexte de chiffrement. Il n'est pas nécessaire de spécifier des actions cryptographiques pour la [description du matériau](#). Le SDK AWS Database Encryption signe automatiquement le champ dans lequel la description du matériau est stockée.

Choisissez vos actions cryptographiques avec soin. En cas de doute, utilisez Chiffrer et signer. Une fois que vous avez utilisé le SDK de chiffrement de AWS base de données pour protéger vos enregistrements, vous ne pouvez pas remplacer un champ existant ENCRYPT_AND_SIGN par un SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT champ existantDO_NOTHING, ni modifier l'action cryptographique affectée à un champ existantDO_NOTHING. SIGN_ONLY Toutefois, vous pouvez toujours [apporter d'autres modifications à votre modèle de données](#). Par exemple, vous pouvez ajouter ou supprimer des champs chiffrés dans le cadre d'un seul déploiement.

Description du matériau

La description du matériau sert d'en-tête à un enregistrement crypté. Lorsque vous chiffrez et signez des champs avec le SDK de chiffrement de AWS base de données, le crypteur enregistre la description du matériel au fur et à mesure qu'il assemble le matériel cryptographique et stocke la description du matériel dans un nouveau champ (aws_dbe_head) qu'il ajoute à votre enregistrement.

La description matérielle est une [structure de données formatée](#) portable qui contient des copies cryptées des clés de données et d'autres informations, telles que les algorithmes de chiffrement, [le contexte de chiffrement](#) et les instructions de chiffrement et de signature. Le crypteur enregistre la description du matériel lorsqu'il assemble le matériel cryptographique pour le chiffrement et la signature. Plus tard, lorsqu'il doit assembler du matériel cryptographique pour vérifier et déchiffrer un champ, il utilise la description du matériel comme guide.

Le stockage des clés de données chiffrées à côté du champ crypté rationalise l'opération de déchiffrement et vous évite d'avoir à stocker et à gérer les clés de données cryptées indépendamment des données qu'elles chiffrent.

Pour des informations techniques sur la description du matériau, voir [Format de description du matériau](#).

Contexte de chiffrement

Pour améliorer la sécurité de vos opérations cryptographiques, le SDK de chiffrement AWS de base de données inclut un contexte de chiffrement dans toutes les demandes de chiffrement et de signature d'un enregistrement.

Un contexte de chiffrement est un ensemble de paires nom-valeur qui contient des données non secrètes arbitraires authentifiées supplémentaires. Le SDK AWS Database Encryption inclut le nom logique de votre base de données et les valeurs des clés primaires (par exemple, une clé de partition et une clé de tri dans une table DynamoDB) dans le contexte du chiffrement. Lorsque vous chiffrez et signez un champ, le contexte de chiffrement est lié cryptographiquement à l'enregistrement chiffré, de sorte que le même contexte de chiffrement est requis pour déchiffrer le champ.

Si vous utilisez un AWS KMS trousseau de clés, le SDK AWS Database Encryption utilise également le contexte de chiffrement pour fournir des données authentifiées supplémentaires (AAD) dans les appels auxquels le trousseau de clés est envoyé. AWS KMS

Chaque fois que vous utilisez la [suite d'algorithmes par défaut](#), le [gestionnaire de matériel cryptographique](#) (CMM) ajoute une paire nom-valeur au contexte de chiffrement composée d'un nom réservé et d'une valeur représentant la clé de vérification publique. `aws-crypto-public-key` La clé de vérification publique est enregistrée dans la [description du matériau](#).

Gestionnaire de matériaux de chiffrement

Le gestionnaire de matériel cryptographique (CMM) assemble le matériel cryptographique utilisé pour chiffrer, déchiffrer et signer vos données. Chaque fois que vous utilisez la [suite d'algorithmes par défaut](#), le matériel cryptographique inclut des clés de données chiffrées et en texte brut, des clés de signature symétriques et une clé de signature asymétrique. Vous n'interagissez jamais directement avec le CMM. Les méthodes de chiffrement et de déchiffrement s'en occupent pour vous.

Comme le CMM fait office de liaison entre le SDK de chiffrement de AWS base de données et un trousseau de clés, il constitue un point idéal pour la personnalisation et l'extension, notamment pour le soutien à l'application des politiques. Vous pouvez spécifier un CMM de manière explicite, mais ce n'est pas obligatoire. Lorsque vous spécifiez un jeu de clés, le SDK de chiffrement AWS de base de données crée un CMM par défaut pour vous. Le CMM par défaut obtient le matériel de chiffrement ou de déchiffrement à partir du trousseau de clés que vous spécifiez. Pour cela, il peut être nécessaire d'appeler un service de chiffrement, comme [AWS Key Management Service](#) (AWS KMS).

Chiffrement symétrique et asymétrique

Le chiffrement symétrique utilise la même clé pour chiffrer et déchiffrer les données.

Le chiffrement asymétrique utilise une paire de clés de données liées mathématiquement. L'une des clés de la paire chiffre les données ; seule l'autre clé de la paire peut les déchiffrer.

Le SDK AWS de chiffrement de base de données utilise le [chiffrement d'enveloppe](#). Il chiffre vos données à l'aide d'une clé de données symétrique. Il chiffre la clé de données symétrique avec une ou plusieurs clés d'encapsulation symétriques ou asymétriques. Il ajoute une [description matérielle](#) à l'enregistrement qui inclut au moins une copie cryptée de la clé de données.

Chiffrement de vos données (chiffrement symétrique)

Pour chiffrer vos données, le SDK de chiffrement AWS de base de données utilise une [clé de données](#) symétrique et une [suite d'algorithmes incluant un algorithme](#) de chiffrement symétrique. Pour déchiffrer les données, le SDK AWS de chiffrement de base de données utilise la même clé de données et la même suite d'algorithmes.

Chiffrement de votre clé de données (chiffrement symétrique ou asymétrique)

Le [trousseau de clés](#) que vous fournissez à une opération de chiffrement et de déchiffrement détermine la manière dont la clé de données symétrique est chiffrée et déchiffrée. Vous pouvez choisir un trousseau de clés utilisant un chiffrement symétrique, tel qu'un trousseau de AWS KMS clés doté d'une clé KMS de chiffrement symétrique, ou un jeu de clés utilisant un chiffrement asymétrique, tel qu'un trousseau de clés doté d'une AWS KMS clé RSA KMS asymétrique.

Engagement clé

Le SDK AWS de chiffrement des bases de données prend en charge l'engagement par clé (parfois appelé robustesse), une propriété de sécurité qui garantit que chaque texte chiffré ne peut être déchiffré qu'en un seul texte clair. Pour ce faire, l'engagement clé garantit que seule la clé de données qui a chiffré votre enregistrement sera utilisée pour le déchiffrer. Le SDK AWS Database Encryption inclut un engagement clé pour toutes les opérations de chiffrement et de déchiffrement.

La plupart des chiffrements symétriques modernes (y compris AES) chiffrent le texte en clair avec une clé secrète unique, comme la clé de [données unique que le SDK de chiffrement de AWS base de données](#) utilise pour chiffrer chaque champ de texte en clair marqué dans un enregistrement. ENCRYPT_AND_SIGN Le déchiffrement de cet enregistrement avec la même clé de données renvoie

un texte en clair identique à l'original. Le déchiffrement avec une autre clé échouera généralement. Bien que cela soit difficile, il est techniquement possible de déchiffrer un texte chiffré sous deux clés différentes. Dans de rares cas, il est possible de trouver une clé capable de déchiffrer partiellement le texte chiffré en un texte clair différent, mais toujours intelligible.

Le SDK AWS de chiffrement de base de données chiffre toujours chaque attribut sous une clé de données unique. Il peut chiffrer cette clé de données sous plusieurs clés d'encapsulation, mais les clés d'encapsulation chiffrent toujours la même clé de données. Néanmoins, un enregistrement chiffré sophistiqué créé manuellement peut en fait contenir différentes clés de données, chacune chiffrée par une clé d'encapsulation différente. Par exemple, si un utilisateur déchiffre l'enregistrement chiffré, il renvoie 0x0 (faux) tandis qu'un autre utilisateur déchiffre le même enregistrement crypté obtient 0x1 (vrai).

Pour éviter ce scénario, le SDK de chiffrement AWS de base de données inclut un engagement clé lors du chiffrement et du déchiffrement. La méthode de cryptage lie cryptographiquement la clé de données unique qui a produit le texte chiffré à l'engagement de clé, un code d'authentification de message basé sur le hachage (HMAC) calculé sur la description du matériel à l'aide d'une dérivation de la clé de données. Ensuite, il enregistre l'engagement clé dans la [description du matériel](#). Lorsqu'il déchiffre un enregistrement avec une clé d'engagement, le SDK de chiffrement de AWS base de données vérifie que la clé de données est la seule clé pour cet enregistrement chiffré. Si la vérification de la clé de données échoue, l'opération de déchiffrement échoue.

Signatures numériques


Le SDK AWS Database Encryption chiffre vos données à l'aide d'un algorithme de chiffrement authentifié, AES-GCM, et le processus de déchiffrement vérifie l'intégrité et l'authenticité d'un message chiffré sans utiliser de signature numérique. Mais comme AES-GCM utilise des clés symétriques, toute personne capable de déchiffrer la clé de données utilisée pour déchiffrer le texte chiffré pourrait également créer manuellement un nouveau texte chiffré, ce qui pourrait poser un problème de sécurité. Par exemple, si vous utilisez un AWS KMS key comme clé d'encapsulation, un utilisateur `kms:Decrypt` autorisé peut créer des textes chiffrés sans appeler `kms:Encrypt`.

Pour éviter ce problème, la [suite d'algorithmes par défaut](#) ajoute une signature ECDSA (Elliptic Curve Digital Signature Algorithm) aux enregistrements chiffrés. La suite d'algorithmes par défaut chiffre les champs de votre enregistrement marqués à l'ENCRYPT_AND_SIGN avec un algorithme de chiffrement authentifié, AES-GCM. Ensuite, il calcule à la fois les codes d'authentification des messages basés sur le hachage (HMACs) et les signatures ECDSA asymétriques sur les champs de votre enregistrement marqués, et. ENCRYPT_AND_SIGN

`SIGN_ONLY SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Le processus de déchiffrement utilise les signatures pour vérifier qu'un utilisateur autorisé a chiffré l'enregistrement.

Lorsque la suite d'algorithmes par défaut est utilisée, le SDK AWS de chiffrement de base de données génère une clé privée temporaire et une paire de clés publiques pour chaque enregistrement chiffré. Le SDK AWS Database Encryption stocke la clé publique dans la [description matérielle](#) et supprime la clé privée. Cela garantit que personne ne peut créer une autre signature vérifiant avec la clé publique. L'algorithme lie la clé publique à la clé de données cryptée en tant que données authentifiées supplémentaires dans la description du matériel, empêchant ainsi les utilisateurs qui ne peuvent que déchiffrer des champs de modifier la clé publique ou d'affecter la vérification de signature.

Le SDK AWS de chiffrement de base de données inclut toujours la vérification HMAC. Les signatures numériques ECDSA sont activées par défaut, mais elles ne sont pas obligatoires. Si les utilisateurs qui chiffrent les données et ceux qui les déchiffrant jouissent de la même confiance, vous pouvez envisager d'utiliser une suite d'algorithmes qui n'inclut pas de signatures numériques pour améliorer vos performances. Pour plus d'informations sur la sélection d'autres suites d'algorithmes, voir [Choisir une suite d'algorithmes](#).

 Note

Si un trousseau de clés ne fait pas de distinction entre les crypteurs et les déchiffreurs, les signatures numériques ne fournissent aucune valeur cryptographique.

AWS KMS les [trousseaux de clés](#), y compris le AWS KMS porte-clés RSA asymétrique, peuvent faire la distinction entre les chiffreurs et les déchiffreurs en fonction des politiques clés et des politiques IAM. AWS KMS

En raison de leur nature cryptographique, les trousseaux de clés suivants ne peuvent pas faire la distinction entre les crypteurs et les déchiffreurs :

- AWS KMS Porte-clés hiérarchique
- AWS KMS Porte-clés ECDH
- Porte-clés AES brut
- Porte-clés RSA brut
- Porte-clés ECDH brut

Fonctionnement du SDK AWS de chiffrement de base de données

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Le SDK AWS Database Encryption fournit des bibliothèques de chiffrement côté client conçues spécifiquement pour protéger les données que vous stockez dans les bases de données. Les bibliothèques incluent les implémentations sécurisées que vous pouvez étendre ou utiliser inchangées. Pour plus d'informations sur la définition et l'utilisation de composants personnalisés, consultez le GitHub référentiel pour l'implémentation de votre base de données.

Les flux de travail présentés dans cette section expliquent comment le SDK AWS Database Encryption chiffre, signe, déchiffre et vérifie les données de votre base de données. Ces flux de travail décrivent le processus de base utilisant des éléments abstraits et les fonctionnalités par défaut. Pour en savoir plus sur le fonctionnement du SDK AWS de chiffrement de base de données avec l'implémentation de votre base de données, consultez la rubrique [Qu'est-ce que le chiffrement pour votre base de données ?](#)

Le SDK AWS Database Encryption utilise le [chiffrement d'enveloppe](#) pour protéger vos données. Chaque enregistrement est crypté sous une [clé de données](#) unique. La clé de données est utilisée pour dériver une clé de chiffrement des données unique pour chaque champ marqué ENCRYPT_AND_SIGN dans vos actions cryptographiques. Ensuite, une copie de la clé de données est chiffrée par les clés d'encapsulation que vous spécifiez. Pour déchiffrer l'enregistrement chiffré, le SDK de chiffrement AWS de base de données utilise les clés d'encapsulation que vous spécifiez pour déchiffrer au moins une clé de données chiffrée. Il peut ensuite déchiffrer le texte chiffré et renvoyer une entrée en texte brut.

Pour plus d'informations sur les termes utilisés dans le SDK de chiffrement AWS de base de données, consultez [AWS Concepts du SDK de chiffrement de base de données](#).


Chiffrer et signer

À la base, le SDK AWS de chiffrement de base de données est un crypteur d'enregistrements qui chiffre, signe, vérifie et déchiffre les enregistrements de votre base de données. Il prend en compte les informations relatives à vos dossiers et les instructions concernant les champs à chiffrer et à signer. Il obtient le matériel de chiffrement et les instructions sur la façon de les utiliser à partir

d'un [gestionnaire de matériel cryptographique](#) configuré à partir de la clé d'encapsulation que vous spécifiez.

La procédure pas à pas suivante décrit comment le SDK AWS de chiffrement de base de données chiffre et signe vos entrées de données.

1. Le gestionnaire de matériel cryptographique fournit au SDK AWS Database Encryption des clés de chiffrement de données uniques : une [clé de données](#) en texte brut, une copie de la clé de données chiffrée par la clé [d'encapsulation spécifiée et une clé MAC](#).


 Note

Vous pouvez chiffrer la clé de données sous plusieurs clés d'encapsulation. Chacune des clés d'encapsulation chiffre une copie séparée de la clé de données. Le SDK AWS de chiffrement de base de données stocke toutes les clés de données cryptées dans la [description du matériel](#). Le SDK AWS Database Encryption ajoute un nouveau champ (`aws_dbe_head`) à l'enregistrement qui contient la description du matériel.

Une clé MAC est dérivée pour chaque copie cryptée de la clé de données. Les clés MAC ne sont pas enregistrées dans la description du matériau. Au lieu de cela, la méthode de déchiffrement utilise les clés d'encapsulation pour dériver à nouveau les clés MAC.

2. La méthode de chiffrement chiffre chaque champ marqué comme indiqué `ENCRYPT_AND_SIGN` dans les [actions cryptographiques que vous avez spécifiées](#).
3. Le procédé de chiffrement `commitKey` dérive une de la clé de données et l'utilise pour générer une [valeur d'engagement de clé](#), puis supprime la clé de données.
4. La méthode de cryptage ajoute une [description matérielle](#) à l'enregistrement. La description du matériel contient les clés de données cryptées et les autres informations relatives à l'enregistrement crypté. Pour une liste complète des informations incluses dans la description du matériau, voir [Format de description du matériau](#).
5. La méthode de chiffrement utilise les clés MAC renvoyées à l'étape 1 pour calculer les valeurs du code d'authentification des messages basé sur le hachage (HMAC) lors de la canonisation de la description du matériel, du [contexte de chiffrement](#) et de chaque champ marqué `ENCRYPT_AND_SIGN` ou `SIGN_ONLY`, ou `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` dans le cadre des actions cryptographiques. Les valeurs HMAC sont stockées dans un nouveau champ (`aws_dbe_foot`) que la méthode de chiffrement ajoute à l'enregistrement.
6. La méthode de chiffrement calcule une [signature ECDSA lors](#) de la canonisation de la description du matériau, du contexte de chiffrement et de chaque champ marqué

ENCRYPT_AND_SIGNSIGN_ONLY, ou SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT stocke les signatures ECDSA dans le champ. aws_dbe_foot

 Note

Les signatures ECDSA sont activées par défaut, mais ne sont pas obligatoires.

7. La méthode de cryptage stocke l'enregistrement crypté et signé dans votre base de données

Déchiffrer et vérifier

1. Le gestionnaire de matériel cryptographique (CMM) fournit la méthode de déchiffrement avec les matériaux de déchiffrement stockés dans la description du matériel, y compris la clé de [données en texte clair et la clé MAC associée](#).
 - Le CMM déchiffre la clé de données cryptée à l'aide des clés d'[encapsulation du jeu de clés](#) spécifié et renvoie la clé de données en texte brut.
2. La méthode de déchiffrement compare et vérifie la valeur d'engagement clé dans la description du matériau.
3. La méthode de déchiffrement vérifie les signatures dans le champ de signature.

Il identifie les champs marqués ENCRYPT_AND_SIGN ou SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT ceux figurant dans la liste des [champs non authentifiés autorisés](#) que vous avez définis. SIGN_ONLY La méthode de déchiffrement utilise la clé MAC renvoyée à l'étape 1 pour recalculer et comparer les valeurs HMAC des champs marqués, ou. ENCRYPT_AND_SIGN SIGN_ONLY SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT [Il vérifie ensuite les signatures ECDSA à l'aide de la clé publique stockée dans le contexte de chiffrement](#).

4. La méthode de déchiffrement utilise la clé de données en texte brut pour déchiffrer chaque valeur marquée. ENCRYPT_AND_SIGN Le SDK AWS de chiffrement de base de données supprime ensuite la clé de données en texte brut.
5. La méthode de déchiffrement renvoie l'enregistrement en texte brut.

Suites d'algorithmes prises en charge dans le SDK AWS de chiffrement de base de données

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Une suite d'algorithmes est un ensemble d'algorithmes de chiffrement et de valeurs connexes. Les systèmes cryptographiques utilisent l'implémentation de l'algorithme pour générer le texte chiffré.

Le SDK AWS Database Encryption utilise une suite d'algorithmes pour chiffrer et signer les champs de votre base de données. Toutes les suites d'algorithmes prises en charge utilisent l'algorithme Advanced Encryption Standard (AES) avec Galois/Counter mode (GCM), connu sous le nom d'AES-GCM, pour chiffrer les données brutes. Le SDK AWS de chiffrement de base de données prend en charge les clés de chiffrement 256 bits. La longueur de la balise d'authentification est toujours de 16 octets.

AWS Suites d'algorithmes du SDK de chiffrement de base de données

Algorithm	Algorithme de chiffrement	Longueur de la clé de données (en bits)	Algorithme de dérivation de clé	Algorithme de signature symétrique	Algorithme de signature asymétrique	Engagement clé
Par défaut	AES-GCM	256	HKDF avec SHA-512	HMAC-SHA-384	ECDSA avec P-384 et SHA-384	HKDF avec SHA-512
AES-GCM sans signatures numériques ECDSA	AES-GCM	256	HKDF avec SHA-512	HMAC-SHA-384	Aucune	HKDF avec SHA-512

Algorithme de chiffrement

Nom et mode de l'algorithme de chiffrement utilisé. Les suites d'algorithmes du SDK AWS de chiffrement de base de données utilisent l'algorithme Advanced Encryption Standard (AES) avec Galois/Counter mode (GCM).

Longueur de la clé de données

Longueur de la [clé de données](#) en bits. Le SDK AWS de chiffrement de base de données prend en charge les clés de données 256 bits. La clé de données est utilisée comme entrée dans une fonction de dérivation de extract-and-expand clé basée sur HMAC (HKDF). La sortie de la fonction HKDF est utilisée comme la clé de chiffrement des données dans l'algorithme de chiffrement.

Algorithme de dérivation de clé

La fonction de dérivation de extract-and-expand clé basée sur HMAC (HKDF) utilisée pour dériver la clé de chiffrement des données. Le SDK AWS de chiffrement de base de données utilise le HKDF défini dans la [RFC 5869](#).

- La fonction de hachage utilisée est SHA-512
- Pour l'étape d'extraction :
 - Aucune valeur salt n'est utilisée. Selon la RFC, le sel est défini sur une chaîne de zéros.
 - Le matériel de saisie est la clé de données du [trousseau de clés](#).
- Pour l'étape de développement :
 - La clé pseudo aléatoire en entrée est la sortie de l'étape d'extraction.
 - L'étiquette clé correspond aux octets codés en UTF-8 de la DERIVEKEY chaîne dans l'ordre des octets en gros endian.
 - Les informations d'entrée sont une concaténation de l'identifiant de l'algorithme et de l'étiquette clé (dans cet ordre).
 - La longueur du matériel de saisie de sortie est la longueur de la clé de données. Cette sortie est utilisée comme la clé de chiffrement des données dans l'algorithme de chiffrement.

Algorithme de signature symétrique

Algorithme HMAC (Hash Based Message Authentication Code) utilisé pour générer une signature symétrique. Toutes les suites d'algorithmes prises en charge incluent la vérification HMAC.

Le SDK AWS Database Encryption sérialise la description du matériel et tous les champs marqués ENCRYPT_AND_SIGNSIGN_ONLY, ou.

`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Ensuite, il utilise HMAC avec un algorithme de fonction de hachage cryptographique (SHA-384) pour signer la canonisation.

La signature HMAC symétrique est stockée dans un nouveau champ (`aws_dbe_foot`) que le SDK de chiffrement de AWS base de données ajoute à l'enregistrement.

Algorithme de signature asymétrique

Algorithme de signature utilisé pour générer une signature numérique asymétrique.

Le SDK AWS Database Encryption sérialise la description du matériel et tous les champs marqués `ENCRYPT_AND_SIGNSIGN_ONLY`, ou.

`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` Il utilise ensuite l'algorithme de signature numérique à courbe elliptique (ECDSA) avec les spécificités suivantes pour signer la canonicalisation :

- La courbe elliptique utilisée est la P-384, telle que définie dans la [norme de signature numérique \(DSS\) \(FIPS PUB 186-4\)](#).
- La fonction de hachage utilisée est SHA-384.

La signature ECDSA asymétrique est stockée avec la signature HMAC symétrique sur le terrain. `aws_dbe_foot`

Les signatures numériques ECDSA sont incluses par défaut, mais elles ne sont pas obligatoires.

Engagement clé

La fonction de dérivation de extract-and-expand clé basée sur HMAC (HKDF) utilisée pour dériver la clé de validation.

- La fonction de hachage utilisée est SHA-512
- Pour l'étape d'extraction :
 - Aucune valeur salt n'est utilisée. Selon la RFC, le sel est défini sur une chaîne de zéros.
 - Le matériel de saisie est la clé de données du [trousseau de clés](#).
- Pour l'étape de développement :
 - La clé pseudo aléatoire en entrée est la sortie de l'étape d'extraction.
 - Les informations d'entrée sont les octets codés en UTF-8 de la `COMMITKEY` chaîne dans l'ordre des octets Big Endian.
 - La longueur du matériel de saisie de sortie est de 256 bits. Cette sortie est utilisée comme clé de validation.

[La clé de validation calcule l'engagement de l'enregistrement, un hachage distinct du code d'authentification des messages basé sur le hachage \(HMAC\) de 256 bits, par rapport à la description du matériau.](#) Pour une explication technique de l'ajout d'un engagement clé à une suite d'algorithmes, voir [Key Commitment AEADs](#) in Cryptology ePrint Archive.

Suite d'algorithmes par défaut

Par défaut, le SDK de chiffrement AWS de base de données utilise une suite d'algorithmes avec AES-GCM, une fonction de dérivation de extract-and-expand clé basée sur HMAC (HKDF), une vérification HMAC, des signatures numériques ECDSA, un engagement de clé et une clé de chiffrement 256 bits.

La suite d'algorithmes par défaut inclut la vérification HMAC (signatures symétriques) et les signatures [numériques ECDSA \(signatures asymétriques\)](#). Ces signatures sont stockées dans un nouveau champ (`aws_dbe_foot`) que le SDK AWS de chiffrement de base de données ajoute à l'enregistrement. Les signatures numériques ECDSA sont particulièrement utiles lorsque la politique d'autorisation permet à un ensemble d'utilisateurs de chiffrer des données et à un autre ensemble d'utilisateurs de les déchiffrer.

La suite d'algorithmes par défaut déduit également un [engagement clé](#) : un hachage HMAC qui lie la clé de données à l'enregistrement. La valeur d'engagement clé est un HMAC calculé à partir de la description du matériau et de la clé de validation. La valeur d'engagement clé est ensuite enregistrée dans la description du matériau. Un engagement clé garantit que chaque texte chiffré est déchiffré en un seul texte clair. Pour ce faire, ils valident la clé de données utilisée comme entrée dans l'algorithme de chiffrement. Lors du chiffrement, la suite d'algorithmes déduit un engagement clé HMAC. Avant le déchiffrement, ils valident que la clé de données produit la même clé d'engagement HMAC. Si ce n'est pas le cas, l'appel de déchiffrement échoue.

AES-GCM sans signatures numériques ECDSA

Bien que la suite d'algorithmes par défaut soit probablement adaptée à la plupart des applications, vous pouvez choisir une autre suite d'algorithmes. Par exemple, certains modèles de confiance seraient satisfaits par une suite d'algorithmes sans signature numérique ECDSA. Utilisez cette suite uniquement lorsque les utilisateurs qui chiffrent les données et ceux qui les déchiffrent jouissent de la même confiance.

Toutes les suites d'algorithmes du SDK de chiffrement de AWS base de données incluent la vérification HMAC (signatures symétriques). La seule différence est que la suite d'algorithmes AES-

GCM sans signature numérique ECDSA ne dispose pas de la signature asymétrique qui fournit une couche supplémentaire d'authenticité et de non-répudiation.

Par exemple, si votre jeu de clés contient plusieurs clés d'encapsulation, `wrappingKeyA`, `wrappingKeyB` et `wrappingKeyC`, et que vous déchiffrez un enregistrement à l'aide de `wrappingKeyA`, la signature symétrique HMAC vérifie que l'enregistrement a été crypté par un utilisateur ayant accès à `wrappingKeyA`. Si vous avez utilisé la suite d'algorithmes par défaut, vous devez fournir la même vérification et utiliser en outre la signature numérique ECDSA pour garantir que l'enregistrement a été crypté par un utilisateur disposant d'autorisations de chiffrement pour `wrappingKeyA`.

Pour sélectionner la suite d'algorithmes AES-GCM sans signature numérique, incluez l'extrait suivant dans votre configuration de chiffrement.

Java

L'extrait suivant spécifie la suite d'algorithmes AES-GCM sans signatures numériques ECDSA. Pour de plus amples informations, veuillez consulter [the section called “Configuration du chiffrement”](#).

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

C# / .NET

L'extrait suivant spécifie la suite d'algorithmes AES-GCM sans signatures numériques ECDSA. Pour de plus amples informations, veuillez consulter [the section called “Configuration du chiffrement”](#).

```
AlgorithmSuiteId =  
    DBEAlgorithmSuiteId.ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY_SYMSIG_HMAC_SHA384
```

Rust

L'extrait suivant spécifie la suite d'algorithmes AES-GCM sans signatures numériques ECDSA. Pour de plus amples informations, veuillez consulter [the section called “Configuration du chiffrement”](#).

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

)

Utilisation du SDK AWS de chiffrement de base de données avec AWS KMS

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Pour utiliser le SDK AWS Database Encryption, vous devez configurer un jeu de [clés](#) et spécifier une ou plusieurs clés d'encapsulation. Si vous n'avez pas d'infrastructure à clé, nous vous recommandons d'utiliser [AWS Key Management Service \(AWS KMS\)](#).

Le SDK AWS Database Encryption prend en charge deux types de AWS KMS trousseaux de clés. Le trousseau de [AWS KMS clés](#) traditionnel est utilisé [AWS KMS keys](#) pour générer, chiffrer et déchiffrer des clés de données. Vous pouvez utiliser un chiffrement symétrique (SYMMETRIC_DEFAULT) ou des clés RSA KMS asymétriques. Étant donné que le SDK AWS de chiffrement des bases de données chiffre et signe chaque enregistrement à l'aide d'une clé de données unique, le AWS KMS trousseau de clés doit faire appel à chaque opération AWS KMS de chiffrement et de déchiffrement. Pour les applications qui doivent minimiser le nombre d'appels AWS KMS, le SDK de chiffrement de AWS base de données prend également en charge le jeu de [clés AWS KMS hiérarchique](#). Le trousseau de clés hiérarchique est une solution de mise en cache des matériaux cryptographiques qui réduit le nombre d' AWS KMS appels en utilisant des clés de branche AWS KMS protégées conservées dans une table Amazon DynamoDB, puis en mettant en cache localement les éléments clés de branche utilisés dans les opérations de chiffrement et de déchiffrement. Nous vous recommandons d'utiliser les AWS KMS porte-clés dans la mesure du possible.

Pour interagir avec AWS KMS, le SDK AWS de chiffrement de base de données nécessite le AWS KMS module du AWS SDK pour Java.

Pour préparer l'utilisation du SDK de chiffrement AWS de base de données avec AWS KMS

1. Créez un Compte AWS. Pour savoir comment procéder, consultez [Comment créer et activer un nouveau compte Amazon Web Services ?](#) dans le AWS Knowledge Center.
2. Créez un chiffrement AWS KMS key symétrique. Pour obtenir de l'aide, consultez [la section Création de clés](#) dans le guide du AWS Key Management Service développeur.

i Tip

Pour l'utiliser AWS KMS key par programmation, vous aurez besoin du nom de ressource Amazon (ARN) du. AWS KMS key Pour obtenir de l'aide pour trouver l'ARN d'un AWS KMS key, consultez la section [Trouver l'ID de clé et l'ARN](#) dans le guide du AWS Key Management Service développeur.

3. Générez un identifiant de clé d'accès et une clé d'accès de sécurité. Vous pouvez utiliser l'ID de clé d'accès et la clé d'accès secrète d'un utilisateur IAM ou vous pouvez les utiliser AWS Security Token Service pour créer une nouvelle session avec des informations d'identification de sécurité temporaires, notamment un ID de clé d'accès, une clé d'accès secrète et un jeton de session. Pour des raisons de sécurité, nous vous recommandons d'utiliser des informations d'identification temporaires au lieu des informations d'identification à long terme associées à vos comptes utilisateur IAM ou utilisateur AWS (root).

Pour créer un utilisateur IAM avec une clé d'accès, consultez la section [Création d'utilisateurs IAM](#) dans le guide de l'utilisateur IAM.

Pour générer des informations d'identification de sécurité temporaires, consultez la section [Demande d'informations d'identification de sécurité temporaires](#) dans le guide de l'utilisateur IAM.

4. Définissez vos AWS informations d'identification à l'aide des instructions contenues [AWS SDK pour Java](#) dans l'ID de clé d'accès et la clé d'accès secrète que vous avez générés à l'étape 3. Si vous avez généré des informations d'identification temporaires, vous devrez également spécifier le jeton de session.

Cette procédure permet AWS SDKs de signer des AWS demandes à votre place. Les exemples de code du SDK AWS de chiffrement de base de données qui interagissent AWS KMS supposent que vous avez terminé cette étape.

Configuration du SDK de chiffrement AWS de base de données

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Le SDK AWS de chiffrement de base de données est conçu pour être facile à utiliser. Bien que le SDK AWS de chiffrement de base de données comporte plusieurs options de configuration, les valeurs par défaut sont soigneusement choisies pour être pratiques et sécurisées pour la plupart des applications. Toutefois, vous devrez peut-être ajuster votre configuration pour améliorer les performances ou inclure une fonctionnalité personnalisée dans votre conception.

Rubriques

- [Sélection d'un langage de programmation](#)
- [Sélection des clés d'emballage](#)
- [Création d'un filtre de découverte](#)
- [Utilisation de bases de données mutualisées](#)
- [Création de balises signées](#)

Sélection d'un langage de programmation

[Le SDK AWS de chiffrement de base de données pour DynamoDB est disponible dans plusieurs langages de programmation.](#) Les implémentations du langage sont conçues pour être totalement interopérables et pour offrir les mêmes fonctionnalités, bien qu'elles puissent être implémentées de différentes manières. Généralement, vous utilisez la bibliothèque compatible avec votre application.

Sélection des clés d'emballage

Le SDK AWS de chiffrement de base de données génère une clé de données symétrique unique pour chiffrer chaque champ. Vous n'avez pas besoin de configurer, de gérer ou d'utiliser les clés de données. Le SDK AWS Database Encryption le fait pour vous.

Toutefois, vous devez sélectionner une ou plusieurs clés d'encapsulation pour chiffrer chaque clé de données. Le SDK AWS de chiffrement de base de données prend en charge [AWS Key Management Service](#) (AWS KMS) les clés KMS de chiffrement symétriques et les clés RSA KMS asymétriques. Il prend également en charge les clés symétriques AES et les clés asymétriques RSA que vous fournissez en différentes tailles. Vous êtes responsable de la sécurité et de la durabilité de vos clés d'encapsulation. Nous vous recommandons donc d'utiliser une clé de chiffrement dans un module de sécurité matériel ou un service d'infrastructure clé, tel que AWS KMS.

Pour spécifier vos clés d'encapsulation pour le chiffrement et le déchiffrement, vous utilisez un [trousseau de clés](#). Selon le [type de trousseau de clés](#) que vous utilisez, vous pouvez spécifier une clé d'encapsulation ou plusieurs clés d'encapsulation de types identiques ou différents. Si vous utilisez plusieurs clés d'encapsulation pour encapsuler une clé de données, chaque clé d'encapsulation chiffrera une copie de la même clé de données. Les clés de données cryptées (une par clé d'encapsulation) sont stockées dans la [description du matériau](#) stockée à côté du champ crypté. Pour déchiffrer les données, le SDK de chiffrement AWS de base de données doit d'abord utiliser l'une de vos clés d'encapsulation pour déchiffrer une clé de données chiffrée.

Nous vous recommandons d'utiliser l'un des AWS KMS porte-clés dans la mesure du possible. Le SDK AWS de chiffrement de base de données fournit le [AWS KMS trousseau de clés](#) et le trousseau de [clés AWS KMS hiérarchique](#), ce qui réduit le nombre d'appels adressés à AWS KMS. Pour spécifier un AWS KMS key dans un trousseau de clés, utilisez un identifiant de AWS KMS clé compatible. Si vous utilisez le trousseau de clés AWS KMS hiérarchique, vous devez spécifier l'ARN de la clé. Pour plus de détails sur les identificateurs de clé d'une AWS KMS clé, consultez la section [Identifiants de clé](#) dans le guide du AWS Key Management Service développeur.

- Lorsque vous chiffrez avec un AWS KMS trousseau de clés, vous pouvez spécifier n'importe quel identifiant de clé valide (ARN de clé, nom d'alias, ARN d'alias ou ID de clé) pour une clé KMS de chiffrement symétrique. Si vous utilisez une clé RSA KMS asymétrique, vous devez spécifier l'ARN de la clé.

Si vous spécifiez un nom d'alias ou un ARN d'alias pour une clé KMS lors du chiffrement, le SDK de chiffrement de AWS base de données enregistre l'ARN de clé actuellement associé à cet alias ; il n'enregistre pas l'alias. Les modifications apportées à l'alias n'affectent pas la clé KMS utilisée pour déchiffrer vos clés de données.

- Par défaut, le AWS KMS trousseau de clés déchiffre les enregistrements en mode strict (où vous spécifiez des clés KMS particulières). Vous devez utiliser un ARN clé AWS KMS keys pour vous identifier en vue du déchiffrement.

Lorsque vous chiffrez à l'aide d'un AWS KMS trousseau de clés, le SDK AWS de chiffrement de base de données stocke l'ARN de la clé AWS KMS key dans la description du matériau avec la clé de données cryptée. Lors du déchiffrement en mode strict, le SDK de chiffrement de AWS base de données vérifie que le même ARN de clé apparaît dans le jeu de clés avant de tenter d'utiliser la clé d'encapsulation pour déchiffrer la clé de données chiffrée. Si vous utilisez un identifiant de clé différent, le SDK AWS de chiffrement de base de données ne le reconnaîtra ni ne l'utilisera AWS KMS key, même si les identifiants font référence à la même clé.

- Lors du déchiffrement en [mode découverte](#), vous ne spécifiez aucune clé d'encapsulation. Tout d'abord, le SDK AWS de chiffrement de base de données tente de déchiffrer l'enregistrement à l'aide de la clé ARN stockée dans la description du matériau. Si cela ne fonctionne pas, le SDK de chiffrement AWS de base de données demande de déchiffrer l'enregistrement AWS KMS à l'aide de la clé KMS qui l'a chiffré, quel que soit le propriétaire de cette clé KMS ou y ayant accès.

Pour spécifier une [clé AES brute](#) ou une [paire de clés RSA brute en tant que clé](#) d'encapsulation dans un trousseau de clés, vous devez spécifier un espace de noms et un nom. Lors du déchiffrement, vous devez utiliser exactement le même espace de noms et le même nom pour chaque clé d'encapsulation brute que ceux que vous avez utilisés lors du chiffrement. Si vous utilisez un espace de noms ou un nom différent, le SDK de chiffrement AWS de base de données ne reconnaîtra ni n'utilisera la clé d'encapsulation, même si le contenu de la clé est le même.

Création d'un filtre de découverte

Lorsque vous déchiffrez des données chiffrées à l'aide de clés KMS, il est recommandé de les déchiffrer en mode strict, c'est-à-dire de limiter les clés d'encapsulation utilisées à celles que vous spécifiez. Toutefois, si nécessaire, vous pouvez également déchiffrer en mode découverte, dans lequel vous ne spécifiez aucune clé d'encapsulation. Dans ce mode, AWS KMS vous pouvez déchiffrer la clé de données chiffrée à l'aide de la clé KMS qui l'a chiffrée, indépendamment de qui possède ou a accès à cette clé KMS.

Si vous devez déchiffrer en mode découverte, nous vous recommandons de toujours utiliser un filtre de découverte, qui limite les clés KMS pouvant être utilisées à celles d'une [partition Compte AWS](#) et spécifiée. Le filtre de découverte est facultatif, mais il s'agit d'une bonne pratique.

Utilisez le tableau suivant pour déterminer la valeur de partition de votre filtre de découverte.

Région	Partition
Régions AWS	aws
Régions Chine	aws-cn
AWS GovCloud (US) Regions	aws-us-gov

L'exemple suivant montre comment créer un filtre de découverte. Avant d'utiliser le code, remplacez les valeurs d'exemple par des valeurs valides pour votre partition Compte AWS and.

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()?;
```

Utilisation de bases de données mutualisées

Avec le SDK AWS Database Encryption, vous pouvez configurer le chiffrement côté client pour les bases de données avec un schéma partagé en isolant chaque client avec des matériaux de

chiffrement distincts. Lorsque vous envisagez une base de données mutualisée, prenez le temps de passer en revue vos exigences en matière de sécurité et l'impact potentiel de la mutualisation sur celles-ci. Par exemple, l'utilisation d'une base de données mutualisée peut avoir un impact sur votre capacité à combiner le SDK AWS Database Encryption avec une autre solution de chiffrement côté serveur.

Si plusieurs utilisateurs effectuent des opérations de chiffrement dans votre base de données, vous pouvez utiliser l'un des AWS KMS trousseaux de clés pour fournir à chaque utilisateur une clé distincte à utiliser dans ses opérations cryptographiques. La gestion des clés de données pour une solution de chiffrement multi-locataires côté client peut s'avérer complexe. Nous vous recommandons d'organiser vos données par locataire dans la mesure du possible. Si le locataire est identifié par les valeurs des clés primaires (par exemple, la clé de partition dans une table Amazon DynamoDB), la gestion de vos clés est plus simple.

Vous pouvez utiliser le [AWS KMS trousseau de clés](#) pour isoler chaque locataire à l'aide d'un trousseau de AWS KMS clés distinct et. AWS KMS keys En fonction du volume d' AWS KMS appels effectués par locataire, vous pouvez utiliser le trousseau de clés AWS KMS hiérarchique pour minimiser le nombre d'appels adressés à AWS KMS. Le trousseau de [clés AWS KMS hiérarchique](#) est une solution de mise en cache des matériaux cryptographiques qui réduit le nombre d' AWS KMS appels en utilisant des clés de branche AWS KMS protégées conservées dans une table Amazon DynamoDB, puis en mettant en cache localement les éléments clés de branche utilisés dans les opérations de chiffrement et de déchiffrement. Vous devez utiliser le trousseau de clés AWS KMS hiérarchique pour implémenter le [chiffrement consultable](#) dans votre base de données.

Création de balises signées

Le SDK AWS Database Encryption utilise des balises [standard et des balises composées](#) pour fournir des solutions de [chiffrement consultables](#) qui vous permettent de rechercher des enregistrements cryptés sans déchiffrer l'intégralité de la base de données interrogée. Toutefois, le SDK AWS Database Encryption prend également en charge les balises signées qui peuvent être entièrement configurées à partir de champs signés en texte brut. Les balises signées sont un type de balise composée qui indexe et exécute des requêtes complexes sur des SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT champs SIGN_ONLY et des champs.

Par exemple, si vous avez une base de données mutualisée, vous souhaitez peut-être créer une balise signée qui vous permettra de rechercher dans votre base de données des enregistrements chiffrés par la clé d'un locataire spécifique. Pour de plus amples informations, veuillez consulter [Interrogation de balises dans une base de données mutualisée](#).

Vous devez utiliser le trousseau de clés AWS KMS hiérarchique pour créer des balises signées.

Pour configurer une balise signée, fournissez les valeurs suivantes.

Java

Configuration de balise composée

L'exemple suivant définit les listes de pièces signées localement dans la configuration des balises signées.

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

Définition de la version de la balise

L'exemple suivant définit les listes de pièces signées globalement dans la version balise. Pour plus d'informations sur la définition de la version des balises, consultez la section [Utilisation des balises](#).

```
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

Voir l'exemple de code complet : [BeaconConfig.cs](#)

Configuration des balises signées

L'exemple suivant définit les listes de pièces signées localement dans la configuration des balises signées.

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

Définition de la version de la balise

L'exemple suivant définit les listes de pièces signées globalement dans la version balise. Pour plus d'informations sur la définition de la version des balises, consultez la section [Utilisation des balises](#).

```
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
            }
        }
    }
}
```

```
};
```

Vous pouvez définir vos pièces signées dans des listes définies localement ou globalement. Nous vous recommandons de définir vos pièces signées dans une liste globale dans la [version balise](#) dans la mesure du possible. En définissant les pièces signées de manière globale, vous pouvez définir chaque pièce une seule fois, puis les réutiliser dans plusieurs configurations de balises composées. Si vous n'avez l'intention d'utiliser une pièce signée qu'une seule fois, vous pouvez la définir dans une liste locale dans la configuration des balises signées. Vous pouvez référencer des pièces locales et globales dans votre [liste de constructeurs](#).

Si vous définissez vos listes de pièces signées globalement, vous devez fournir une liste de pièces du constructeur identifiant toutes les manières possibles dont la balise signée peut assembler les champs dans la configuration de votre balise.

Note

Pour définir des listes de pièces signées de manière globale, vous devez utiliser la version 3.2 ou ultérieure du SDK AWS Database Encryption. Déployez la nouvelle version auprès de tous les lecteurs avant de définir de nouvelles pièces de manière globale.

Vous ne pouvez pas mettre à jour les configurations de balises existantes pour définir des listes de pièces signées de manière globale.

Nom de la balise

Le nom que vous utilisez lorsque vous interrogez la balise.

Le nom d'une balise signé ne peut pas être le même que celui d'un champ non chiffré. Deux balises ne peuvent pas porter le même nom de balise.

Caractère divisé

Le caractère utilisé pour séparer les parties qui composent votre balise signée.

Le caractère divisé ne peut apparaître dans les valeurs en texte brut d'aucun des champs à partir desquels la balise signée est construite.

Liste de pièces signée

Identifie les champs signés inclus dans la balise signée.

Chaque partie doit inclure un nom, une source et un préfixe. La source est le `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` champ `SIGN_ONLY` ou que l'article identifie. La source doit être un nom de champ ou un index faisant référence à la valeur d'un champ imbriqué. Si le nom de votre pièce identifie la source, vous pouvez omettre la source et le SDK AWS de chiffrement de base de données utilisera automatiquement le nom comme source. Nous recommandons de spécifier la source comme nom de pièce dans la mesure du possible. Le préfixe peut être n'importe quelle chaîne, mais il doit être unique. Deux parties signées d'une balise signée ne peuvent pas avoir le même préfixe. Nous recommandons d'utiliser une valeur courte qui distingue la pièce des autres parties desservies par la balise composée.

Nous vous recommandons de définir vos pièces signées de manière globale dans la mesure du possible. Vous pouvez envisager de définir une pièce signée localement si vous avez l'intention de ne l'utiliser que dans une seule balise composée. Une pièce définie localement ne peut pas avoir le même préfixe ou le même nom qu'une pièce définie globalement.

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-" }
};
```

Liste des constructeurs (facultatif)

Identifie les constructeurs qui définissent les différentes manières dont les pièces signées peuvent être assemblées par la balise signée.

Si vous ne spécifiez pas de liste de constructeurs, le SDK AWS Database Encryption assemble la balise signée avec le constructeur par défaut suivant.

- Toutes les pièces signées dans l'ordre dans lequel elles ont été ajoutées à la liste des pièces signées

- Toutes les pièces sont requises

Constructeurs

Chaque constructeur est une liste ordonnée de pièces du constructeur qui définit une manière dont la balise signée peut être assemblée. Les pièces du constructeur sont assemblées dans l'ordre dans lequel elles sont ajoutées à la liste, chaque partie étant séparée par le caractère divisé spécifié.

Chaque partie du constructeur nomme une pièce signée et définit si cette partie est obligatoire ou facultative dans le constructeur. Par exemple, si vous souhaitez interroger une balise signée sur `Field1`, `Field1.Field2`, et `Field1.Field2.Field3`, marquer `Field2` et `Field3` comme facultatif et créer un constructeur.

Chaque constructeur doit avoir au moins une pièce requise. Nous vous recommandons de rendre obligatoire la première partie de chaque constructeur afin que vous puissiez utiliser l'`BEGINS_WITH` opérateur dans vos requêtes.

Un constructeur réussit si toutes ses pièces requises sont présentes dans l'enregistrement. Lorsque vous écrivez un nouvel enregistrement, la balise signée utilise la liste des constructeurs pour déterminer si la balise peut être assemblée à partir des valeurs fournies. Il tente d'assembler la balise dans l'ordre dans lequel les constructeurs ont été ajoutés à la liste des constructeurs, et il utilise le premier constructeur qui réussit. Si aucun constructeur ne réussit, la balise n'est pas écrite dans l'enregistrement.

Tous les lecteurs et rédacteurs doivent spécifier le même ordre de constructeurs pour s'assurer que les résultats de leurs requêtes sont corrects.

Utilisez les procédures suivantes pour spécifier votre propre liste de constructeurs.

1. Créez une pièce constructeur pour chaque pièce signée afin de définir si cette pièce est requise ou non.

Le nom de la partie constructeur doit être le nom du champ signé.

L'exemple suivant montre comment créer une partie constructeur pour un champ signé.

Java

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

```
.build();
```

C# / .NET

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

2. Créez un constructeur pour chaque manière possible d'assembler la balise signée à l'aide des pièces du constructeur que vous avez créées à l'étape 1.

Par exemple, si vous souhaitez effectuer une requête sur `Field1.Field2.Field3` et `Field4.Field2.Field3`, vous devez créer deux constructeurs. `Field1` et `Field4` peuvent tous deux être requis car ils sont définis dans deux constructeurs distincts.

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. Créez une liste de constructeurs qui inclut tous les constructeurs que vous avez créés à l'étape 2.

Java

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

C# / .NET

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

4. Spécifiez le constructorList moment où vous créez votre balise signée.

Stockage des clés dans le SDK de chiffrement des AWS bases de données

[Dans le SDK AWS Database Encryption, un magasin de clés est une table Amazon DynamoDB qui conserve les données hiérarchiques utilisées AWS KMS par le trousseau de clés hiérarchique.](#) Le magasin de clés permet de réduire le nombre d'appels que vous devez effectuer pour AWS KMS effectuer des opérations cryptographiques avec le trousseau de clés hiérarchique.

Le magasin de clés conserve et gère les clés de branche que le trousseau de clés hiérarchique utilise pour chiffrer les enveloppes et protéger les clés de chiffrement des données. Le magasin de clés stocke la clé de branche active et toutes les versions précédentes de la clé de branche. La clé de branche active est la version de clé de branche la plus récente. Le trousseau de clés hiérarchique utilise une clé de chiffrement des données unique pour chaque demande de chiffrement et chiffre chaque clé de chiffrement des données à l'aide d'une clé d'encapsulation unique dérivée de la clé de branche active. Le trousseau de clés hiérarchique dépend de la hiérarchie établie entre les clés de branche actives et leurs clés d'encapsulation dérivées.

Terminologie et concepts clés du magasin

Key store (Magasin de clés)

Table DynamoDB qui conserve les données hiérarchiques, telles que les clés de branche et les clés de balise.

Clé racine

Une clé KMS de chiffrement symétrique qui génère et protège les clés de branche et les clés de balise de votre magasin de clés.

Clé de branche

Clé de données réutilisée pour obtenir une clé d'encapsulation unique pour le chiffrement des enveloppes. Vous pouvez créer plusieurs clés de branche dans un même magasin de clés, mais chaque clé de branche ne peut avoir qu'une seule version de clé de branche active à la fois. La clé de branche active est la version de clé de branche la plus récente.

Les clés de branche sont dérivées de AWS KMS keys l'utilisation de l'`GenerateDataKeyWithoutPlaintext` opération [kms](#) .

Clé d'emballage

Clé de données unique utilisée pour chiffrer la clé de chiffrement des données utilisée dans les opérations de chiffrement.

Les clés d'encapsulation sont dérivées des clés de branche. Pour plus d'informations sur le processus de dérivation des clés, voir Détails [techniques du trousseau de clés AWS KMS hiérarchique](#).

Clé de chiffrement des données

Clé de données utilisée dans les opérations de chiffrement. Le trousseau de clés hiérarchique utilise une clé de chiffrement des données unique pour chaque demande de chiffrement.

Clé de balise

Clé de données utilisée pour générer des balises à des fins de chiffrement consultable. Pour plus d'informations, consultez la section [Chiffrement consultable](#).

Implémentation des autorisations avec le moindre privilégié

Lorsque vous utilisez un magasin de clés et des trousseaux de clés AWS KMS hiérarchiques, nous vous recommandons de suivre le principe du moindre privilège en définissant les rôles suivants :

Administrateur du magasin de clés

Les administrateurs du magasin de clés sont chargés de créer et de gérer le magasin de clés et les clés de succursale qu'il conserve et protège. Les administrateurs de magasin de clés doivent être les seuls utilisateurs autorisés à écrire sur la table Amazon DynamoDB qui vous sert de magasin de clés. Ils doivent être les seuls utilisateurs ayant accès à des opérations d'administrateur privilégiées, telles que [CreateKey](#) et [VersionKey](#). Vous ne pouvez effectuer ces opérations que lorsque vous [configurez de manière statique les actions de votre magasin de clés](#).

`CreateKey` est une opération privilégiée qui peut ajouter un nouvel ARN de clé KMS à votre liste d'autorisations de stockage de clés. Cette clé KMS peut créer de nouvelles clés de branche actives. Nous recommandons de limiter l'accès à cette opération car une fois qu'une clé KMS est ajoutée au magasin de clés de succursale, elle ne peut pas être supprimée.

Utilisateur du magasin de clés

Dans la plupart des cas d'utilisation, l'utilisateur du magasin de clés n'interagit avec le magasin de clés que par le biais du trousseau de clés hiérarchique lorsqu'il chiffre, déchiffre, signe et vérifie les données. Par conséquent, ils n'ont besoin que d'autorisations de lecture pour accéder à la table Amazon DynamoDB qui sert de magasin de clés. Les utilisateurs du magasin de clés ne doivent avoir accès qu'aux opérations d'utilisation qui rendent les opérations cryptographiques possibles, telles que `GetActiveBranchKey`, `GetBranchKeyVersion`, et `GetBeaconKey`. Ils n'ont pas besoin d'autorisations pour créer ou gérer les clés de branche qu'ils utilisent.

Vous pouvez effectuer des opérations d'utilisation lorsque les actions de votre magasin de clés sont [configurées de manière statique](#) ou lorsqu'elles sont configurées pour la [découverte](#). Vous ne pouvez pas effectuer d'opérations d'administrateur (`CreateKeyetVersionKey`) lorsque les actions de votre magasin de clés sont configurées pour la découverte.

Si l'administrateur de votre magasin de clés de succursale a autorisé plusieurs clés KMS dans votre magasin de clés de succursale, nous recommandons aux utilisateurs de votre magasin de clés de configurer leurs actions de magasin de clés à des fins de découverte afin que leur jeu de clés hiérarchique puisse utiliser plusieurs clés KMS.

Créez un magasin de clés

Avant de [créer des clés de branche](#) ou d'utiliser un trousseau de [clés AWS KMS hiérarchique](#), vous devez créer votre magasin de clés, une table Amazon DynamoDB qui gère et protège vos clés de branche.

Important

Ne supprimez pas la table DynamoDB qui conserve vos clés de branche. Si vous supprimez ce tableau, vous ne pourrez pas déchiffrer les données chiffrées à l'aide du trousseau de clés hiérarchique.

Suivez les procédures de [création d'une table](#) du guide du développeur Amazon DynamoDB, en utilisant les valeurs de chaîne requises suivantes pour la clé de partition et la clé de tri.

	Clé de partition	Clé de tri
Table de base	branch-key-id	type

Nom du magasin de clés logique

Lorsque vous nommez la table DynamoDB qui sert de banque de clés, il est important de prendre soigneusement en compte le nom logique de la banque de clés que vous allez spécifier [lors de la configuration de vos actions de banque de clés](#). Le nom logique du magasin de clés sert d'identifiant à votre magasin de clés et ne peut pas être modifié une fois qu'il a été initialement défini par le premier utilisateur. Vous devez toujours spécifier le même nom logique de banque de clés dans vos [actions de banque de clés](#).

Il doit y avoir un one-to-one mappage entre le nom de la table DynamoDB et le nom du magasin de clés logiques. Le nom du magasin de clés logique est lié de manière cryptographique à toutes les données stockées dans la table afin de simplifier les opérations de restauration DynamoDB. Bien que le nom du magasin de clés logique puisse être différent du nom de votre table DynamoDB, nous vous recommandons vivement de spécifier le nom de votre table DynamoDB comme nom de magasin de clés logique. Si le nom de votre table change après avoir [restauré votre table DynamoDB à partir d'une sauvegarde, le nom logique du magasin de clés peut être mappé au nouveau nom de table](#) DynamoDB afin de garantir que le trousseau de clés hiérarchique peut toujours accéder à votre magasin de clés.

N'incluez pas d'informations confidentielles ou sensibles dans le nom de votre banque de clés logique. Le nom du magasin de clés logique est affiché en texte clair dans les AWS KMS CloudTrail événements sous la forme `detablename`.

Étapes suivantes

1. [the section called “Configurer les actions clés du magasin”](#)
2. [the section called “Création de clés de branche”](#)
3. [Création d'un porte-clés AWS KMS hiérarchique](#)

Configurer les actions clés du magasin

Les actions du magasin de clés déterminent les opérations que vos utilisateurs peuvent effectuer et la manière dont leur AWS KMS jeu de clés hiérarchique utilise les clés KMS autorisées dans votre magasin de clés. Le SDK AWS Database Encryption prend en charge les configurations d'actions de stockage de clés suivantes.

Statique

Lorsque vous configurez votre magasin de clés de manière statique, celui-ci ne peut utiliser que la clé KMS associée à l'ARN de clé KMS que vous fournissez `kmsConfiguration` lors de la configuration de vos actions de magasin de clés. Une exception est déclenchée si un ARN de clé KMS différent est rencontré lors de la création, de la gestion des versions ou de l'obtention d'une clé de branche.

Vous pouvez spécifier une clé KMS multirégionale dans votre `kmsConfiguration`, mais l'ensemble de l'ARN de la clé, y compris la région, est conservé dans les clés de branche dérivées de la clé KMS. Vous ne pouvez pas spécifier de clé dans une autre région, vous devez fournir exactement la même clé multirégionale pour que les valeurs correspondent.

Lorsque vous configurez de manière statique les actions de votre magasin de clés, vous pouvez effectuer des opérations d'utilisation (`GetActiveBranchKey`, `GetBranchKeyVersion`, `GetBeaconKey`) et des opérations administratives (`CreateKey` et `VersionKey`). `CreateKey` est une opération privilégiée qui peut ajouter un nouvel ARN de clé KMS à votre liste d'autorisations de stockage de clés. Cette clé KMS peut créer de nouvelles clés de branche actives. Nous recommandons de limiter l'accès à cette opération car une fois qu'une clé KMS est ajoutée au magasin de clés, elle ne peut pas être supprimée.

Découverte

Lorsque vous configurez les actions de votre magasin de clés pour la découverte, le magasin de clés peut utiliser n'importe quel AWS KMS key ARN autorisé dans votre magasin de clés. Toutefois, une exception est déclenchée lorsqu'une clé KMS multirégionale est détectée et que la région dans l'ARN de la clé ne correspond pas à la région du AWS KMS client utilisé.

Lorsque vous configurez votre banque de clés pour la découverte, vous ne pouvez pas effectuer d'opérations administratives telles que `CreateKey` et `VersionKey`. Vous ne pouvez effectuer que les opérations d'utilisation qui permettent de chiffrer, de déchiffrer, de signer et de vérifier.

Pour de plus amples informations, veuillez consulter [the section called “Implémentation des autorisations avec le moindre privilégié”](#).

Configurez les actions clés de votre boutique

Avant de configurer les actions de votre magasin de clés, assurez-vous que les conditions préalables suivantes sont remplies.

- Déterminez les opérations que vous devez effectuer. Pour de plus amples informations, veuillez consulter [the section called “Implémentation des autorisations avec le moindre privilégié”](#).
- Choisissez un nom de magasin de clés logique

Il doit y avoir un one-to-one mappage entre le nom de la table DynamoDB et le nom du magasin de clés logiques. Le nom du magasin de clés logique est lié de manière cryptographique à toutes les données stockées dans la table afin de simplifier les opérations de restauration DynamoDB. Il ne peut pas être modifié une fois défini initialement par le premier utilisateur. Vous devez toujours spécifier le même nom logique de banque de clés dans les actions de votre banque de clés. Pour de plus amples informations, veuillez consulter [logical key store name](#).

Configuration statique

L'exemple suivant configure de manière statique les actions du magasin de clés. Vous devez spécifier le nom de la table DynamoDB qui sert de magasin de clés, un nom logique pour le magasin de clés et l'ARN de clé KMS qui identifie une clé KMS de chiffrement symétrique.

Note

Examinez attentivement l'ARN de la clé KMS que vous spécifiez lors de la configuration statique de votre service de banque de clés. L'`CreateKey` opération ajoute l'ARN de la clé KMS à la liste d'autorisation de votre magasin de clés de succursale. Une fois qu'une clé KMS est ajoutée au magasin de clés de succursale, elle ne peut pas être supprimée.

Java

```
final KeyStore keystore = KeyStore.builder().KeyStoreConfig(  
    KeyStoreConfig.builder()
```

```

        .ddbClient(DynamoDbClient.create())
        .ddbTableName(keyStoreName)
        .logicalKeyName(logicalKeyName)
        .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(),
    LogicalKeyName = logicalKeyName
};
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)?;

```

Configuration de la découverte

L'exemple suivant configure les actions du magasin de clés pour la découverte. Vous devez spécifier le nom de la table DynamoDB qui sert de magasin de clés et un nom de magasin de clés logique.

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()?;
```

Création d'une clé de branche active

Une clé de branche est une clé de données dérivée d'une clé AWS KMS key que le trousseau de clés AWS KMS hiérarchique utilise pour réduire le nombre d'appels effectués. AWS KMS La clé de

branche active est la version de clé de branche la plus récente. Le trousseau de clés hiérarchique génère une clé de données unique pour chaque demande de chiffrement et chiffre chaque clé de données avec une clé d'encapsulation unique dérivée de la clé de branche active.

Pour créer une nouvelle clé de branche active, vous devez [configurer de manière statique les](#) actions de votre magasin de clés. `CreateKey` est une opération privilégiée qui ajoute l'ARN de la clé KMS spécifié dans la configuration des actions de votre magasin de clés à votre liste d'autorisations de stockage de clés. Ensuite, la clé KMS est utilisée pour générer la nouvelle clé de branche active. Nous recommandons de limiter l'accès à cette opération car une fois qu'une clé KMS est ajoutée au magasin de clés, elle ne peut pas être supprimée.

Nous vous recommandons d'utiliser l'`CreateKey` opération via l'interface `KeyStore` d'administration du plan de contrôle de votre application. Cette approche est conforme aux meilleures pratiques en matière de gestion des clés.

Ne créez pas de clés de branche dans le plan de données. Cette pratique peut entraîner :

- Appels inutiles à AWS KMS
- Plusieurs appels simultanés AWS KMS dans des environnements à haute simultanéité
- Plusieurs `TransactWriteItems` appels à la table DynamoDB de base.

L'`CreateKey` opération inclut une vérification de l'état de l'`TransactWriteItems` appel afin d'éviter de remplacer les clés de branche existantes. Cependant, la création de clés dans le plan de données peut toujours entraîner une utilisation inefficace des ressources et des problèmes de performances potentiels.

Vous pouvez autoriser la mise en liste d'une clé KMS dans votre banque de clés, ou vous pouvez autoriser la mise en liste de plusieurs clés KMS en mettant à jour l'ARN de la clé KMS que vous spécifiez dans la configuration des actions de votre banque de clés et en appelant `CreateKey` à nouveau. Si vous autorisez plusieurs clés KMS sur liste, les utilisateurs de votre magasin de clés doivent configurer leurs actions de magasin de clés pour la découverte afin qu'ils puissent utiliser n'importe laquelle des clés autorisées dans le magasin de clés auquel ils ont accès. Pour de plus amples informations, veuillez consulter [the section called "Configurer les actions clés du magasin"](#).

Autorisations nécessaires

Pour créer des clés de branche, vous avez besoin des `ReEncrypt` autorisations [kms : `GenerateDataKeyWithoutPlaintext`](#) et [kms :](#) sur la clé KMS spécifiée dans les actions de votre magasin de clés.

Création d'une clé de branche

L'opération suivante crée une nouvelle clé de branche active à l'aide de la clé KMS que vous avez [spécifiée dans la configuration des actions de votre magasin de clés](#), et ajoute la clé de branche active à la table DynamoDB qui sert de magasin de clés.

Lorsque vous appelez `CreateKey`, vous pouvez choisir de spécifier les valeurs facultatives suivantes.

- `branchKeyIdentifier`: définit une coutume `branch-key-id`.

Pour créer une personnalisation `branch-key-id`, vous devez également inclure un contexte de chiffrement supplémentaire dans le `encryptionContext` paramètre.

- `encryptionContext`: [définit un ensemble facultatif de paires clé-valeur non secrètes qui fournissent des données authentifiées supplémentaires \(AAD\) dans le contexte de chiffrement inclus dans l'appel `kms : GenerateDataKeyWithoutPlaintext`](#)

Ce contexte de chiffrement supplémentaire est affiché avec le `aws-crypto-ec`: préfixe.

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_identifiant("custom-branch-key-id") // OPTIONAL
    .encryption_context(additional_encryption_context) // OPTIONAL
    .send()
    .await?
    .branch_key_identifiant
    .unwrap();
```

Tout d'abord, l'CreateKeyopération génère les valeurs suivantes.

- Un [identifiant unique universel](#) (UUID) de version 4 pour le branch-key-id (sauf si vous avez spécifié un identifiant personnalisébranch-key-id).
- Un UUID version 4 pour la version de la clé de branche
- A timestamp au [format de date et d'heure ISO 8601](#) en temps universel coordonné (UTC).

Ensuite, l'CreateKeyopération appelle [kms : GenerateDataKeyWithoutPlaintext](#) en utilisant la requête suivante.

```
{
  "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

L'opération `CreateKey` crée une clé de branche active et une clé de balise, même si vous n'avez pas configuré votre base de données pour un [chiffrement consultable](#). Les deux clés sont stockées dans votre magasin de clés. Pour plus d'informations, voir [Utilisation du trousseau de clés hiérarchique pour le chiffrement consultable](#).

Ensuite, l'opération `CreateKey` appelle [kms : ReEncrypt](#) pour créer un enregistrement actif pour la clé de branche en mettant à jour le contexte de chiffrement.

Enfin, l'opération `CreateKey` appelle [ddb : TransactWriteItems](#) pour écrire un nouvel élément qui conservera la clé de branche dans la table que vous avez créée à l'étape 2. L'article possède les attributs suivants.

```
{
  "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"
}
```

Faites pivoter votre clé de branche active

Il ne peut y avoir qu'une seule version active à la fois pour chaque clé de branche. En général, chaque version de clé de branche active est utilisée pour satisfaire plusieurs demandes. Mais vous contrôlez la mesure dans laquelle les clés de branche actives sont réutilisées et vous déterminez la fréquence à laquelle la clé de branche active est pivotée.

Les clés de branche ne sont pas utilisées pour chiffrer les clés de données en texte brut. Ils sont utilisés pour dériver les clés d'encapsulation uniques qui chiffrent les clés de données en texte brut. Le [processus de dérivation de la clé d'encapsulation](#) produit une clé d'encapsulation unique de 32 octets avec 28 octets aléatoires. Cela signifie qu'une clé de branche peut obtenir plus de 79 octillions,

soit 2^{96} , clés d'encapsulation uniques avant que l'usure cryptographique ne se produise. Malgré ce très faible risque d'épuisement, il se peut que vous deviez effectuer une rotation de vos clés de succursale actives en raison de règles commerciales, contractuelles ou gouvernementales.

La version active de la clé de branche reste active jusqu'à ce que vous la fassiez pivoter. Les versions précédentes de la clé de branche active ne seront pas utilisées pour effectuer des opérations de chiffrement et ne peuvent pas être utilisées pour obtenir de nouvelles clés d'encapsulation, mais elles peuvent toujours être interrogées et fournir des clés d'encapsulation pour déchiffrer les clés de données qu'elles chiffraient lorsqu'elles étaient actives.

Warning

La suppression des clés de branche dans les environnements de test est irréversible. Vous ne pouvez pas récupérer les clés de branche supprimées. Lorsque vous supprimez et recréez des clés de branche avec le même identifiant dans des environnements de test, les problèmes suivants peuvent survenir :

- Les matériaux des tests précédents peuvent rester dans le cache
- Certains hôtes ou threads de test peuvent chiffrer des données à l'aide de clés de branche supprimées
- Les données chiffrées avec des branches supprimées ne peuvent pas être déchiffrées

Pour éviter les échecs de chiffrement lors des tests d'intégration :

- Réinitialisez la référence hiérarchique du trousseau de clés avant de créer de nouvelles clés de branche OU
- Utiliser une clé de branche unique IDs pour chaque test

Autorisations requises

Pour faire pivoter les clés de branche, vous avez besoin des ReEncrypt autorisations [kms : GenerateDataKeyWithoutPlaintext](#) et [kms :](#) sur la clé KMS spécifiée dans les actions de votre magasin de clés.

Faire pivoter une clé de branche active

Utilisez cette `VersionKey` opération pour faire pivoter votre clé de branche active. Lorsque vous faites pivoter la clé de branche active, une nouvelle clé de branche est créée pour remplacer la

version précédente. `branch-key-id` Cela ne change pas lorsque vous faites pivoter la clé de branche active. Vous devez spécifier le code `branch-key-id` qui identifie la clé de branche active actuelle lorsque vous appelez `VersionKey`.

Java

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

C# / .NET

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

Rust

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

Porte-clés

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Le SDK AWS Database Encryption utilise des trousseaux de clés pour chiffrer les [enveloppes](#). Les porte-clés génèrent, chiffrent et déchiffrent des clés de données. Les porte-clés déterminent la source des clés de données uniques qui protègent chaque enregistrement chiffré, ainsi que les [clés d'encapsulation qui chiffrent cette clé](#) de données. Vous spécifiez un porte-clés lors du chiffrement et le même porte-clés ou un autre porte-clés lors du déchiffrement.

Vous pouvez utiliser chaque porte-clés individuellement ou combiner les porte-clés dans un [porte-clés multiple](#). Bien que la plupart des porte-clés peuvent générer, chiffrer et déchiffrer des clés de données, vous pouvez créer un porte-clés qui effectue une seule opération particulière, par exemple un porte-clés qui génère uniquement des clés de données, et utiliser ce porte-clés en combinaison avec d'autres.

Nous vous recommandons d'utiliser un trousseau de clés qui protège vos clés d'encapsulation et effectue des opérations cryptographiques dans une limite sécurisée, tel que le AWS KMS trousseau de clés, qui utilise AWS KMS keys that never leave [AWS Key Management Service](#)(AWS KMS non chiffré). Vous pouvez également créer un trousseau de clés utilisant des clés d'encapsulation stockées dans vos modules de sécurité matériels (HSMs) ou protégées par d'autres services de clés principales.

Votre trousseau de clés détermine les clés d'encapsulation qui protègent vos clés de données et, en fin de compte, vos données. Utilisez les clés d'emballage les plus sûres et les plus pratiques pour votre tâche. Dans la mesure du possible, utilisez des clés d'encapsulation protégées par un module de sécurité matériel (HSM) ou une infrastructure de gestion des clés, comme les clés KMS in [AWS Key Management Service](#)(AWS KMS) ou les clés de chiffrement in [AWS CloudHSM](#).

Le SDK AWS Database Encryption fournit plusieurs ensembles de clés et configurations de trousseaux de clés, et vous pouvez créer vos propres trousseaux de clés personnalisés. Vous pouvez également créer un [porte-clés multiple](#) comprenant un ou plusieurs porte-clés du même type ou d'un type différent.

Rubriques

- [Fonctionnement des porte-clés](#)
- [AWS KMS porte-clés](#)
- [AWS KMS Porte-clés hiérarchiques](#)
- [AWS KMS Porte-clés ECDH](#)
- [Porte-clés AES brut](#)
- [Porte-clés RSA bruts](#)
- [Porte-clés ECDH bruts](#)
- [Porte-clés multiples](#)

Fonctionnement des porte-clés

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Lorsque vous chiffrez et signez un champ dans votre base de données, le SDK de chiffrement de AWS base de données demande au trousseau de clés le matériel de chiffrement. Le porte-clés renvoie une clé de données en texte brut, une copie de la clé de données chiffrée par chacune des clés d'encapsulation du trousseau de clés et une clé MAC associée à la clé de données. Le SDK AWS de chiffrement de base de données utilise la clé en texte brut pour chiffrer les données, puis la supprime de la mémoire dès que possible. Le SDK AWS de chiffrement de base de données ajoute ensuite une [description matérielle](#) qui inclut les clés de données chiffrées et d'autres informations, telles que les instructions de chiffrement et de signature. Le SDK AWS Database Encryption utilise la clé MAC pour calculer les codes d'authentification des messages basés sur le hachage (HMACs) lors de la canonisation de la description du matériel et de tous les champs marqués ou. ENCRYPT_AND_SIGN SIGN_ONLY

Lorsque vous déchiffrez des données, vous pouvez utiliser le même trousseau de clés que celui que vous avez utilisé pour chiffrer les données, ou un autre. Pour déchiffrer les données, un jeu de clés de déchiffrement doit avoir accès à au moins une clé d'encapsulation du jeu de clés de chiffrement.

Le SDK AWS de chiffrement de base de données transmet les clés de données chiffrées de la description matérielle au trousseau de clés et demande au trousseau de déchiffrer l'une d'entre elles.

Le porte-clés utilise ses clés d'encapsulation pour déchiffrer l'une des clés de données chiffrées et renvoie une clé de données en texte brut. Le SDK AWS de chiffrement de base de données utilise la clé de données en texte brut pour déchiffrer les données. Si aucune des clés d'encapsulation du porte-clés ne peut déchiffrer les clés de données chiffrées, l'opération de déchiffrement échoue.

Vous pouvez utiliser un seul porte-clés ou également combiner des porte-clés du même type ou de types différents dans un [porte-clés multiple](#). Lorsque vous chiffrez des données, le jeu de clés multiples renvoie une copie de la clé de données chiffrée par toutes les clés d'encapsulation de tous les trousseaux de clés qui le composent et une clé MAC associée à la clé de données. Vous pouvez déchiffrer les données à l'aide d'un trousseau de clés avec n'importe laquelle des clés d'encapsulation du trousseau de clés multiples.

AWS KMS porte-clés

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Un AWS KMS trousseau de clés utilise le chiffrement symétrique ou le RSA asymétrique [AWS KMS keys](#) pour générer, chiffrer et déchiffrer les clés de données. AWS Key Management Service (AWS KMS) protège vos clés KMS et effectue des opérations cryptographiques dans les limites de la norme FIPS. Nous vous recommandons d'utiliser un AWS KMS trousseau de clés, ou un trousseau de clés présentant des propriétés de sécurité similaires, dans la mesure du possible.

Vous pouvez également utiliser une clé KMS multirégionale symétrique dans un AWS KMS trousseau de clés. Pour plus de détails et des exemples d'utilisation de plusieurs régions AWS KMS keys, consultez [Utilisation de plusieurs régions AWS KMS keys](#). Pour plus d'informations sur les clés multirégionales, consultez la section [Utilisation des clés multirégionales](#) dans le manuel du AWS Key Management Service développeur.

AWS KMS les porte-clés peuvent inclure deux types de clés d'emballage :

- Clé du générateur : génère une clé de données en texte brut et la chiffre. Un trousseau de clés qui chiffre des données doit comporter une clé génératrice.
- Clés supplémentaires : chiffre la clé de données en texte brut générée par la clé du générateur. AWS KMS les porte-clés peuvent comporter zéro ou plusieurs clés supplémentaires.

Vous devez disposer d'une clé génératrice pour chiffrer les enregistrements. Lorsqu'un AWS KMS trousseau de clés ne comporte qu'une seule AWS KMS clé, celle-ci est utilisée pour générer et chiffrer la clé de données.

Comme tous les porte-clés, les AWS KMS porte-clés peuvent être utilisés indépendamment ou dans un [porte-clés multiple avec d'autres porte-clés](#) du même type ou d'un type différent.

Rubriques

- [Autorisations requises pour les AWS KMS porte-clés](#)
- [Identification AWS KMS keys dans un AWS KMS porte-clés](#)
- [Création d'un AWS KMS porte-clés](#)
- [Utilisation de plusieurs régions AWS KMS keys](#)
- [Utilisation d'un porte-clés AWS KMS Discovery](#)
- [Utiliser un porte-clés de découverte AWS KMS régional](#)

Autorisations requises pour les AWS KMS porte-clés

Le SDK AWS de chiffrement de base de données n'en nécessite pas Compte AWS et n'en dépend pas Service AWS. Toutefois, pour utiliser un AWS KMS trousseau de clés, vous devez disposer des autorisations minimales suivantes AWS KMS keys sur celui-ci. Compte AWS

- Pour chiffrer avec un AWS KMS trousseau de clés, vous avez besoin de l'GenerateDataKey autorisation [kms](#) : sur la clé du générateur. Vous devez disposer de l'autorisation [KMS:Encrypt pour](#) toutes les clés supplémentaires du trousseau de clés. AWS KMS
- Pour déchiffrer avec un AWS KMS trousseau de clés, vous devez disposer de l'autorisation [KMS:Decrypt](#) sur au moins une des clés du trousseau de clés. AWS KMS
- Pour chiffrer avec un trousseau de clés multiples composé de trousseaux de AWS KMS clés, vous avez besoin de l'GenerateDataKey autorisation [kms](#) : sur la clé du générateur située dans le trousseau de clés du générateur. Vous avez besoin de l'autorisation [KMS:Encrypt](#) sur toutes les autres clés de tous les autres trousseaux de clés. AWS KMS
- Pour chiffrer avec un jeu de AWS KMS clés RSA asymétrique, vous n'avez pas besoin de [kms](#) : [GenerateDataKey](#) ou de [KMS:Encrypt](#) car vous devez spécifier le matériel de clé publique que vous souhaitez utiliser pour le chiffrement lorsque vous créez le trousseau de clés. Aucun AWS KMS appel n'est effectué lors du chiffrement avec ce trousseau de clés. [Pour déchiffrer avec un trousseau de AWS KMS clés RSA asymétrique, vous devez disposer de l'autorisation KMS:Decrypt.](#)

Pour obtenir des informations détaillées sur les autorisations pour AWS KMS keys, voir [Authentification et contrôle d'accès](#) dans le Guide du AWS Key Management Service développeur.

Identification AWS KMS keys dans un AWS KMS porte-clés

Un AWS KMS porte-clés peut en inclure un ou plusieurs AWS KMS keys. Pour spécifier un AWS KMS key dans un AWS KMS trousseau de clés, utilisez un identifiant de AWS KMS clé compatible. Les identificateurs de clé que vous pouvez utiliser pour identifier un élément AWS KMS key dans un trousseau de clés varient en fonction de l'opération et de l'implémentation du langage. Pour plus de détails sur les identificateurs de clé d'un AWS KMS key, consultez la section [Identifiants de clé](#) dans le guide du AWS Key Management Service développeur.

Il est recommandé d'utiliser l'identifiant de clé le plus précis qui soit adapté à votre tâche.

- Pour chiffrer à l'aide d'un AWS KMS trousseau de clés, vous pouvez utiliser un [identifiant de clé](#), un [ARN](#) de clé, un [nom d'alias](#) ou un [ARN](#) d'alias pour chiffrer les données.

Note

Si vous spécifiez un nom d'alias ou un alias ARN pour une clé KMS dans un jeu de clés de chiffrement, l'opération de chiffrement enregistre l'ARN de clé actuellement associé à l'alias dans les métadonnées de la clé de données chiffrée. Cela n'enregistre pas l'alias. Les modifications apportées à l'alias n'affectent pas la clé KMS utilisée pour déchiffrer vos clés de données chiffrées.

- Pour déchiffrer à l'aide d'un AWS KMS trousseau de clés, vous devez utiliser un ARN de clé pour vous identifier. AWS KMS keys Pour en savoir plus, consultez [Sélection des clés d'emballage](#).
- Dans un porte-clés utilisé pour le chiffrement et le déchiffrement, vous devez utiliser un ARN de clé pour identifier les AWS KMS keys.

Lors du déchiffrement, le SDK de chiffrement de AWS base de données recherche dans le jeu de AWS KMS clés une clé capable de déchiffrer l'une AWS KMS key des clés de données chiffrées. Plus précisément, le SDK AWS de chiffrement de base de données utilise le modèle suivant pour chaque clé de données cryptée dans la description du matériel.

- Le SDK AWS de chiffrement de base de données obtient l'ARN de la clé AWS KMS key qui a chiffré la clé de données à partir des métadonnées de la description du matériel.

- Le SDK AWS Database Encryption recherche dans le trousseau de clés de déchiffrement un AWS KMS key code ARN correspondant.
- S'il trouve un AWS KMS key ARN correspondant dans le jeu de clés, le SDK de chiffrement de AWS base de données demande d'utiliser la clé KMS AWS KMS pour déchiffrer la clé de données chiffrée.
- Dans le cas contraire, il passe à la clé de données chiffrée suivante, le cas échéant.

Création d'un AWS KMS porte-clés

Vous pouvez configurer chaque AWS KMS porte-clés avec un AWS KMS key ou plusieurs AWS KMS keys éléments identiques ou différents Comptes AWS . Régions AWS AWS KMS key Il doit s'agir d'une clé de chiffrement symétrique (SYMMETRIC_DEFAULT) ou d'une clé RSA KMS asymétrique. Vous pouvez également utiliser une clé [KMS multirégionale](#) à chiffrement symétrique. Vous pouvez utiliser un ou plusieurs AWS KMS porte-clés dans un porte-clés [multiple](#).

Vous pouvez créer un AWS KMS trousseau de clés qui chiffre et déchiffre les données, ou vous pouvez créer un trousseau de AWS KMS clés spécialement conçu pour le chiffrement ou le déchiffrement. Lorsque vous créez un AWS KMS trousseau de clés pour chiffrer des données, vous devez spécifier une clé de génération, AWS KMS key qui est utilisée pour générer une clé de données en texte brut et la chiffrer. La clé de données n'est mathématiquement pas liée à la clé KMS. Ensuite, si vous le souhaitez, vous pouvez en spécifier d'autres AWS KMS keys qui chiffrent la même clé de données en texte brut. Pour déchiffrer un champ crypté protégé par ce trousseau de clés, le trousseau de déchiffrement que vous utilisez doit inclure au moins l'un des éléments AWS KMS keys définis dans le trousseau de clés, ou non. AWS KMS keys(Un AWS KMS porte-clés sans numéro AWS KMS keys est connu sous le nom de [porte-clés AWS KMS Discovery](#).)

Toutes les clés d'encapsulation d'un jeu de clés de chiffrement ou d'un jeu de clés multiples doivent être en mesure de chiffrer la clé de données. Si le chiffrement d'une clé d'encapsulation échoue, la méthode de chiffrement échoue. Par conséquent, l'appelant doit disposer des [autorisations requises](#) pour toutes les clés du trousseau de clés. Si vous utilisez un trousseau de découverte pour chiffrer des données, seul ou dans un jeu de clés multiples, l'opération de chiffrement échoue.

Les exemples suivants utilisent la `CreateAwsKmsMrkMultiKeyring` méthode pour créer un jeu de AWS KMS clés avec une clé KMS de chiffrement symétrique. La `CreateAwsKmsMrkMultiKeyring` méthode crée automatiquement le AWS KMS client et garantit que le trousseau de clés gère correctement les clés à région unique et à régions multiples. Ces

exemples utilisent une [clé ARNs](#) pour identifier les clés KMS. Pour plus d'informations, consultez [Identification AWS KMS keys dans un AWS KMS porte-clés](#).

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?;
```

Les exemples suivants utilisent la `CreateAwsKmsRsaKeyring` méthode pour créer un AWS KMS trousseau de clés avec une clé RSA KMS asymétrique. Pour créer un trousseau de AWS KMS clés RSA asymétrique, entrez les valeurs suivantes.

- `kmsClient`: créer un nouveau AWS KMS client
- `kmsKeyID`: l'ARN de clé qui identifie votre clé RSA KMS asymétrique

- `publicKey`: un ByteBuffer fichier PEM codé en UTF-8 qui représente la clé publique de la clé que vous avez transmise `kmsKeyId`
- `encryptionAlgorithm`: l'algorithme de chiffrement doit être `RSAES_OAEP_SHA_256` ou `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?;
```

Utilisation de plusieurs régions AWS KMS keys

Vous pouvez utiliser plusieurs régions AWS KMS keys comme clés d'encapsulation dans le SDK de chiffrement AWS de base de données. Si vous chiffrez avec une clé multirégionale dans une Région AWS, vous pouvez déchiffrer en utilisant une clé multirégionale associée dans une autre Région AWS.

Les clés KMS multirégionales sont un ensemble de AWS KMS keys clés différentes Régions AWS qui ont le même matériau clé et le même identifiant de clé. Vous pouvez utiliser ces clés associées comme s'il s'agissait de la même clé dans différentes régions. Les clés multirégionales prennent en charge les scénarios courants de reprise après sinistre et de sauvegarde qui nécessitent le chiffrement dans une région et le déchiffrement dans une autre région sans passer un appel interrégional à AWS KMS. Pour plus d'informations sur les clés multirégionales, consultez la section [Utilisation des clés multirégionales](#) dans le manuel du AWS Key Management Service développeur.

Pour prendre en charge les clés multirégionales, le SDK de chiffrement de AWS base de données inclut AWS KMS multi-Region-aware des trousseaux de clés. La `CreateAwsKmsMrkMultiKeyring` méthode prend en charge les clés à région unique et à régions multiples.

- Pour les clés à région unique, le multi-Region-aware symbole se comporte comme le porte-clés à région unique AWS KMS. Il tente de déchiffrer le texte chiffré uniquement à l'aide de la clé à région unique qui a chiffré les données. Pour simplifier votre expérience en matière de trousseau de AWS KMS clés, nous vous recommandons d'utiliser `CreateAwsKmsMrkMultiKeyring` cette méthode chaque fois que vous utilisez une clé KMS de chiffrement symétrique.
- Pour les clés multirégionales, le multi-Region-aware symbole tente de déchiffrer le texte chiffré avec la même clé multirégionale qui a chiffré les données ou avec la clé multirégionale associée dans la région que vous spécifiez.

Dans les multi-Region-aware trousseaux de clés qui utilisent plusieurs clés KMS, vous pouvez spécifier plusieurs clés à région unique ou multirégionale. Toutefois, vous ne pouvez spécifier qu'une

seule clé pour chaque ensemble de clés multirégionales associées. Si vous spécifiez plusieurs identificateurs de clé avec le même identifiant de clé, l'appel du constructeur échoue.

Les exemples suivants créent un AWS KMS trousseau de clés avec une clé KMS multirégionale. Les exemples spécifient une clé multirégionale comme clé génératrice et une clé mono-région comme clé enfant.

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?;
```

Lorsque vous utilisez des trousseaux de AWS KMS clés multirégionaux, vous pouvez déchiffrer le texte chiffré en mode strict ou en mode découverte. Pour déchiffrer le texte chiffré en mode strict, instanciez le symbole multi-Region-aware avec la clé ARN de la clé multirégionale associée dans la région dans laquelle vous êtes en train de déchiffrer le texte chiffré. Si vous spécifiez l'ARN d'une clé multirégionale associée dans une région différente (par exemple, la région où l'enregistrement a été chiffré), le multi-Region-aware symbole émettra un appel interrégional pour cela. AWS KMS key

Lors du déchiffrement en mode strict, le multi-Region-aware symbole nécessite un ARN clé. Il n'accepte qu'un seul ARN de clé pour chaque ensemble de clés multirégionales associées.

Vous pouvez également déchiffrer en mode découverte à l'aide de clés AWS KMS multirégionales. Lorsque vous déchiffrez en mode découverte, vous n'en spécifiez aucune. AWS KMS keys (Pour plus d'informations sur les porte-clés de AWS KMS découverte d'une seule région, consultez [Utilisation d'un porte-clés AWS KMS Discovery](#).)

Si vous avez chiffré avec une clé multirégionale, le multi-Region-aware symbole en mode découverte essaiera de le déchiffrer en utilisant une clé multirégionale associée dans la région locale. S'il n'en existe aucun, l'appel échoue. En mode découverte, le SDK AWS de chiffrement de base de données ne tente pas d'appeler plusieurs régions pour obtenir la clé multirégionale utilisée pour le chiffrement.

Utilisation d'un porte-clés AWS KMS Discovery

Lors du déchiffrement, il est recommandé de spécifier les clés d'encapsulation que le SDK de chiffrement de AWS base de données peut utiliser. Pour suivre cette bonne pratique, utilisez un jeu de clés de AWS KMS déchiffrement qui limite les clés AWS KMS d'encapsulation à celles que vous spécifiez. Toutefois, vous pouvez également créer un trousseau de clés de AWS KMS découverte, c'est-à-dire un AWS KMS trousseau de clés ne spécifiant aucune clé d'encapsulation.

Le SDK AWS Database Encryption fournit un jeu de clés de AWS KMS découverte standard et un jeu de clés de découverte pour les AWS KMS clés multirégionales. Pour plus d'informations sur l'utilisation de clés multirégionales avec le SDK AWS de chiffrement de base de données, consultez [Utilisation de plusieurs régions AWS KMS keys](#)

Comme il ne spécifie aucune clé d'encapsulation, un jeu de clés de découverte ne peut pas chiffrer les données. Si vous utilisez un trousseau de découverte pour chiffrer des données, seul ou dans un jeu de clés multiples, l'opération de chiffrement échoue.

Lors du déchiffrement, un jeu de clés de découverte permet au SDK de chiffrement de AWS base de données de demander AWS KMS à déchiffrer toute clé de données chiffrée à l'aide de AWS

KMS key celle qui l'a chiffrée, indépendamment de son propriétaire ou de son accès. AWS KMS key L'appel ne réussit que lorsque l'appelant est kms :Decrypt autorisé à utiliser le. AWS KMS key

Important

Si vous incluez un jeu de clés de AWS KMS découverte dans un jeu de clés [multiples de déchiffrement, le jeu de clés](#) de découverte remplace toutes les restrictions relatives aux clés KMS spécifiées par les autres trousseaux de clés du jeu de clés multiples. Le porte-clés multiple se comporte comme le porte-clés le moins restrictif. Si vous utilisez un trousseau de découverte pour chiffrer des données, seul ou dans un jeu de clés multiples, l'opération de chiffrement échoue

Le SDK AWS Database Encryption fournit un jeu de clés de AWS KMS découverte pour plus de commodité. Cependant, nous vous recommandons d'utiliser un porte-clés plus limité chaque fois que possible pour les raisons suivantes.

- **Authenticité** — Un trousseau de clés de AWS KMS découverte peut utiliser tout AWS KMS key élément utilisé pour chiffrer une clé de données dans la description du matériel, à condition que l'appelant soit autorisé à l'utiliser pour le déchiffrer. AWS KMS key Ce n'est peut-être pas AWS KMS key celui que l'appelant a l'intention d'utiliser. Par exemple, l'une des clés de données cryptées peut avoir été cryptée sous une forme moins sécurisée AWS KMS key que tout le monde peut utiliser.
- **Latence et performances** : un jeu de clés de AWS KMS découverte peut être sensiblement plus lent que les autres car le SDK de chiffrement de AWS base de données essaie de déchiffrer toutes les clés de données chiffrées, y compris celles chiffrées AWS KMS keys dans d'autres régions, Comptes AWS et AWS KMS keys que l'appelant n'est pas autorisé à utiliser pour le déchiffrement.

Si vous utilisez un trousseau de clés de découverte, nous vous recommandons d'utiliser un [filtre de découverte](#) pour limiter les clés KMS pouvant être utilisées à celles Comptes AWS des [partitions](#) spécifiées. Pour obtenir de l'aide pour trouver votre identifiant de compte et votre partition, consultez la section [Vos Compte AWS identifiants](#) et votre [format ARN](#) dans le Références générales AWS.

Les exemples de code suivants instancient un jeu de clés de AWS KMS découverte avec un filtre de découverte qui limite les clés KMS que le SDK de chiffrement de AWS base de données peut utiliser à celles de la aws partition et du compte d'exemple. 111122223333

Avant d'utiliser ce code, remplacez les valeurs d'exemple Compte AWS et de partition par des valeurs valides pour votre partition Compte AWS and. Si vos clés KMS se trouvent dans les régions de Chine, utilisez la valeur de `aws-cn` partition. Si vos clés KMS sont incluses AWS GovCloud (US) Regions, utilisez la valeur de `aws-us-gov` partition. Pour tous les autres Régions AWS, utilisez la valeur de `aws` partition.

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

Utiliser un porte-clés de découverte AWS KMS régional

Un trousseau de découverte AWS KMS régional est un trousseau de clés qui ne précise pas les ARNs clés KMS. Au lieu de cela, il permet au SDK AWS de chiffrement de base de données de déchiffrer en utilisant uniquement les clés KMS en particulier. Régions AWS

Lors du déchiffrement à l'aide d'un jeu de clés de découverte AWS KMS régional, le SDK AWS de chiffrement de base de données déchiffre toute clé de données chiffrée selon un dans le spécifié. AWS KMS key Région AWS Pour réussir, l'appelant doit avoir l'`kms:Decrypt` autorisation d'utiliser au moins l'un des éléments spécifiés Région AWS qui ont chiffré une clé de données. AWS KMS keys

Comme les autres trousseaux de découverte, le trousseau de clés de découverte régional n'a aucun effet sur le chiffrement. Cela ne fonctionne que lors du déchiffrement de champs chiffrés. Si vous utilisez un jeu de clés de découverte régional dans un jeu de clés multiples utilisé pour le chiffrement et le déchiffrement, il n'est efficace que lors du déchiffrement. Si vous utilisez un jeu de clés de découverte multirégional pour chiffrer des données, seul ou dans un jeu de clés multirégional, l'opération de chiffrement échoue.

Important

Si vous incluez un trousseau de clés de découverte AWS KMS régional dans un jeu de clés [multiples de déchiffrement, le jeu de clés](#) de découverte régional remplace toutes les restrictions relatives aux clés KMS spécifiées par les autres trousseaux de clés du jeu de clés multiples. Le porte-clés multiple se comporte comme le porte-clés le moins restrictif. Un trousseau de AWS KMS découverte n'a aucun effet sur le chiffrement lorsqu'il est utilisé seul ou dans un jeu de clés multiples.

Le jeu de clés de découverte régional du SDK AWS de chiffrement de base de données tente de déchiffrer uniquement avec des clés KMS dans la région spécifiée. Lorsque vous utilisez un trousseau de clés de découverte, vous configurez la région sur le AWS KMS client. Ces implémentations du SDK de chiffrement de AWS base de données ne filtrent pas les clés KMS par région, mais AWS KMS échouera à une demande de déchiffrement de clés KMS en dehors de la région spécifiée.

Si vous utilisez un trousseau de clés de découverte, nous vous recommandons d'utiliser un filtre de découverte afin de limiter les clés KMS utilisées pour le déchiffrement à celles figurant dans les partitions Comptes AWS et les partitions spécifiées.

Par exemple, le code suivant crée un trousseau de clés de découverte AWS KMS régional avec un filtre de découverte. Ce jeu de clés limite le SDK AWS de chiffrement de base de données aux clés KMS du compte 111122223333 dans la région USA Ouest (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 Porte-clés hiérarchiques

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Note

Depuis le 24 juillet 2023, les clés de branche créées lors de la version préliminaire pour les développeurs ne sont plus prises en charge. Créez de nouvelles clés de branche pour continuer à utiliser le magasin de clés que vous avez créé lors de la version préliminaire pour les développeurs.

Avec le trousseau de clés AWS KMS hiérarchique, vous pouvez protéger vos documents cryptographiques sous une clé KMS de chiffrement symétrique sans avoir à appeler AWS KMS chaque fois que vous chiffrez ou déchiffrez un enregistrement. Il s'agit d'un bon choix pour les applications qui doivent minimiser les appels AWS KMS, ainsi que pour les applications qui peuvent réutiliser certains matériels cryptographiques sans enfreindre leurs exigences de sécurité.

Le trousseau de clés hiérarchique est une solution de mise en cache des matériaux cryptographiques qui réduit le nombre d' AWS KMS appels en utilisant des clés de branche AWS KMS protégées conservées dans une table Amazon DynamoDB, puis en mettant en cache localement les éléments clés de branche utilisés dans les opérations de chiffrement et de déchiffrement. La table DynamoDB sert de magasin de clés qui gère et protège les clés de branche. Il stocke la clé de branche active et toutes les versions précédentes de la clé de branche. La clé de branche active est la version de clé de branche la plus récente. Le trousseau de clés hiérarchique utilise une clé de chiffrement de données unique pour chaque demande de chiffrement et chiffre chaque clé de chiffrement de données avec une clé d'encapsulation unique dérivée de la clé de branche active. Le trousseau de clés hiérarchique dépend de la hiérarchie établie entre les clés de branche actives et leurs clés d'encapsulation dérivées.

Le trousseau de clés hiérarchique utilise généralement chaque version de clé de branche pour satisfaire plusieurs demandes. Mais vous contrôlez la mesure dans laquelle les clés de branche actives sont réutilisées et vous déterminez la fréquence à laquelle la clé de branche active est pivotée. La version active de la clé de branche reste active jusqu'à ce que vous la [fassiez pivoter](#). Les versions précédentes de la clé de branche active ne seront pas utilisées pour effectuer des opérations de chiffrement, mais elles peuvent toujours être interrogées et utilisées dans des opérations de déchiffrement.

Lorsque vous instanciez le trousseau de clés hiérarchique, il crée un cache local. Vous spécifiez une [limite de cache](#) qui définit la durée maximale pendant laquelle les éléments clés de branche sont stockés dans le cache local avant leur expiration et leur expulsion du cache. Le trousseau de clés hiérarchique effectue un AWS KMS appel pour déchiffrer la clé de branche et assembler les matériaux de la clé de branche la première fois que `branch-key-id` est spécifié dans une opération. Les éléments clés de branche sont ensuite stockés dans le cache local et réutilisés pour toutes les opérations de chiffrement et de déchiffrement qui le spécifient `branch-key-id` jusqu'à l'expiration de la limite de cache. Le stockage des éléments clés de branche dans le cache local réduit le nombre d' AWS KMS appels. Par exemple, considérez une limite de cache de 15 minutes. Si vous effectuez 10 000 opérations de chiffrement dans cette limite de cache, le trousseau de [AWS KMS clés traditionnel](#) devra effectuer 10 000 AWS KMS appels pour satisfaire 10 000 opérations de

chiffrement. Si vous en avez un actif `branch-key-id`, le trousseau de clés hiérarchique n'a besoin que d'un seul AWS KMS appel pour satisfaire 10 000 opérations de chiffrement.

Le cache local sépare le matériel de chiffrement du matériel de déchiffrement. Les matériaux de chiffrement sont assemblés à partir de la clé de branche active et réutilisés pour toutes les opérations de chiffrement jusqu'à l'expiration de la limite de cache. Les matériaux de déchiffrement sont assemblés à partir de l'ID et de la version de la clé de branche identifiés dans les métadonnées du champ crypté, et ils sont réutilisés pour toutes les opérations de déchiffrement liées à l'ID et à la version de la clé de branche jusqu'à l'expiration de la limite de cache. Le cache local peut stocker plusieurs versions de la même clé de branche à la fois. Lorsque le cache local est configuré pour utiliser un [branch key ID supplier](#), il peut également stocker des éléments clés de branche provenant de plusieurs clés de branche actives à la fois.

Note

Toutes les mentions du jeu de clés hiérarchique dans le SDK de chiffrement AWS de base de données font référence au jeu de clés AWS KMS hiérarchique.

Rubriques

- [Comment ça marche](#)
- [Conditions préalables](#)
- [Autorisations requises](#)
- [Choisissez un cache](#)
- [Création d'un trousseau de clés hiérarchique](#)
- [Utilisation du trousseau de clés hiérarchique pour un chiffrement consultable](#)

Comment ça marche

Les procédures pas à pas suivantes décrivent comment le trousseau de clés hiérarchique assemble le matériel de chiffrement et de déchiffrement, ainsi que les différents appels effectués par le trousseau de clés pour les opérations de chiffrement et de déchiffrement. Pour plus de détails techniques sur les processus de dérivation des clés d'encapsulation et de chiffrement des clés de données en texte clair, consultez la section Détails techniques du trousseau de [clés AWS KMS hiérarchique](#).

Chiffrer et signer

La procédure pas à pas suivante décrit comment le trousseau de clés hiérarchique assemble les matériaux de chiffrement et en déduit une clé d'encapsulation unique.

1. La méthode de cryptage demande au trousseau de clés hiérarchique le matériel de cryptage. Le trousseau de clés génère une clé de données en texte brut, puis vérifie s'il existe des éléments de clé de branche valides dans le cache local pour générer la clé d'encapsulation. S'il existe des documents relatifs aux clés de succursale valides, le porte-clés passe à l'étape 4.
2. S'il n'existe aucun matériel de clé de branche valide, le trousseau de clés hiérarchique interroge le magasin de clés pour trouver la clé de branche active.
 - a. Le magasin de clés appelle AWS KMS pour déchiffrer la clé de branche active et renvoie la clé de branche active en texte clair. Les données identifiant la clé de branche active sont sérialisées pour fournir des données authentifiées supplémentaires (AAD) lors de l'appel de déchiffrement à AWS KMS
 - b. Le magasin de clés renvoie la clé de branche en texte brut et les données qui l'identifient, telles que la version de la clé de branche.
3. Le trousseau de clés hiérarchique assemble les éléments clés de branche (version de clé de branche en texte clair et de clé de branche) et en stocke une copie dans le cache local.
4. Le trousseau de clés hiérarchique déduit une clé d'encapsulation unique à partir de la clé de branche en texte brut et d'un sel aléatoire de 16 octets. Il utilise la clé d'encapsulation dérivée pour chiffrer une copie de la clé de données en texte brut.

La méthode de cryptage utilise le matériel de cryptage pour chiffrer et signer l'enregistrement. Pour plus d'informations sur la façon dont les enregistrements sont chiffrés et signés dans le SDK AWS de chiffrement de base de données, voir [Chiffrer et signer](#).

Déchiffrer et vérifier

La procédure pas à pas suivante décrit comment le trousseau de clés hiérarchique assemble le matériel de déchiffrement et déchiffre la clé de données chiffrée.

1. Le procédé de déchiffrement identifie la clé de données chiffrée dans le champ de description matérielle de l'enregistrement chiffré et la transmet au trousseau de clés hiérarchique.
2. Le trousseau hiérarchique déserialise les données identifiant la clé de données chiffrée, y compris la version de la clé de branche, le sel de 16 octets et d'autres informations décrivant la manière dont la clé de données a été cryptée.

Pour de plus amples informations, veuillez consulter [AWS KMS Détails techniques du porte-clés hiérarchique](#).

3. Le trousseau de clés hiérarchique vérifie si le cache local contient des éléments de clé de branche valides qui correspondent à la version de clé de branche identifiée à l'étape 2. S'il existe des documents relatifs aux clés de succursale valides, le porte-clés passe à l'étape 6.
4. S'il n'existe aucun matériel de clé de branche valide, le trousseau de clés hiérarchique interroge le magasin de clés pour trouver la clé de branche correspondant à la version de clé de branche identifiée à l'étape 2.
 - a. Le magasin de clés appelle AWS KMS pour déchiffrer la clé de branche et renvoie la clé de branche active en texte clair. Les données identifiant la clé de branche active sont sérialisées pour fournir des données authentifiées supplémentaires (AAD) lors de l'appel de déchiffrement à AWS KMS
 - b. Le magasin de clés renvoie la clé de branche en texte brut et les données qui l'identifient, telles que la version de la clé de branche.
5. Le trousseau de clés hiérarchique assemble les éléments clés de branche (version de clé de branche en texte clair et de clé de branche) et en stocke une copie dans le cache local.
6. Le trousseau de clés hiérarchique utilise les éléments de clé de branche assemblés et le sel de 16 octets identifié à l'étape 2 pour reproduire la clé d'encapsulation unique qui a chiffré la clé de données.
7. Le trousseau de clés hiérarchique utilise la clé d'encapsulation reproduite pour déchiffrer la clé de données et renvoie la clé de données en texte brut.

La méthode de déchiffrement utilise le matériel de déchiffrement et la clé de données en texte brut pour déchiffrer et vérifier l'enregistrement. [Pour plus d'informations sur la façon dont les enregistrements sont déchiffrés et vérifiés dans le SDK de chiffrement de AWS base de données, voir Déchiffrer et vérifier.](#)

Conditions préalables

Avant de créer et d'utiliser un trousseau de clés hiérarchique, assurez-vous que les conditions préalables suivantes sont remplies.

- Vous, ou l'administrateur de votre magasin de clés, avez [créé un magasin de clés](#) et [créé au moins une clé de branche active](#).

- Vous avez [configuré les actions de votre magasin de clés](#).

Note

La façon dont vous configurez les actions de votre magasin de clés détermine les opérations que vous pouvez effectuer et les clés KMS que le trousseau de clés hiérarchique peut utiliser. Pour plus d'informations, consultez la section [Actions du magasin de clés](#).

- Vous disposez des AWS KMS autorisations requises pour accéder aux clés du magasin de clés et des succursales et les utiliser. Pour de plus amples informations, veuillez consulter [the section called “Autorisations requises”](#).
- Vous avez examiné les types de cache pris en charge et configuré le type de cache le mieux adapté à vos besoins. Pour de plus amples informations, consultez [the section called “Choisissez un cache”](#).

Autorisations requises

Le SDK AWS de chiffrement de base de données n'en nécessite pas Compte AWS et n'en dépend pas Service AWS. Toutefois, pour utiliser un trousseau de clés hiérarchique, vous devez disposer Compte AWS des autorisations minimales suivantes sur le ou les AWS KMS key chiffrements symétriques de votre magasin de clés.

- [Pour chiffrer et déchiffrer des données à l'aide du trousseau de clés hiérarchique, vous avez besoin de KMS:Decrypt](#).
- Pour [créer](#) et [faire pivoter](#) des clés de branche, vous avez besoin de [kms : GenerateDataKeyWithoutPlaintext](#) et [kms : ReEncrypt](#).

Pour plus d'informations sur le contrôle de l'accès à vos clés de succursale et à votre magasin de clés, consultez [the section called “Implémentation des autorisations avec le moindre privilégié”](#).

Choisissez un cache

Le trousseau de clés hiérarchique réduit le nombre d'appels en mettant AWS KMS en cache localement les éléments clés de branche utilisés dans les opérations de chiffrement et de déchiffrement. Avant de [créer votre trousseau de clés hiérarchique](#), vous devez choisir le type de

cache que vous souhaitez utiliser. Vous pouvez utiliser le cache par défaut ou le personnaliser en fonction de vos besoins.

Le trousseau de clés hiérarchique prend en charge les types de cache suivants :

- [the section called “Cache par défaut”](#)
- [the section called “MultiThreaded cache”](#)
- [the section called “StormTracking cache”](#)
- [the section called “Cache partagé”](#)

Cache par défaut

Pour la plupart des utilisateurs, le cache par défaut répond à leurs exigences en matière de threading. Le cache par défaut est conçu pour prendre en charge les environnements fortement multithread. Lorsqu'une entrée de matériel de clé de branche expire, le cache par défaut empêche plusieurs threads d'appeler AWS KMS en notifiant à un thread que l'entrée de matériaux de clé de branche va expirer 10 secondes à l'avance. Cela garantit qu'un seul thread envoie une demande AWS KMS pour actualiser le cache.

La valeur par défaut et StormTracking les caches prennent en charge le même modèle de thread, mais il suffit de spécifier la capacité d'entrée pour utiliser le cache par défaut. Pour des personnalisations de cache plus détaillées, utilisez le [the section called “StormTracking cache”](#)

À moins que vous ne souhaitiez personnaliser le nombre d'entrées de matériaux clés de branche pouvant être stockées dans le cache local, il n'est pas nécessaire de spécifier un type de cache lorsque vous créez le trousseau de clés hiérarchique. Si vous ne spécifiez aucun type de cache, le trousseau de clés hiérarchique utilise le type de cache par défaut et définit la capacité d'entrée à 1 000.

Pour personnaliser le cache par défaut, spécifiez les valeurs suivantes :

- Capacité d'entrée : limite le nombre d'entrées de matériaux clés de branche qui peuvent être stockées dans le cache local.

Java

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

```
.build())
```

C# / .NET

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

Rust

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

MultiThreaded cache

Le MultiThreaded cache peut être utilisé en toute sécurité dans les environnements multithread, mais il ne fournit aucune fonctionnalité permettant de minimiser les appels Amazon AWS KMS DynamoDB. Par conséquent, lorsqu'une entrée de contenu clé de branche expire, tous les fils de discussion seront avertis en même temps. Cela peut entraîner plusieurs AWS KMS appels pour actualiser le cache.

Pour utiliser le MultiThreaded cache, spécifiez les valeurs suivantes :

- Capacité d'entrée : limite le nombre d'entrées de matériaux clés de branche qui peuvent être stockées dans le cache local.
- Taille de la queue d'élagage d'entrée : définit le nombre d'entrées à élaguer si la capacité d'entrée est atteinte.

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 cache

Le StormTracking cache est conçu pour prendre en charge les environnements fortement multithread. Lorsqu'une entrée de matériel de clé de branche expire, le StormTracking cache empêche plusieurs threads d'appeler AWS KMS en notifiant à l'un d'entre eux que l'entrée de matériaux de clé de branche va expirer à l'avance. Cela garantit qu'un seul thread envoie une demande AWS KMS pour actualiser le cache.

Pour utiliser le StormTracking cache, spécifiez les valeurs suivantes :

- Capacité d'entrée : limite le nombre d'entrées de matériaux clés de branche qui peuvent être stockées dans le cache local.

Valeur par défaut : 1000 entrées

- Taille de la queue d'élagage d'entrée : définit le nombre d'entrées de matériaux clés de branche à tailler à la fois.

Valeur par défaut : 1 entrée

- Période de grâce : définit le nombre de secondes avant l'expiration pendant lesquelles une tentative d'actualisation des documents clés de la branche est effectuée.

Valeur par défaut : 10 secondes

- Intervalle de grâce : définit le nombre de secondes entre les tentatives d'actualisation des éléments clés de la branche.

Valeur par défaut : 1 seconde

- Ventilateur : définit le nombre de tentatives simultanées qui peuvent être effectuées pour actualiser les documents clés de la branche.

Valeur par défaut : 20 tentatives

- In flight time to live (TTL) : définit le nombre de secondes avant l'expiration d'une tentative d'actualisation des informations clés de branche. Chaque fois que le cache revient `NoSuchEntry` en réponse à `unGetCacheEntry`, cette clé de branche est considérée comme étant en vol jusqu'à ce que la même clé soit écrite avec une `PutCache` entrée.

Valeur par défaut : 10 secondes

- Sommeil : définit le nombre de secondes pendant lesquelles un thread doit être mis en veille si le `fanOut` délai est dépassé.

Valeur par défaut : 20 millisecondes

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 partagé

Par défaut, le trousseau de clés hiérarchique crée un nouveau cache local chaque fois que vous instanciez le trousseau de clés. Cependant, le cache partagé peut contribuer à économiser de la mémoire en vous permettant de partager un cache entre plusieurs trousseaux de clés hiérarchiques. Plutôt que de créer un nouveau cache de matériaux cryptographiques pour chaque jeu de clés hiérarchique que vous instanciez, le cache partagé ne stocke qu'un seul cache en mémoire, qui peut être utilisé par tous les jeux de clés hiérarchiques qui le référencent. Le cache partagé permet d'optimiser l'utilisation de la mémoire en évitant la duplication du matériel cryptographique entre les trousseaux de clés. Au lieu de cela, les trousseaux de clés hiérarchiques peuvent accéder au même cache sous-jacent, réduisant ainsi l'encombrement mémoire global.

Lorsque vous créez votre cache partagé, vous définissez toujours le type de cache. Vous pouvez spécifier un [the section called “Cache par défaut”](#), [the section called “MultiThreaded cache”](#), ou [the section called “StormTracking cache”](#) comme type de cache, ou le remplacer par un cache personnalisé compatible.

Partitions

Plusieurs trousseaux de clés hiérarchiques peuvent utiliser un seul cache partagé. Lorsque vous créez un trousseau de clés hiérarchique avec un cache partagé, vous pouvez définir un ID de partition facultatif. L'ID de partition permet de distinguer le jeu de clés hiérarchique qui écrit dans le cache. Si deux trousseaux de clés hiérarchiques font référence au même ID de partition et au même ID de clé de branche [logical key store name](#), les deux trousseaux de clés partageront les mêmes entrées de cache dans le cache. Si vous créez deux trousseaux de clés hiérarchiques avec le même cache partagé, mais avec une partition différente IDs, chaque trousseau de clés n'accèdera aux entrées du cache qu'à partir de sa propre partition désignée dans le cache partagé. Les partitions agissent comme des divisions logiques au sein du cache partagé, permettant à chaque jeu de clés hiérarchique de fonctionner indépendamment sur sa propre partition désignée, sans interférer avec les données stockées dans l'autre partition.

Si vous avez l'intention de réutiliser ou de partager les entrées du cache d'une partition, vous devez définir votre propre ID de partition. Lorsque vous transmettez l'ID de partition à votre trousseau de clés hiérarchique, celui-ci peut réutiliser les entrées du cache déjà présentes dans le cache partagé, sans avoir à récupérer et à réautoriser les éléments clés de branche. Si vous ne spécifiez pas d'ID de partition, un ID de partition unique est automatiquement attribué au trousseau de clés chaque fois que vous instanciez le trousseau de clés hiérarchique.

Les procédures suivantes montrent comment créer un cache partagé avec le [type de cache par défaut](#) et le transmettre à un trousseau de clés hiérarchique.

1. Créez un `CryptographicMaterialsCache` (CMC) à l'aide de la [bibliothèque des fournisseurs de matériaux](#) (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. Créez un CacheType objet pour le cache partagé.

Transmettez le `sharedCryptographicMaterialsCache` que vous avez créé à l'étape 1 au nouvel `CacheType` objet.

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. Passez l'`sharedCache` objet de l'étape 2 à votre trousseau de clés hiérarchique.

Lorsque vous créez un trousseau de clés hiérarchique avec un cache partagé, vous pouvez éventuellement définir un `partitionID` pour partager les entrées du cache entre plusieurs trousseaux hiérarchiques. Si vous ne spécifiez pas d'ID de partition, le trousseau de clés hiérarchique attribue automatiquement un identifiant de partition unique au trousseau de clés.

Note

Vos trousseaux de clés hiérarchiques partageront les mêmes entrées de cache dans un cache partagé si vous créez au moins deux trousseaux de clés faisant référence au même ID de partition et au même ID de clé de branche. [logical key store name](#) Si vous ne souhaitez pas que plusieurs trousseaux de clés partagent les mêmes entrées

de cache, vous devez utiliser un identifiant de partition unique pour chaque jeu de clés hiérarchique.

L'exemple suivant crée un jeu de clés hiérarchique avec une [branch key ID fournisseur limite de cache](#) de 600 secondes. Pour plus d'informations sur les valeurs définies dans la configuration hiérarchique du trousseau de clés suivante, consultez [the section called "Création d'un trousseau de clés hiérarchique"](#).

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?;
```

Création d'un trousseau de clés hiérarchique

Pour créer un trousseau de clés hiérarchique, vous devez fournir les valeurs suivantes :

- Le nom d'un magasin clé

Le nom de la table DynamoDB que vous, ou votre administrateur de magasin de clés, avez créée pour servir de magasin de clés.

-

Une limite de durée de vie du cache (TTL)

Durée en secondes pendant laquelle une entrée de clé de branche dans le cache local peut être utilisée avant son expiration. La limite de cache TTL détermine la fréquence à laquelle le client appelle AWS KMS pour autoriser l'utilisation des clés de branche. Cette valeur doit être supérieure à zéro. Une fois la limite de cache TTL expirée, l'entrée n'est jamais servie et sera expulsée du cache local.

- Un identifiant de clé de branche

Vous pouvez soit configurer statiquement le `branch-key-id` identifiant d'une seule clé de branche active dans votre magasin de clés, soit fournir un fournisseur d'ID de clé de branche.

Le fournisseur d'ID de clé de branche utilise les champs stockés dans le contexte de chiffrement pour déterminer quelle clé de branche est requise pour déchiffrer un enregistrement. Par défaut,

seules les clés de partition et de tri sont incluses dans le contexte de chiffrement. Toutefois, vous pouvez utiliser l'[action SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT cryptographique](#) pour inclure des champs supplémentaires dans le contexte de chiffrement.

Nous vous recommandons vivement de faire appel à un fournisseur d'ID de clé de branche pour les bases de données mutualisées où chaque locataire possède sa propre clé de branche. Vous pouvez utiliser le fournisseur d'ID de clé de succursale pour créer un nom convivial pour votre clé de succursale IDs afin de reconnaître facilement l'ID de clé de branche correct pour un locataire spécifique. Par exemple, le nom convivial vous permet de faire référence à une clé de branche au tenant1 lieu deb3f61619-4d35-48ad-a275-050f87e15122.

Pour les opérations de déchiffrement, vous pouvez soit configurer de manière statique un jeu de clés hiérarchique unique pour limiter le déchiffrement à un seul locataire, soit utiliser le fournisseur d'ID de clé de branche pour identifier le locataire responsable du déchiffrement d'un enregistrement.

- (Facultatif) Un cache

Si vous souhaitez personnaliser le type de cache ou le nombre d'entrées clés de branche pouvant être stockées dans le cache local, spécifiez le type de cache et la capacité d'entrée lorsque vous initialisez le trousseau de clés.

Le trousseau de clés hiérarchique prend en charge les types de cache suivants : par défaut MultiThreaded, StormTracking, et partagé. Pour plus d'informations et des exemples illustrant comment définir chaque type de cache, consultez [the section called “Choisissez un cache”](#).

Si vous ne spécifiez pas de cache, le trousseau de clés hiérarchique utilise automatiquement le type de cache par défaut et définit la capacité d'entrée à 1 000.

- (Facultatif) Un identifiant de partition

Si vous spécifiez le [the section called “Cache partagé”](#), vous pouvez éventuellement définir un ID de partition. L'ID de partition permet de distinguer le jeu de clés hiérarchique qui écrit dans le cache. Si vous avez l'intention de réutiliser ou de partager les entrées du cache d'une partition, vous devez définir votre propre ID de partition. Vous pouvez spécifier n'importe quelle chaîne pour l'ID de partition. Si vous ne spécifiez pas d'ID de partition, un ID de partition unique est automatiquement attribué au trousseau de clés lors de sa création.

Pour de plus amples informations, veuillez consulter [Partitions](#).

Note

Vos trousseaux de clés hiérarchiques partageront les mêmes entrées de cache dans un cache partagé si vous créez au moins deux trousseaux de clés faisant référence au même ID de partition et au même ID de clé de branche. [logical key store name](#) Si vous ne souhaitez pas que plusieurs trousseaux de clés partagent les mêmes entrées de cache, vous devez utiliser un identifiant de partition unique pour chaque jeu de clés hiérarchique.

- (Facultatif) Une liste de jetons de subvention

Si vous contrôlez l'accès à la clé KMS dans votre trousseau de clés hiérarchique avec [des autorisations](#), vous devez fournir tous les jetons de subvention nécessaires lorsque vous initialisez le trousseau de clés.

Création d'un trousseau de clés hiérarchique avec un identifiant de clé de branche statique

Les exemples suivants montrent comment créer un jeu de clés hiérarchique avec un identifiant de clé de branche statique, [lethe section called “Cache par défaut”](#), et une limite de cache TTL de 600 secondes.

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?;

```

Créez un trousseau de clés hiérarchique avec un fournisseur d'identifiant de clé de succursale

Les procédures suivantes montrent comment créer un trousseau de clés hiérarchique avec un fournisseur d'ID de clé de branche.

1. Création d'un fournisseur d'ID de clé de succursale

L'exemple suivant crée des noms conviviaux pour les deux clés de branche créées à l'étape 1 et appelle `CreateDynamoDbEncryptionBranchKeyIdSupplier` à créer un fournisseur d'ID de clé de branche avec le SDK AWS Database Encryption pour le client 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. Création d'un trousseau de clés hiérarchique

Les exemples suivants initialisent un jeu de clés hiérarchique avec le fournisseur d'ID de clé de branche créé à l'étape 1, une limite de cache TLL de 600 secondes et une taille de cache maximale de 1 000.

Java

```

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

```

C# / .NET

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

Rust

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

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

Utilisation du trousseau de clés hiérarchique pour un chiffrement consultable

[Le chiffrement consultable](#) vous permet de rechercher des enregistrements cryptés sans déchiffrer l'intégralité de la base de données. [Pour ce faire, il faut indexer la valeur en texte brut d'un champ chiffré à l'aide d'une balise.](#) Pour implémenter le chiffrement consultable, vous devez utiliser un trousseau de clés hiérarchique.

L'opération `CreateKey` de stockage des clés génère à la fois une clé de branche et une clé de balise. La clé de branche est utilisée dans les opérations de chiffrement et de déchiffrement des enregistrements. La clé de balise est utilisée pour générer des balises.

La clé de branche et la clé de balise sont protégées par la même protection AWS KMS key que celle que vous spécifiez lors de la création de votre service de banque de clés. Une fois que l'`CreateKey` opération appelle AWS KMS pour générer la clé de branche, elle appelle [kms : GenerateDataKeyWithoutPlaintext](#) une seconde fois pour générer la clé de balise en utilisant la requête suivante.

```
{
  "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"
}
```

Après avoir généré les deux clés, l'`CreateKey` opération appelle [ddb : TransactWriteItems](#) pour écrire deux nouveaux éléments qui conserveront la clé de branche et la clé de balise dans votre magasin de clés de succursale.

Lorsque vous [configurez une balise standard](#), le SDK AWS de chiffrement de base de données interroge le magasin de clés pour obtenir la clé de balise. Il utilise ensuite une fonction de dérivation de `extract-and-expand` clé basée sur HMAC ([HKDF](#)) pour combiner la clé de balise avec le nom de la [balise standard](#) afin de créer la clé HMAC pour une balise donnée.

Contrairement aux clés de branche, il n'existe qu'une seule version de clé de balise par clé `branch-key-id` dans un magasin de clés. La clé de la balise n'est jamais tournée.

Définition de la source clé de votre balise

Lorsque vous définissez la [version de balise](#) pour vos balises standard et composées, vous devez identifier la clé de balise et définir une durée de vie limite de cache (TTL) pour les éléments clés de la balise. Les éléments clés des balises sont stockés dans un cache local distinct de celui des clés de branche. L'extrait suivant montre comment définir le pour une base de données à `keySource` locale unique. Identifiez la clé de votre balise par celle à laquelle `branch-key-id` elle est associée.

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()?,
    ))
```

Définition de la source de balise dans une base de données mutualisée

Si vous disposez d'une base de données mutualisée, vous devez spécifier les valeurs suivantes lors de la keySource configuration de.

-

keyFieldName

Définit le nom du champ qui stocke la clé `branch-key-id` associée à la balise utilisée pour générer des balises pour un locataire donné. Il `keyFieldName` peut s'agir de n'importe quelle chaîne, mais elle doit être unique à tous les autres champs de votre base de données. Lorsque vous écrivez de nouveaux enregistrements dans votre base de données, la clé `branch-key-id` identifiant la clé de balise utilisée pour générer des balises pour cet enregistrement est stockée dans ce champ. Vous devez inclure ce champ dans vos requêtes sur les balises et identifier les éléments clés de balise appropriés nécessaires pour recalculer la balise. Pour de plus amples informations, veuillez consulter [Interrogation de balises dans une base de données mutualisée](#).

- CacheTTL

Durée en secondes pendant laquelle une entrée de contenu clé de balise dans le cache de balises local peut être utilisée avant son expiration. Cette valeur doit être supérieure à zéro. Lorsque la limite de cache TTL expire, l'entrée est expulsée du cache local.

- (Facultatif) Un cache

Si vous souhaitez personnaliser le type de cache ou le nombre d'entrées clés de branche pouvant être stockées dans le cache local, spécifiez le type de cache et la capacité d'entrée lorsque vous initialisez le trousseau de clés.

Le trousseau de clés hiérarchique prend en charge les types de cache suivants : par défaut `MultiThreaded`, `StormTracking`, et partagé. Pour plus d'informations et des exemples illustrant comment définir chaque type de cache, consultez [the section called "Choisissez un cache"](#).

Si vous ne spécifiez pas de cache, le trousseau de clés hiérarchique utilise automatiquement le type de cache par défaut et définit la capacité d'entrée à 1 000.

L'exemple suivant crée un jeu de clés hiérarchique avec un fournisseur d'ID de clé de branche, une limite de cache (TLL) de 600 secondes et une capacité d'entrée de 1 000.

Java

```
final MaterialProviders matProv = MaterialProviders.builder()
    .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
    .build();
final CreateAwsKmsHierarchicalKeyringInput keyringInput =
    CreateAwsKmsHierarchicalKeyringInput.builder()
        .keyStore(branchKeyName)
        .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 Porte-clés ECDH

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Important

Le porte-clés AWS KMS ECDH n'est disponible qu'avec la version 1.5.0 ou ultérieure de la bibliothèque des fournisseurs de matériaux.

Un porte-clés AWS KMS ECDH utilise un accord de clé asymétrique [AWS KMS keyspour](#) pour dériver une clé d'encapsulation symétrique partagée entre deux parties. Tout d'abord, le trousseau de clés utilise l'algorithme d'accord de clé Elliptic Curve Diffie-Hellman (ECDH) pour dériver un secret partagé à partir de la clé privée contenue dans la paire de clés KMS de l'expéditeur et de la clé publique du destinataire. Le trousseau de clés utilise ensuite le secret partagé pour dériver la clé d'encapsulation partagée qui protège vos clés de chiffrement des données. La fonction de dérivation de clé utilisée par le SDK AWS de chiffrement de base de données (KDF_CTR_HMAC_SHA384) pour dériver la clé d'encapsulation partagée est conforme aux [recommandations du NIST](#) pour la dérivation de clés.

La fonction de dérivation de clés renvoie 64 octets de matériel de saisie. Pour garantir que les deux parties utilisent le bon matériel de saisie, le SDK de chiffrement de AWS base de données utilise les 32 premiers octets comme clé d'engagement et les 32 derniers octets comme clé d'encapsulation partagée. Lors du déchiffrement, si le trousseau de clés ne peut pas reproduire la même clé d'engagement et la même clé d'encapsulation partagée que celles stockées dans le champ de description du matériel de l'enregistrement crypté, l'opération échoue. Par exemple, si vous chiffrez un enregistrement avec un trousseau de clés configuré avec la clé privée d'Alice et la clé publique de Bob, un trousseau de clés configuré avec la clé privée de Bob et la clé publique d'Alice reproduira la même clé d'engagement et la même clé d'encapsulation partagée et pourra déchiffrer l'enregistrement. Si la clé publique de Bob ne provient pas d'une paire de clés KMS, Bob peut créer un jeu de [clés ECDH brut](#) pour déchiffrer l'enregistrement.

Le trousseau de clés AWS KMS ECDH chiffre les enregistrements avec une clé symétrique à l'aide de l'AES-GCM. La clé de données est ensuite cryptée par enveloppe avec la clé d'encapsulation partagée dérivée à l'aide d'AES-GCM. [Chaque porte-clés AWS KMS ECDH ne peut avoir qu'une](#)

seule clé d'encapsulation partagée, mais vous pouvez inclure plusieurs porte-clés AWS KMS ECDH, seuls ou avec d'autres porte-clés, dans un porte-clés multiple.

Rubriques

- [Autorisations requises pour les AWS KMS porte-clés ECDH](#)
- [Création d'un AWS KMS porte-clés ECDH](#)
- [Création d'un AWS KMS porte-clés de découverte ECDH](#)

Autorisations requises pour les AWS KMS porte-clés ECDH

Le SDK AWS Database Encryption ne nécessite pas de AWS compte et ne dépend d'aucun AWS service. Toutefois, pour utiliser un porte-clés AWS KMS ECDH, vous devez disposer d'un AWS compte et des autorisations minimales suivantes sur le porte-clés de votre trousseau AWS KMS keys de clés. Les autorisations varient en fonction du schéma d'accord clé que vous utilisez.

- Pour chiffrer et déchiffrer des enregistrements à l'aide du schéma d'accord de `KmsPrivateKeyToStaticPublicKey` clés, vous avez besoin de [kms : GetPublicKey](#) et [kms : DeriveSharedSecret](#) sur la paire de clés KMS asymétrique de l'expéditeur. Si vous fournissez directement la clé publique codée DER de l'expéditeur lorsque vous instanciez votre jeu de clés, vous n'avez besoin que de l'`DeriveSharedSecret` autorisation [kms :](#) sur la paire de clés KMS asymétrique de l'expéditeur.
- Pour déchiffrer des enregistrements à l'aide du schéma d'accord de `KmsPublicKeyDiscovery` clés, vous avez besoin des `GetPublicKey` autorisations [kms : DeriveSharedSecret](#) et [kms :](#) sur la paire de clés KMS asymétrique spécifiée.

Création d'un AWS KMS porte-clés ECDH

Pour créer un jeu de clés AWS KMS ECDH qui chiffre et déchiffre les données, vous devez utiliser le schéma d'accord de clés. `KmsPrivateKeyToStaticPublicKey` Pour initialiser un trousseau de clés AWS KMS ECDH avec le schéma d'accord de `KmsPrivateKeyToStaticPublicKey` clés, fournissez les valeurs suivantes :

- AWS KMS key Identifiant de l'expéditeur

Doit identifier une paire de clés KMS à courbe elliptique (ECC) asymétrique recommandée par le NIST avec une valeur de `KeyUsage` `KEY_AGREEMENT` La clé privée de l'expéditeur est utilisée pour dériver le secret partagé.

- (Facultatif) Clé publique de l'expéditeur

[Il doit s'agir d'une clé publique X.509 codée DER, également connue sous le nom de SubjectPublicKeyInfo \(SPKI\), telle que définie dans la RFC 5280.](#)

L' AWS KMS [GetPublicKey](#) opération renvoie la clé publique d'une paire de clés KMS asymétriques dans le format CODÉ DER requis.

Pour réduire le nombre d' AWS KMS appels effectués par votre trousseau de clés, vous pouvez fournir directement la clé publique de l'expéditeur. Si aucune valeur n'est fournie pour la clé publique de l'expéditeur, le keyring appelle AWS KMS pour récupérer la clé publique de l'expéditeur.

- Clé publique du destinataire

[Vous devez fournir la clé publique X.509 codée DER du destinataire, également connue sous le nom de SubjectPublicKeyInfo \(SPKI\), telle que définie dans la RFC 5280.](#)

L' AWS KMS [GetPublicKey](#) opération renvoie la clé publique d'une paire de clés KMS asymétriques dans le format CODÉ DER requis.

- Spécification de la courbe

Identifie la spécification de la courbe elliptique dans les paires de clés spécifiées. Les paires de clés de l'expéditeur et du destinataire doivent avoir la même spécification de courbe.

Valeurs valides : ECC_NIST_P256, ECC_NIS_P384, ECC_NIST_P512

- (Facultatif) Une liste de jetons de subvention

Si vous contrôlez l'accès à la clé KMS dans votre trousseau de clés AWS KMS ECDH avec [des autorisations](#), vous devez fournir tous les jetons d'autorisation nécessaires lors de l'initialisation du trousseau de clés.

C# / .NET

L'exemple suivant crée un jeu de clés AWS KMS ECDH avec la clé KMS de l'expéditeur, la clé publique de l'expéditeur et la clé publique du destinataire. Cet exemple utilise le `senderPublicKey` paramètre facultatif pour fournir la clé publique de l'expéditeur. Si vous ne fournissez pas la clé publique de l'expéditeur, le keyring appelle AWS KMS pour récupérer la clé publique de l'expéditeur. Les paires de clés de l'expéditeur et du destinataire sont toutes deux en ECC_NIST_P256 évolution.

```
// 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

L'exemple suivant crée un jeu de clés AWS KMS ECDH avec la clé KMS de l'expéditeur, la clé publique de l'expéditeur et la clé publique du destinataire. Cet exemple utilise le `senderPublicKey` paramètre facultatif pour fournir la clé publique de l'expéditeur. Si vous ne fournissez pas la clé publique de l'expéditeur, le keyring appelle AWS KMS pour récupérer la clé publique de l'expéditeur. Les paires de clés de l'expéditeur et du destinataire sont toutes deux en ECC_NIST_P256 évolution.

```
// 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

L'exemple suivant crée un jeu de clés AWS KMS ECDH avec la clé KMS de l'expéditeur, la clé publique de l'expéditeur et la clé publique du destinataire. Cet exemple utilise le `sender_public_key` paramètre facultatif pour fournir la clé publique de l'expéditeur. Si vous ne fournissez pas la clé publique de l'expéditeur, le keyring appelle AWS KMS pour récupérer la clé publique de l'expéditeur.

```

// 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_identifieur(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?;
```

Création d'un AWS KMS porte-clés de découverte ECDH

Lors du déchiffrement, il est recommandé de spécifier les clés que le SDK de chiffrement de AWS base de données peut utiliser. Pour suivre cette bonne pratique, utilisez un porte-clés AWS KMS ECDH avec le schéma d'accord de `KmsPrivateKeyToStaticPublicKey` clés. Toutefois, vous pouvez également créer un jeu de clés de découverte AWS KMS ECDH, c'est-à-dire un trousseau de clés AWS KMS ECDH capable de déchiffrer tout enregistrement dont la clé publique de la paire de clés KMS spécifiée correspond à la clé publique du destinataire stockée dans le champ de description du matériel de l'enregistrement chiffré.

Important

Lorsque vous déchiffrez des enregistrements à l'aide du schéma d'accord de `KmsPublicKeyDiscovery` clés, vous acceptez toutes les clés publiques, quel que soit leur propriétaire.

Pour initialiser un trousseau de clés AWS KMS ECDH avec le schéma d'accord de `KmsPublicKeyDiscovery` clés, fournissez les valeurs suivantes :

- AWS KMS key Identifiant du destinataire

Doit identifier une paire de clés KMS à courbe elliptique (ECC) asymétrique recommandée par le NIST avec une valeur de `KeyUsage` `KEY_AGREEMENT`

- Spécification de la courbe

Identifie la spécification de courbe elliptique dans la paire de clés KMS du destinataire.

Valeurs valides : `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

- (Facultatif) Une liste de jetons de subvention

Si vous contrôlez l'accès à la clé KMS dans votre trousseau de clés AWS KMS ECDH avec [des autorisations](#), vous devez fournir tous les jetons d'autorisation nécessaires lors de l'initialisation du trousseau de clés.

C# / .NET

L'exemple suivant crée un trousseau de clés de découverte AWS KMS ECDH avec une paire de clés KMS sur la `ECC_NIST_P256` courbe. Vous devez disposer des `DeriveSharedSecret` autorisations [kms : GetPublicKey](#) et [kms :](#) sur la paire de clés KMS spécifiée. Ce porte-clés peut déchiffrer n'importe quel enregistrement où la clé publique de la paire de clés KMS spécifiée correspond à la clé publique du destinataire stockée dans le champ de description matérielle de l'enregistrement crypté.

```
// 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

L'exemple suivant crée un trousseau de clés de découverte AWS KMS ECDH avec une paire de clés KMS sur la ECC_NIST_P256 courbe. Vous devez disposer des `DeriveSharedSecret` autorisations [kms : GetPublicKey](#) et [kms :](#) sur la paire de clés KMS spécifiée. Ce porte-clés peut déchiffrer n'importe quel enregistrement où la clé publique de la paire de clés KMS spécifiée correspond à la clé publique du destinataire stockée dans le champ de description matérielle de l'enregistrement crypté.

```

// 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_idenfier(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?;
```

Porte-clés AES brut

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Le SDK AWS Database Encryption vous permet d'utiliser une clé symétrique AES que vous fournissez comme clé d'encapsulation qui protège votre clé de données. Vous devez générer, stocker et protéger le matériel clé, de préférence dans un module de sécurité matériel (HSM) ou un système de gestion des clés. Utilisez un trousseau de clés AES brut lorsque vous devez fournir la clé d'encapsulation et chiffrer les clés de données localement ou hors ligne.

Le jeu de clés AES brut chiffre les données à l'aide de l'algorithme AES-GCM et d'une clé d'encapsulation que vous spécifiez sous forme de tableau d'octets. [Vous ne pouvez spécifier qu'une seule clé d'encapsulation par jeu de clés AES brut, mais vous pouvez inclure plusieurs porte-clés AES bruts, seuls ou avec d'autres trousseaux de clés, dans un jeu de clés multiple.](#)

Espaces de noms et noms clés

Pour identifier la clé AES dans un trousseau de clés, le trousseau de clés AES brut utilise un espace de noms de clé et un nom de clé que vous fournissez. Ces valeurs ne sont pas secrètes. Ils apparaissent en texte brut dans la [description matérielle](#) que le SDK AWS de chiffrement de base de données ajoute à l'enregistrement. Nous vous recommandons d'utiliser un espace de noms de clé dans votre HSM ou votre système de gestion de clés et un nom de clé identifiant la clé AES dans ce système.

Note

L'espace de noms clé et le nom de clé sont équivalents aux champs ID du fournisseur (ou fournisseur) et ID clé du `JceMasterKey`.

Si vous créez différents trousseaux de clés pour chiffrer et déchiffrer un champ donné, l'espace de noms et les valeurs des noms sont essentiels. Si l'espace de noms de clé et le nom de clé du jeu de clés de déchiffrement ne correspondent pas exactement, en distinguant majuscules et minuscules, à l'espace de noms de clé et au nom de clé du jeu de clés de chiffrement, le jeu de clés de déchiffrement n'est pas utilisé, même si les octets essentiels sont identiques.

Par exemple, vous pouvez définir un trousseau de clés AES brut avec un espace de noms de clé `HSM_01` et un nom de clé `AES_256_012`. Ensuite, vous utilisez ce trousseau de clés pour chiffrer certaines données. Pour déchiffrer ces données, créez un jeu de clés AES brut avec le même espace de noms, le même nom de clé et le même matériau clé.

Les exemples suivants montrent comment créer un trousseau de clés AES brut. La `AESWrappingKey` variable représente le matériel clé que vous fournissez.

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?;
```

Porte-clés RSA bruts

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Le trousseau RSA brut effectue le chiffrement et le déchiffrement asymétriques des clés de données dans la mémoire locale à l'aide des clés publiques et privées RSA que vous fournissez. Vous devez générer, stocker et protéger la clé privée, de préférence dans un module de sécurité matériel (HSM) ou un système de gestion des clés. La fonction de chiffrement chiffre la clé de données sous la clé

publique RSA. La fonction de déchiffrement déchiffre la clé de données à l'aide de la clé privée. Vous pouvez choisir parmi plusieurs modes de remplissage RSA.

Un porte-clés RSA brut qui chiffre et déchiffre doit inclure une clé publique asymétrique et une clé privée en paire. Toutefois, vous pouvez chiffrer des données à l'aide d'un jeu de clés RSA brut contenant uniquement une clé publique, et vous pouvez déchiffrer des données à l'aide d'un jeu de clés RSA brut contenant uniquement une clé privée. [Vous pouvez inclure n'importe quel trousseau RSA brut dans un trousseau à clés multiples](#). Si vous configurez un jeu de clés RSA brut avec une clé publique et une clé privée, assurez-vous qu'elles font partie de la même paire de clés.

Le trousseau de clés RSA brut est équivalent au fichier [JceMasterKey](#) et interagit avec celui-ci Kit SDK de chiffrement AWS pour Java lorsqu'il est utilisé avec des clés de chiffrement asymétriques RSA.

Note

Le porte-clés RSA brut ne prend pas en charge les clés KMS asymétriques. [Pour utiliser des clés RSA KMS asymétriques, créez un AWS KMS trousseau de clés](#).

Espaces de noms et noms

Pour identifier le contenu clé RSA d'un trousseau de clés, le trousseau RSA brut utilise un espace de noms de clé et un nom de clé que vous fournissez. Ces valeurs ne sont pas secrètes. Ils apparaissent en texte brut dans la [description matérielle](#) que le SDK AWS de chiffrement de base de données ajoute à l'enregistrement. Nous vous recommandons d'utiliser l'espace de noms de clé et le nom de clé qui identifient la paire de clés RSA (ou sa clé privée) dans votre HSM ou votre système de gestion des clés.

Note

L'espace de noms clé et le nom de clé sont équivalents aux champs ID du fournisseur (ou fournisseur) et ID clé du `JceMasterKey`.

Si vous créez différents trousseaux de clés pour chiffrer et déchiffrer un enregistrement donné, l'espace de noms et les valeurs des noms sont essentiels. Si l'espace de noms de clé et le nom de clé du jeu de clés de déchiffrement ne correspondent pas exactement, en distinguant majuscules et

minuscules, à l'espace de noms de clé et au nom de clé du jeu de clés de chiffrement, le jeu de clés de déchiffrement n'est pas utilisé, même si les clés proviennent de la même paire de clés.

L'espace de noms de clé et le nom de clé du contenu clé des trousseaux de clés de chiffrement et de déchiffrement doivent être identiques, que le jeu de clés contienne la clé publique RSA, la clé privée RSA ou les deux clés de la paire de clés. Supposons, par exemple, que vous cryptiez des données à l'aide d'un jeu de clés RSA brut pour une clé publique RSA avec un espace de noms HSM_01 de clé et un nom de clé. RSA_2048_06 Pour déchiffrer ces données, créez un jeu de clés RSA brut avec la clé privée (ou paire de clés), ainsi que le même espace de noms et le même nom de clé.

Mode de rembourrage

Vous devez spécifier un mode de remplissage pour les porte-clés RSA bruts utilisés pour le chiffrement et le déchiffrement, ou utiliser les fonctionnalités de l'implémentation de votre langage qui le spécifient pour vous.

Il AWS Encryption SDK prend en charge les modes de remplissage suivants, sous réserve des contraintes de chaque langue. Nous recommandons un mode de rembourrage [OAEP](#), en particulier OAEP avec SHA-256 et avec rembourrage SHA-256. MGF1 Le mode de [PKCS1](#) rembourrage n'est pris en charge que pour des raisons de rétrocompatibilité.

- OAEP avec SHA-1 et rembourrage SHA-1 MGF1
- OAEP avec SHA-256 et avec rembourrage SHA-256 MGF1
- OAEP avec SHA-384 et avec rembourrage SHA-384 MGF1
- OAEP avec SHA-512 et avec rembourrage SHA-512 MGF1
- PKCS1 Rembourrage v1.5

L'exemple Java suivant montre comment créer un jeu de clés RSA brut avec les clés publique et privée d'une paire de clés RSA et l'OAEP avec SHA-256 et avec le mode de remplissage SHA-256. MGF1 Les `RSAPrivateKey` variables `RSAPublicKey` et représentent le matériel clé que vous fournissez.

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::0aepSha256Mgf1)
    .public_key(RSA_public_key)
    .private_key(RSA_private_key)

```

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

Porte-clés ECDH bruts

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Important

Le porte-clés Raw ECDH n'est disponible qu'avec la version 1.5.0 de la bibliothèque Material Providers.

Le porte-clés ECDH brut utilise les paires de clés publique-privée à courbe elliptique que vous fournissez pour dériver une clé d'encapsulation partagée entre deux parties. Tout d'abord, le trousseau de clés déduit un secret partagé à l'aide de la clé privée de l'expéditeur, de la clé publique du destinataire et de l'algorithme d'accord de clé Elliptic Curve Diffie-Hellman (ECDH). Le trousseau de clés utilise ensuite le secret partagé pour dériver la clé d'encapsulation partagée qui protège vos clés de chiffrement des données. La fonction de dérivation de clé utilisée par le SDK AWS de chiffrement de base de données (KDF_CTR_HMAC_SHA384) pour dériver la clé d'encapsulation partagée est conforme aux [recommandations du NIST](#) pour la dérivation de clés.

La fonction de dérivation de clés renvoie 64 octets de matériel de saisie. Pour garantir que les deux parties utilisent le bon matériel de saisie, le SDK de chiffrement de AWS base de données utilise les 32 premiers octets comme clé d'engagement et les 32 derniers octets comme clé d'encapsulation partagée. Lors du déchiffrement, si le trousseau de clés ne peut pas reproduire la même clé d'engagement et la même clé d'encapsulation partagée que celles stockées dans le champ de description du matériel de l'enregistrement crypté, l'opération échoue. Par exemple, si vous chiffrez un enregistrement avec un trousseau de clés configuré avec la clé privée d'Alice et la clé publique de Bob, un trousseau de clés configuré avec la clé privée de Bob et la clé publique d'Alice reproduira la même clé d'engagement et la même clé d'encapsulation partagée et pourra déchiffrer l'enregistrement. Si la clé publique de Bob provient d'une AWS KMS key paire, Bob peut créer un jeu de [clés AWS KMS ECDH](#) pour déchiffrer l'enregistrement.

Le trousseau de clés ECDH brut chiffre les enregistrements avec une clé symétrique à l'aide de l'AES-GCM. La clé de données est ensuite cryptée par enveloppe avec la clé d'encapsulation partagée dérivée à l'aide d'AES-GCM. [Chaque porte-clés Raw ECDH ne peut avoir qu'une seule clé d'encapsulation partagée, mais vous pouvez inclure plusieurs porte-clés Raw ECDH, seuls ou avec d'autres porte-clés, dans un porte-clés multiple.](#)

Vous êtes responsable de la génération, du stockage et de la protection de vos clés privées, de préférence dans un module de sécurité matériel (HSM) ou un système de gestion des clés. Les paires de clés de l'expéditeur et du destinataire doivent se trouver sur la même courbe elliptique. Le SDK AWS Database Encryption prend en charge les spécifications de courbe elliptique suivantes :

- ECC_NIST_P256
- ECC_NIST_P384
- ECC_NIST_P512

Création d'un porte-clés ECDH brut

Le trousseau de clés Raw ECDH prend en charge trois schémas d'accord de clés : `RawPrivateKeyToStaticPublicKey`, `EphemeralPrivateKeyToStaticPublicKey`, et `PublicKeyDiscovery`. Le schéma d'accord de clé que vous sélectionnez détermine les opérations cryptographiques que vous pouvez effectuer et la manière dont les matériaux de clé sont assemblés.

Rubriques

- [RawPrivateKeyToStaticPublicKey](#)
- [EphemeralPrivateKeyToStaticPublicKey](#)
- [PublicKeyDiscovery](#)

RawPrivateKeyToStaticPublicKey

Utilisez le schéma d'accord des `RawPrivateKeyToStaticPublicKey` clés pour configurer de manière statique la clé privée de l'expéditeur et la clé publique du destinataire dans le trousseau de clés. Ce schéma d'accord de clé permet de chiffrer et de déchiffrer des enregistrements.

Pour initialiser un jeu de clés ECDH brut avec le schéma d'accord de `RawPrivateKeyToStaticPublicKey` clés, fournissez les valeurs suivantes :

- Clé privée de l'expéditeur

[Vous devez fournir la clé privée codée PEM de l'expéditeur \(PrivateKeyInfo structures PKCS #8\), telle que définie dans la RFC 5958.](#)

- Clé publique du destinataire

[Vous devez fournir la clé publique X.509 codée DER du destinataire, également connue sous le nom de SubjectPublicKeyInfo \(SPKI\), telle que définie dans la RFC 5280.](#)

Vous pouvez spécifier la clé publique d'une paire de clés KMS à accord de clé asymétrique ou la clé publique à partir d'une paire de clés générée en dehors de AWS.

- Spécification de la courbe

Identifie la spécification de la courbe elliptique dans les paires de clés spécifiées. Les paires de clés de l'expéditeur et du destinataire doivent avoir la même spécification de courbe.

Valeurs valides : 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

L'exemple Java suivant utilise le schéma d'accord de `RawPrivateKeyToStaticPublicKey` clé pour configurer de manière statique la clé privée de l'expéditeur et la clé publique du destinataire. Les deux paires de clés sont sur la `ECC_NIST_P256` courbe.

```
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

L'exemple Python suivant utilise le schéma d'accord de `raw_ecdh_static_configuration` clé pour configurer de manière statique la clé privée de l'expéditeur et la clé publique du destinataire. Les deux paires de clés doivent se trouver sur la même courbe.

```
// 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_input);

// 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

Les porte-clés configurés avec le schéma d'accord de `EphemeralPrivateKeyToStaticPublicKey` clés créent une nouvelle paire de clés localement et dérivent une clé d'encapsulation partagée unique pour chaque appel de chiffrement.

Ce schéma d'accord clé ne peut chiffrer que des enregistrements. Pour déchiffrer les enregistrements chiffrés avec le schéma d'accord de `EphemeralPrivateKeyToStaticPublicKey` clé, vous devez utiliser un schéma d'accord de clé de découverte configuré avec la clé publique du même destinataire. Pour le déchiffrer, vous pouvez utiliser un jeu de clés ECDH brut avec l'algorithme d'accord de [PublicKeyDiscovery](#) clés ou, si la clé publique du destinataire provient d'une paire de

clés KMS à accord de clé asymétrique, vous pouvez utiliser un porte-clés AWS KMS ECDH avec le schéma d'accord de clés. [KmsPublicKeyDiscovery](#)

Pour initialiser un jeu de clés ECDH brut avec le schéma d'accord de EphemeralPrivateKeyToStaticPublicKey clés, fournissez les valeurs suivantes :

- Clé publique du destinataire

[Vous devez fournir la clé publique X.509 codée DER du destinataire, également connue sous le nom de SubjectPublicKeyInfo \(SPKI\), telle que définie dans la RFC 5280.](#)

Vous pouvez spécifier la clé publique d'une paire de clés KMS à accord de clé asymétrique ou la clé publique à partir d'une paire de clés générée en dehors de AWS.

- Spécification de la courbe

Identifie la spécification de la courbe elliptique dans la clé publique spécifiée.

Lors du chiffrement, le trousseau de clés crée une nouvelle paire de clés sur la courbe spécifiée et utilise la nouvelle clé privée et la clé publique spécifiée pour dériver une clé d'encapsulation partagée.

Valeurs valides : ECC_NIST_P256, ECC_NIS_P384, ECC_NIST_P512

C# / .NET

L'exemple suivant crée un trousseau de clés ECDH brut avec le schéma d'accord de EphemeralPrivateKeyToStaticPublicKey clés. Lors du chiffrement, le trousseau de clés créera une nouvelle paire de clés localement sur la courbe spécifiée 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

L'exemple suivant crée un trousseau de clés ECDH brut avec le schéma d'accord de EphemeralPrivateKeyToStaticPublicKey clés. Lors du chiffrement, le trousseau de clés créera une nouvelle paire de clés localement sur la courbe spécifiée 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

L'exemple suivant crée un trousseau de clés ECDH brut avec le schéma d'accord de `ephemeral_raw_ecdh_static_configuration` clés. Lors du chiffrement, le trousseau de clés créera une nouvelle paire de clés localement sur la courbe spécifiée.

```
// 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_configuration)

// 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

Lors du déchiffrement, il est recommandé de spécifier les clés d'encapsulation que le SDK de chiffrement de AWS base de données peut utiliser. Pour suivre cette bonne pratique, utilisez un trousseau de clés ECDH qui spécifie à la fois la clé privée de l'expéditeur et la clé publique du destinataire. Cependant, vous pouvez également créer un jeu de clés de découverte ECDH brut, c'est-à-dire un jeu de clés ECDH brut capable de déchiffrer tout enregistrement dont la clé publique spécifiée correspond à la clé publique du destinataire stockée dans le champ de description matérielle de l'enregistrement crypté. Ce schéma d'accord clé ne peut déchiffrer que les enregistrements.

⚠ Important

Lorsque vous déchiffrez des enregistrements à l'aide du schéma d'accord de `PublicKeyDiscovery` clés, vous acceptez toutes les clés publiques, quel que soit leur propriétaire.

Pour initialiser un jeu de clés ECDH brut avec le schéma d'accord de `PublicKeyDiscovery` clés, fournissez les valeurs suivantes :

- Clé privée statique du destinataire

[Vous devez fournir la clé privée codée PEM du destinataire \(`PrivateKeyInfo` structures PKCS #8\), telle que définie dans la RFC 5958.](#)

- Spécification de la courbe

Identifie la spécification de la courbe elliptique dans la clé privée spécifiée. Les paires de clés de l'expéditeur et du destinataire doivent avoir la même spécification de courbe.

Valeurs valides : `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

C# / .NET

L'exemple suivant crée un trousseau de clés ECDH brut avec le schéma d'accord de `PublicKeyDiscovery` clés. Ce porte-clés peut déchiffrer tout enregistrement dont la clé publique de la clé privée spécifiée correspond à la clé publique du destinataire stockée dans le champ de description du matériel de l'enregistrement crypté.

```
// 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

L'exemple suivant crée un trousseau de clés ECDH brut avec le schéma d'accord de `PublicKeyDiscovery` clés. Ce porte-clés peut déchiffrer tout enregistrement dont la clé publique de la clé privée spécifiée correspond à la clé publique du destinataire stockée dans le champ de description du matériel de l'enregistrement crypté.

```
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

L'exemple suivant crée un trousseau de clés ECDH brut avec le schéma d'accord de `discovery_raw_ecdh_static_configuration` clés. Ce porte-clés peut déchiffrer tout message dont la clé publique de la clé privée spécifiée correspond à la clé publique du destinataire enregistrée dans le texte chiffré du message.

```
// 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_input);

// 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?;
```

Porte-clés multiples

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Vous pouvez combiner plusieurs porte-clés au sein d'un porte-clés multiple. Un porte-clés multiple est un porte-clés qui se compose d'un ou plusieurs porte-clés individuels différents ou partageant le même type. L'effet est semblable à l'utilisation de plusieurs porte-clés en série. Lorsque vous utilisez

un porte-clés multiple pour chiffrer des données, toutes les clés d'encapsulation de tous ses porte-clés sont capables de déchiffrer les données.

Lorsque vous créez un porte-clés multiple pour chiffrer des données, vous désignez l'un des porte-clés en tant que porte-clés générateur. Tous les autres porte-clés sont appelés porte-clés enfants. Le porte-clés générateur génère et chiffre la clé de données en texte brut. Ensuite, toutes les clés d'encapsulation dans l'ensemble des porte-clés enfants chiffrent la même clé de données en texte brut. Le porte-clés multiple renvoie la clé en texte brut et une clé de données chiffrée pour chaque clé d'encapsulation du porte-clés multiple. Si le trousseau de clés du générateur est un [trousseau KMS](#), la clé du générateur du AWS KMS trousseau génère et chiffre la clé en texte brut. Ensuite, toutes les clés supplémentaires AWS KMS keys du AWS KMS trousseau de clés et toutes les clés enveloppantes de tous les porte-clés enfants du trousseau de clés multiples chiffrent la même clé en texte brut.

Lors du déchiffrement, le SDK de chiffrement de AWS base de données utilise les trousseaux de clés pour tenter de déchiffrer l'une des clés de données chiffrées. Les porte-clés sont appelés dans l'ordre dans lequel ils sont spécifiés dans le porte-clés multiple. Le traitement s'arrête dès qu'une clé d'un porte-clés peut déchiffrer une clé de données chiffrée.

Pour créer un porte-clés multiple, vous devez d'abord instancier les porte-clés enfants. Dans cet exemple, nous utilisons un AWS KMS porte-clés et un porte-clés AES brut, mais vous pouvez combiner tous les porte-clés compatibles dans un porte-clés multiple.

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?;
```

Ensuite, créez le porte-clés multiple et spécifiez son porte-clés générateur, le cas échéant. Dans cet exemple, nous créons un porte-clés multiple dans lequel le porte-clés est le AWS KMS porte-clés générateur et le porte-clés AES est le porte-clés enfant.

Java

Le `CreateMultiKeyringInput` constructeur Java vous permet de définir un porte-clés générateur et des trousseaux-clés enfants. L'objet `CreateMultiKeyringInput` obtenu est immuable.

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

C# / .NET

Le `CreateMultiKeyringInput` constructeur .NET vous permet de définir un porte-clés générateur et un trousseau de clés enfant. L'objet `CreateMultiKeyringInput` obtenu est immuable.

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

À présent, vous pouvez utiliser le porte-clés multiple pour chiffrer et déchiffrer les données.

Chiffrement consultable

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Le chiffrement consultable vous permet de rechercher des enregistrements cryptés sans déchiffrer l'intégralité de la base de données. Cela se fait à l'aide de balises, qui créent une carte entre la valeur en texte brut écrite dans un champ et la valeur cryptée réellement stockée dans votre base de données. Le SDK AWS Database Encryption stocke la balise dans un nouveau champ qu'il ajoute à l'enregistrement. Selon le type de balise que vous utilisez, vous pouvez effectuer des recherches de correspondance exacte ou des requêtes complexes plus personnalisées sur vos données chiffrées.

Note

[Le chiffrement consultable dans le SDK AWS de chiffrement de base de données est différent du chiffrement symétrique consultable défini dans la recherche universitaire, tel que le chiffrement symétrique consultable.](#)

Une balise est une balise HMAC (Hash Based Message Authentication Code) tronquée qui crée une carte entre le texte brut et les valeurs chiffrées d'un champ. Lorsque vous écrivez une nouvelle valeur dans un champ chiffré configuré pour le chiffrement consultable, le SDK de chiffrement de AWS base de données calcule un HMAC par rapport à la valeur en texte brut. Cette sortie HMAC correspond un à un (1:1) à la valeur en texte brut de ce champ. La sortie HMAC est tronquée afin que plusieurs valeurs de texte clair distinctes correspondent à la même balise HMAC tronquée. Ces faux positifs limitent la capacité d'un utilisateur non autorisé à identifier des informations distinctives concernant la valeur en texte brut. Lorsque vous interrogez une balise, le SDK AWS Database Encryption filtre automatiquement ces faux positifs et renvoie le résultat en texte brut de votre requête.

Le nombre moyen de faux positifs générés pour chaque balise est déterminé par la longueur restante de la balise après troncature. Pour obtenir de l'aide pour déterminer la longueur de balise appropriée pour votre implémentation, consultez la section [Détermination de la longueur de balise](#).

Note

Le chiffrement consultable est conçu pour être mis en œuvre dans de nouvelles bases de données non peuplées. Toute balise configurée dans une base de données existante ne cartographiera que les nouveaux enregistrements téléchargés dans la base de données, il n'y a aucun moyen pour une balise de mapper des données existantes.

Rubriques

- [Les balises sont-elles adaptées à mon ensemble de données ?](#)
- [Scénario de chiffrement consultable](#)

Les balises sont-elles adaptées à mon ensemble de données ?

L'utilisation de balises pour effectuer des requêtes sur des données chiffrées réduit les coûts de performance associés aux bases de données chiffrées côté client. Lorsque vous utilisez des balises, il existe un compromis inhérent entre l'efficacité de vos requêtes et la quantité d'informations révélées sur la distribution de vos données. La balise ne modifie pas l'état chiffré du champ. Lorsque vous chiffrez et signez un champ avec le SDK de chiffrement AWS de base de données, la valeur en texte brut du champ n'est jamais exposée à la base de données. La base de données stocke la valeur chiffrée aléatoire du champ.

Les balises sont stockées à côté des champs cryptés à partir desquels elles sont calculées. Cela signifie que même si un utilisateur non autorisé ne peut pas voir les valeurs en texte clair d'un champ crypté, il peut être en mesure d'effectuer une analyse statistique sur les balises pour en savoir plus sur la distribution de votre ensemble de données et, dans les cas extrêmes, identifier les valeurs en texte clair auxquelles une balise correspond. La façon dont vous configurez vos balises peut atténuer ces risques. En particulier, [le choix de la bonne longueur de balise](#) peut vous aider à préserver la confidentialité de votre ensemble de données.

Sécurité et performance

- Plus la longueur de la balise est courte, plus la sécurité est préservée.
- Plus la longueur de la balise est longue, plus les performances sont préservées.

Le chiffrement consultable peut ne pas être en mesure de fournir les niveaux de performance et de sécurité souhaités pour tous les ensembles de données. Passez en revue votre modèle de menace, vos exigences de sécurité et vos besoins en matière de performances avant de configurer des balises.

Tenez compte des exigences d'unicité des ensembles de données suivantes pour déterminer si le chiffrement consultable convient à votre ensemble de données.

Diffusion

Le niveau de sécurité préservé par une balise dépend de la distribution de votre ensemble de données. Lorsque vous configurez un champ chiffré pour le chiffrement consultable, le SDK de chiffrement AWS de base de données calcule un HMAC sur la base des valeurs en texte brut écrites dans ce champ. Toutes les balises calculées pour un champ donné sont calculées à l'aide de la même clé, à l'exception des bases de données mutualisées qui utilisent une clé distincte pour chaque locataire. Cela signifie que si la même valeur en texte brut est écrite plusieurs fois dans le champ, la même balise HMAC est créée pour chaque instance de cette valeur en texte clair.

Vous devez éviter de créer des balises à partir de champs contenant des valeurs très communes. Prenons l'exemple d'une base de données qui enregistre l'adresse de chaque résident de l'État de l'Illinois. Si vous créez une balise à partir du City champ crypté, la balise calculée sur « Chicago » sera surreprésentée en raison du pourcentage élevé de la population de l'Illinois qui vit à Chicago. Même si un utilisateur non autorisé peut uniquement lire les valeurs cryptées et les valeurs des balises, il peut être en mesure d'identifier les enregistrements contenant des données relatives aux résidents de Chicago si la balise préserve cette distribution. Pour minimiser la quantité d'informations distinctives révélées concernant votre distribution, vous devez suffisamment tronquer votre balise. La longueur de balise requise pour masquer cette distribution inégale entraîne des coûts de performance importants qui risquent de ne pas répondre aux besoins de votre application.

Vous devez analyser soigneusement la distribution de votre jeu de données afin de déterminer dans quelle mesure vos balises doivent être tronquées. La longueur de balise restante après la troncature est directement corrélée à la quantité d'informations statistiques pouvant être identifiées sur votre distribution. Vous devrez peut-être choisir des longueurs de balise plus courtes pour minimiser suffisamment la quantité d'informations distinctives révélées à propos de votre ensemble de données.

Dans les cas extrêmes, vous ne pouvez pas calculer la longueur d'une balise pour un jeu de données distribué de manière inégale afin d'équilibrer efficacement les performances et la sécurité. Par exemple, vous ne devez pas créer de balise à partir d'un champ qui contient le résultat d'un test médical pour une maladie rare. Étant donné que les NEGATIVE résultats devraient être nettement plus répandus dans l'ensemble de données, les POSITIVE résultats peuvent être facilement identifiés en fonction de leur rareté. Il est très difficile de masquer la distribution lorsque le champ ne comporte que deux valeurs possibles. Si vous utilisez une balise suffisamment courte pour masquer la distribution, toutes les valeurs en texte brut correspondent à la même balise HMAC. Si vous utilisez une longueur de balise plus longue, il est évident de savoir quelles balises correspondent à des valeurs en texte brut POSITIVE.

Corrélation

Nous vous recommandons vivement d'éviter de créer des balises distinctes à partir de champs contenant des valeurs corrélées. Les balises construites à partir de champs corrélés nécessitent des longueurs de balise plus courtes pour minimiser suffisamment la quantité d'informations révélées sur la distribution de chaque ensemble de données à un utilisateur non autorisé. Vous devez analyser soigneusement votre ensemble de données, notamment son entropie et la distribution conjointe des valeurs corrélées, afin de déterminer dans quelle mesure vos balises doivent être tronquées. Si la longueur des balises qui en résulte ne répond pas à vos besoins en termes de performances, les balises ne conviennent peut-être pas à votre ensemble de données.

Par exemple, vous ne devez pas créer deux balises City et deux ZIPCode champs distincts, car le code postal sera probablement associé à une seule ville. Généralement, les faux positifs générés par une balise limitent la capacité d'un utilisateur non autorisé à identifier des informations distinctives concernant votre ensemble de données. Mais la corrélation entre les ZIPCode champs City et signifie qu'un utilisateur non autorisé peut facilement identifier les résultats faussement positifs et distinguer les différents codes postaux.

Vous devez également éviter de créer des balises à partir de champs contenant les mêmes valeurs en texte brut. Par exemple, vous ne devez pas créer de balise à partir de preferredPhone champs mobilePhone et de champs, car ils contiennent probablement les mêmes valeurs. Si vous créez des balises distinctes à partir des deux champs, le SDK AWS de chiffrement de base de données crée les balises pour chaque champ sous des clés différentes. Il en résulte deux balises HMAC différentes pour la même valeur en texte brut. Il est peu probable que les deux balises distinctes présentent les mêmes faux positifs et un utilisateur non autorisé pourrait être en mesure de distinguer différents numéros de téléphone.

Même si votre jeu de données contient des champs corrélés ou présente une distribution inégale, vous pouvez peut-être créer des balises qui préservent la confidentialité de votre ensemble de données en utilisant des longueurs de balise plus courtes. Cependant, la longueur des balises ne garantit pas que chaque valeur unique de votre ensemble de données produira un certain nombre de faux positifs qui minimiseront efficacement la quantité d'informations distinctives révélées à propos de votre ensemble de données. La longueur de la balise estime uniquement le nombre moyen de faux positifs produits. Plus votre jeu de données est inégalement distribué, moins la longueur de balise est efficace pour déterminer le nombre moyen de faux positifs produits.

Examinez attentivement la distribution des champs à partir desquels vous construisez les balises et déterminez dans quelle mesure vous devrez tronquer la longueur des balises pour répondre à vos exigences de sécurité. Les rubriques suivantes de ce chapitre supposent que vos balises sont distribuées de manière uniforme et ne contiennent pas de données corrélées.

Scénario de chiffrement consultable

L'exemple suivant illustre une solution de chiffrement consultable simple. En application, les champs d'exemple utilisés dans cet exemple peuvent ne pas respecter les recommandations d'unicité de distribution et de corrélation pour les balises. Vous pouvez utiliser cet exemple comme référence lorsque vous découvrirez les concepts de chiffrement consultables dans ce chapitre.

Prenons l'exemple d'une base de données nommée `Employees` qui suit les données des employés d'une entreprise. Chaque enregistrement de la base de données contient des champs appelés `EmployeeID`, `LastName`, `FirstName`, et `Address`. Chaque champ de la `Employees` base de données est identifié par la clé primaire `EmployeeID`.

Voici un exemple d'enregistrement en texte brut dans la base de données.

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

Si vous avez marqué les `FirstName` champs `LastName` et comme `ENCRYPT_AND_SIGN` dans vos [actions cryptographiques](#), les valeurs de ces champs sont chiffrées localement avant d'être téléchargées dans la base de données. Les données cryptées téléchargées sont entièrement aléatoires, la base de données ne reconnaît pas ces données comme étant protégées. Il détecte simplement les entrées de données typiques. Cela signifie que l'enregistrement réellement stocké dans la base de données peut ressembler à ce qui suit.

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

Si vous devez interroger la base de données pour obtenir des correspondances exactes dans le `LastName` champ, [configurez une balise standard](#) nommée `LastName` pour mapper les valeurs en texte clair écrites dans le `LastName` champ aux valeurs cryptées stockées dans la base de données.

Cette balise effectue des calculs HMACs à partir des valeurs en texte brut du `LastName` champ. Chaque sortie HMAC est tronquée de sorte qu'elle ne correspond plus exactement à la valeur en texte brut. Par exemple, le hachage complet et le hachage tronqué pour Jones peuvent ressembler à ce qui suit.

Hachage complet

2aa4e9b404c68182562b6ec761fcca5306de527826a69468885e59dc36d0c3f824bdd44cab45526f

Hachage tronqué

b35099d408c833

Une fois la balise standard configurée, vous pouvez effectuer des recherches d'égalité sur le `LastName` terrain. Par exemple, si vous souhaitez effectuer une recherche Jones, utilisez la `LastName` balise pour effectuer la requête suivante.

```
LastName = Jones
```

Le SDK AWS Database Encryption filtre automatiquement les faux positifs et renvoie le résultat en texte brut de votre requête.

Balises

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Une balise est une balise HMAC (Hash Based Message Authentication Code) tronquée qui crée une carte entre la valeur en texte brut écrite dans un champ et la valeur cryptée réellement stockée dans votre base de données. La balise ne modifie pas l'état chiffré du champ. La balise calcule un HMAC sur la valeur en texte brut du champ et le stocke à côté de la valeur cryptée. Cette sortie HMAC correspond un à un (1:1) à la valeur en texte brut de ce champ. La sortie HMAC est tronquée afin que plusieurs valeurs de texte clair distinctes correspondent à la même balise HMAC tronquée. Ces faux positifs limitent la capacité d'un utilisateur non autorisé à identifier des informations distinctives concernant la valeur en texte brut.

Les balises ne peuvent être créées qu'à partir de champs marqués ou `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` dans `ENCRYPT_AND_SIGN` le `SIGN_ONLY` cadre de vos actions [cryptographiques](#). La balise elle-même n'est ni signée ni cryptée. Vous ne pouvez pas créer une balise avec des champs marqués `DO_NOTHING`.

Le type de balise que vous configurez détermine le type de requêtes que vous pouvez effectuer. Il existe deux types de balises compatibles avec le chiffrement consultable. Les balises standard effectuent des recherches d'égalité. Les balises composées combinent des chaînes de texte en clair littérales et des balises standard pour effectuer des opérations de base de données complexes. Après avoir [configuré vos balises](#), vous devez configurer un index secondaire pour chaque balise avant de pouvoir effectuer une recherche dans les champs chiffrés. Pour de plus amples informations, veuillez consulter [Configuration des index secondaires avec des balises](#).

Rubriques

- [Balises standard](#)
- [Balises composées](#)

Balises standard

Les balises standard constituent le moyen le plus simple d'implémenter un chiffrement consultable dans votre base de données. Ils ne peuvent effectuer des recherches d'égalité que pour un seul champ crypté ou virtuel. Pour savoir comment configurer les balises standard, consultez la section [Configuration des balises standard](#).

Le champ à partir duquel une balise standard est construite est appelé source de balise. Il identifie l'emplacement des données que la balise doit cartographier. La source de la balise peut être un champ crypté ou un champ virtuel. La source de balise de chaque balise standard doit être unique. Vous ne pouvez pas configurer deux balises avec la même source de balise.

Les balises standard peuvent être utilisées pour effectuer des recherches d'égalité pour un champ crypté ou virtuel. Ils peuvent également être utilisés pour créer des balises composées afin d'effectuer des opérations de base de données plus complexes. Pour vous aider à organiser et à gérer les balises standard, le SDK AWS de chiffrement de base de données fournit les styles de balises facultatifs suivants qui définissent l'utilisation prévue d'une balise standard. Pour plus d'informations, voir [Définition des styles de balises](#).

Vous pouvez créer une balise standard qui effectue des recherches d'égalité pour un seul champ crypté, ou vous pouvez créer une balise standard qui effectue des recherches d'égalité sur la concaténation de plusieurs `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` champs `ENCRYPT_AND_SIGNSIGN_ONLY`, et en créant un champ virtuel.

Champs virtuels

Un champ virtuel est un champ conceptuel construit à partir d'un ou de plusieurs champs sources. La création d'un champ virtuel n'entraîne pas l'inscription d'un nouveau champ dans votre enregistrement. Le champ virtuel n'est pas explicitement stocké dans votre base de données. Il est utilisé dans une configuration de balise standard pour donner à la balise des instructions sur la façon d'identifier un segment spécifique d'un champ ou de concaténer plusieurs champs dans un enregistrement pour effectuer une requête spécifique. Un champ virtuel nécessite au moins un champ crypté.

Note

L'exemple suivant montre les types de transformations et de requêtes que vous pouvez effectuer avec un champ virtuel. En application, les champs d'exemple utilisés dans cet exemple peuvent ne pas respecter les recommandations d'unicité de [distribution](#) et de [corrélation](#) pour les balises.

Par exemple, si vous souhaitez effectuer des recherches d'égalité sur la concaténation de LastName champs FirstName et, vous pouvez créer l'un des champs virtuels suivants.

- Un NameTag champ virtuel, construit à partir de la première lettre du FirstName champ, suivie du LastName champ, le tout en minuscules. Ce champ virtuel vous permet d'effectuer des requêtesNameTag=mjones.
- Un LastFirst champ virtuel, qui est construit à partir du LastName champ, suivi du FirstName champ. Ce champ virtuel vous permet d'effectuer des requêtesLastFirst=JonesMary.

Ou, si vous souhaitez effectuer des recherches d'égalité sur un segment spécifique d'un champ chiffré, créez un champ virtuel qui identifie le segment que vous souhaitez interroger.

Par exemple, si vous souhaitez interroger un IPAddress champ chiffré à l'aide des trois premiers segments de l'adresse IP, créez le champ virtuel suivant.

- Un IPSegment champ virtuel, construit à partir deSegments('.', 0, 3). Ce champ virtuel vous permet d'effectuer des requêtesIPSegment=192.0.2. La requête renvoie tous les enregistrements dont IPAddress la valeur commence par « 192.0.2 ».

Les champs virtuels doivent être uniques. Il est impossible de créer deux champs virtuels à partir des mêmes champs sources.

Pour obtenir de l'aide sur la configuration des champs virtuels et des balises qui les utilisent, consultez la section [Création d'un champ virtuel](#).

Balises composées

Les balises composées créent des index qui améliorent les performances des requêtes et vous permettent d'effectuer des opérations de base de données plus complexes. Vous pouvez utiliser des balises composées pour combiner des chaînes de texte brut littérales et des balises standard pour effectuer des requêtes complexes sur des enregistrements chiffrés, telles que l'interrogation de

deux types d'enregistrements différents à partir d'un seul index ou l'interrogation d'une combinaison de champs à l'aide d'une clé de tri. Pour d'autres exemples de solutions de balises composées, voir [Choisir un type de balise](#).

Les balises composées peuvent être construites à partir de balises standard ou d'une combinaison de balises standard et de champs signés. Ils sont construits à partir d'une liste de pièces. Toutes les balises composées doivent inclure une liste de [parties cryptées](#) identifiant les ENCRYPT_AND_SIGN champs inclus dans la balise. Chaque ENCRYPT_AND_SIGN champ doit être identifié par une balise standard. Les balises composées plus complexes peuvent également inclure une liste de [parties signées](#) identifiant le texte en clair SIGN_ONLY ou SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT les champs inclus dans la balise, et une liste de [pièces de constructeur](#) identifiant toutes les manières possibles dont la balise composée peut assembler les champs.

Note

Le SDK AWS Database Encryption prend également en charge les balises signées qui peuvent être entièrement configurées à partir de texte brut SIGN_ONLY et de champs. SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT Les balises signées sont un type de balise composée qui indexe et exécute des requêtes complexes sur des champs signés, mais non chiffrés. Pour de plus amples informations, veuillez consulter [Création de balises signées](#).

Pour obtenir de l'aide sur la configuration des balises composées, consultez [la section Configuration des balises composées](#).

La façon dont vous configurez votre balise composée détermine les types de requêtes qu'elle peut effectuer. Par exemple, vous pouvez rendre certaines parties cryptées et signées facultatives pour permettre une plus grande flexibilité dans vos requêtes. Pour plus d'informations sur les types de requêtes que les balises composées peuvent effectuer, consultez [Interrogation de balises](#).

Balises de planification

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Les balises sont conçues pour être mises en œuvre dans de nouvelles bases de données non peuplées. Toute balise configurée dans une base de données existante ne mapperait que les nouveaux enregistrements écrits dans la base de données. Les balises sont calculées à partir de la valeur en texte brut d'un champ. Une fois le champ crypté, il n'est plus possible pour la balise de mapper les données existantes. Une fois que vous avez écrit de nouveaux enregistrements avec une balise, vous ne pouvez pas mettre à jour la configuration de la balise. Cependant, vous pouvez ajouter de nouvelles balises pour les nouveaux champs que vous ajoutez à votre enregistrement.

Pour implémenter le chiffrement consultable, vous devez utiliser le trousseau de [clés AWS KMS hiérarchique](#) pour générer, chiffrer et déchiffrer les clés de données utilisées pour protéger vos enregistrements. Pour de plus amples informations, veuillez consulter [Utilisation du trousseau de clés hiérarchique pour un chiffrement consultable](#).

Avant de configurer les [balises](#) pour le chiffrement consultable, vous devez passer en revue vos exigences en matière de chiffrement, vos modèles d'accès à la base de données et votre modèle de menace afin de déterminer la meilleure solution pour votre base de données.

Le [type de balise](#) que vous configurez détermine le type de requêtes que vous pouvez effectuer. La [longueur de balise](#) que vous spécifiez dans la configuration de balise standard détermine le nombre attendu de faux positifs produits pour une balise donnée. Nous vous recommandons vivement d'identifier et de planifier les types de requêtes que vous devez effectuer avant de configurer vos balises. Une fois que vous avez utilisé une balise, la configuration ne peut pas être mise à jour.

Nous vous recommandons vivement de passer en revue et d'effectuer les tâches suivantes avant de configurer des balises.

- [Déterminez si les balises sont adaptées à votre ensemble de données](#)
- [Choisissez un type de balise](#)
- [Choisissez une longueur de balise](#)
- [Choisissez un nom de balise](#)

N'oubliez pas les exigences d'unicité des balises suivantes lorsque vous planifiez la solution de chiffrement consultable pour votre base de données.

- Chaque balise standard doit avoir une [source de balise](#) unique

Il est impossible de créer plusieurs balises standard à partir du même champ crypté ou virtuel.

Cependant, une seule balise standard peut être utilisée pour construire plusieurs balises composées.

- Évitez de créer un champ virtuel dont les champs source se chevauchent avec les balises standard existantes

La construction d'une balise standard à partir d'un champ virtuel contenant un champ source utilisé pour créer une autre balise standard peut réduire la sécurité des deux balises.

Pour de plus amples informations, veuillez consulter [Considérations relatives à la sécurité des champs virtuels](#).

Considérations relatives aux bases de données mutualisées

Pour interroger les balises configurées dans une base de données mutualisée, vous devez inclure le champ qui stocke le fichier `branch-key-id` associé au locataire qui a chiffré l'enregistrement dans votre requête. Vous définissez ce champ lorsque vous [définissez la source de la clé de balise](#). Pour que la requête aboutisse, la valeur de ce champ doit identifier les éléments clés de balise appropriés requis pour recalculer la balise.

Avant de configurer vos balises, vous devez décider de la manière dont vous comptez les inclure `branch-key-id` dans vos requêtes. Pour plus d'informations sur les différentes manières dont vous pouvez inclure le `branch-key-id` dans vos requêtes, consultez [Interrogation de balises dans une base de données mutualisée](#).

Choisir un type de balise

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Grâce au chiffrement consultable, vous pouvez rechercher des enregistrements chiffrés en mappant les valeurs en texte brut dans un champ crypté à l'aide d'une balise. Le type de balise que vous configurez détermine le type de requêtes que vous pouvez effectuer.

Nous vous recommandons vivement d'identifier et de planifier les types de requêtes que vous devez effectuer avant de configurer vos balises. Après avoir [configuré vos balises](#), vous devez configurer

un index secondaire pour chaque balise avant de pouvoir effectuer une recherche dans les champs chiffrés. Pour de plus amples informations, veuillez consulter [Configuration des index secondaires avec des balises](#).

Les balises créent une carte entre la valeur en texte brut écrite dans un champ et la valeur cryptée réellement stockée dans votre base de données. Vous ne pouvez pas comparer les valeurs de deux balises standard, même si elles contiennent le même texte brut sous-jacent. Les deux balises standard produiront deux balises HMAC différentes pour les mêmes valeurs en texte brut. Par conséquent, les balises standard ne peuvent pas effectuer les requêtes suivantes.

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

Vous ne pouvez effectuer les requêtes ci-dessus que si vous comparez les [parties signées](#) des balises composées, à l'exception de l'CONTAINSopérateur, que vous pouvez utiliser avec les balises composées pour identifier la valeur complète d'un champ chiffré ou signé que contient la balise assemblée. Lorsque vous comparez des parties signées, vous pouvez éventuellement inclure le préfixe d'une [partie chiffrée](#), mais vous ne pouvez pas inclure la valeur cryptée d'un champ. Pour plus d'informations sur les types de requêtes que les balises standard et composées peuvent effectuer, consultez la section [Interrogation](#) de balises.

Envisagez les solutions de chiffrement consultables suivantes lorsque vous examinez les modèles d'accès à votre base de données. Les exemples suivants définissent la balise à configurer pour répondre aux différentes exigences de chiffrement et d'interrogation.

Balises standard

[Les balises standard](#) peuvent uniquement effectuer des recherches d'égalité. Vous pouvez utiliser des balises standard pour effectuer les requêtes suivantes.

Interrogez un seul champ chiffré

Si vous souhaitez identifier les enregistrements contenant une valeur spécifique pour un champ chiffré, créez une balise standard.

Exemples

Dans l'exemple suivant, considérez une base de données nommée `UnitInspection` qui suit les données d'inspection d'une installation de production. Chaque enregistrement de la base de données contient des champs appelés `work_id`, `inspection_date`, `inspector_id_last4`, et `unit`. L'identifiant complet de l'inspecteur est un nombre compris entre 0 et 99 999 999. Toutefois, pour garantir une distribution uniforme de l'ensemble de données, `inspector_id_last4` seuls les quatre derniers chiffres de l'identifiant de l'inspecteur sont enregistrés. Chaque champ de la base de données est identifié par la clé primaire `work_id`. Les unit champs `inspector_id_last4` et sont marqués `ENCRYPT_AND_SIGN` dans les [actions cryptographiques](#).

Voici un exemple d'entrée en texte brut dans la `UnitInspection` base de données.

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

Interroger un seul champ chiffré dans un enregistrement

Si le `inspector_id_last4` champ doit être chiffré, mais que vous devez tout de même l'interroger pour obtenir des correspondances exactes, créez une balise standard à partir du `inspector_id_last4` champ. Utilisez ensuite la balise standard pour créer un index secondaire. Vous pouvez utiliser cet index secondaire pour effectuer une requête sur le `inspector_id_last4` champ crypté.

Pour obtenir de l'aide sur la configuration des balises standard, consultez [la section Configuration des balises standard](#).

Interroger un champ virtuel

Un [champ virtuel](#) est un champ conceptuel construit à partir d'un ou de plusieurs champs sources. Si vous souhaitez effectuer des recherches d'égalité pour un segment spécifique d'un champ crypté, ou effectuer des recherches d'égalité sur la concaténation de plusieurs champs, créez une balise standard à partir d'un champ virtuel. Tous les champs virtuels doivent inclure au moins un champ source chiffré.

Exemples

Les exemples suivants créent des champs virtuels pour la Employees base de données. Voici un exemple d'enregistrement en texte brut dans la Employees base de données.

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

Interroger un segment d'un champ chiffré

Dans cet exemple, le SSN champ est crypté.

Si vous souhaitez interroger le SSN champ à l'aide des quatre derniers chiffres d'un numéro de sécurité sociale, créez un champ virtuel qui identifie le segment que vous souhaitez interroger.

Un Last4SSN champ virtuel, construit à partir de, vous `Suffix(4)` permet d'effectuer des requêtes `Last4SSN=0000`. Utilisez ce champ virtuel pour créer une balise standard. Utilisez ensuite la balise standard pour créer un index secondaire. Vous pouvez utiliser cet index secondaire pour effectuer des requêtes sur le champ virtuel. Cette requête renvoie tous les enregistrements dont SSN la valeur se termine par les quatre derniers chiffres que vous avez spécifiés.

Interrogez la concaténation de plusieurs champs

Note

L'exemple suivant montre les types de transformations et de requêtes que vous pouvez effectuer avec un champ virtuel. En application, les champs d'exemple utilisés dans cet exemple peuvent ne pas respecter les recommandations d'unicité de [distribution](#) et de [corrélation](#) pour les balises.

Si vous souhaitez effectuer des recherches d'égalité sur une concaténation de `LastName` champs `FirstName` et, vous pouvez créer un `NameTag` champ virtuel, construit à partir de la première lettre du `FirstName` champ, suivie du champ, le `LastName` tout en minuscules. Utilisez ce champ virtuel pour créer une balise standard. Utilisez ensuite la balise standard pour créer un index secondaire. Vous pouvez utiliser cet index secondaire pour effectuer des requêtes `NameTag=mjones` sur le champ virtuel.

Au moins un des champs source doit être chiffré. L'`FirstName` ou l'autre `LastName` pourrait être chiffré, ou les deux pourraient être chiffrés. Tous les champs source en texte brut doivent être marqués en tant que `SIGN_ONLY` ou `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` dans vos actions [cryptographiques](#).

Pour obtenir de l'aide sur la configuration des champs virtuels et des balises qui les utilisent, consultez la section [Création d'un champ virtuel](#).

Balises composées

[Les balises composées](#) créent un index à partir de chaînes de texte brut littérales et de balises standard pour effectuer des opérations de base de données complexes. Vous pouvez utiliser des balises composées pour effectuer les requêtes suivantes.

Interrogez une combinaison de champs chiffrés sur un seul index

Si vous devez interroger une combinaison de champs chiffrés sur un seul index, créez une balise composée qui combine les balises standard individuelles construites pour chaque champ chiffré afin de former un index unique.

Après avoir configuré la balise composée, vous pouvez créer un index secondaire qui spécifie la balise composée comme clé de partition pour effectuer des requêtes de correspondance exacte ou avec une clé de tri pour effectuer des requêtes plus complexes. Les index secondaires qui spécifient la balise composée comme clé de tri peuvent effectuer des requêtes de correspondance exacte et des requêtes complexes plus personnalisées.

Exemples

Pour les exemples suivants, considérez une base de données nommée `UnitInspection` qui suit les données d'inspection d'une installation de production. Chaque enregistrement de la base de données contient des champs appelés `work_idinspection_date`, `inspector_id_last4`, et `unit`. L'identifiant complet de l'inspecteur est un nombre compris entre 0 et 99 999 999. Toutefois, pour garantir une distribution uniforme de l'ensemble de données, `inspector_id_last4` seuls les

quatre derniers chiffres de l'identifiant de l'inspecteur sont enregistrés. Chaque champ de la base de données est identifié par la clé primaire `work_id`. Les unit champs `inspector_id_last4` et sont marqués `ENCRYPT_AND_SIGN` dans les [actions cryptographiques](#).

Voici un exemple d'entrée en texte brut dans la `UnitInspection` base de données.

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

Effectuez des recherches d'égalité sur une combinaison de champs cryptés

Si vous souhaitez interroger la `UnitInspection` base de données pour obtenir des correspondances exactes `inspector_id_last4.unit`, créez d'abord des balises standard distinctes pour les unit champs `inspector_id_last4` et. Créez ensuite une balise composée à partir des deux balises standard.

Après avoir configuré la balise composée, créez un index secondaire qui spécifie la balise composée comme clé de partition. Utilisez cet index secondaire pour rechercher des correspondances exactes sur `inspector_id_last4.unit`. Par exemple, vous pouvez interroger cette balise pour trouver une liste des inspections effectuées par un inspecteur pour une unité donnée.

Réaliser des requêtes complexes sur une combinaison de champs chiffrés

Si vous souhaitez interroger la `UnitInspection` base de données sur `inspector_id_last4` et `inspector_id_last4.unit`, créez d'abord des balises standard distinctes pour les unit champs `inspector_id_last4` et. Créez ensuite une balise composée à partir des deux balises standard.

Après avoir configuré la balise composée, créez un index secondaire qui spécifie la balise composée comme clé de tri. Utilisez cet index secondaire pour rechercher dans la `UnitInspection` base de données les entrées commençant par un certain inspecteur ou pour demander à la base de données une liste de toutes les unités appartenant à une plage d'identifiants d'unités spécifique qui ont été inspectées par un certain inspecteur. Vous pouvez également effectuer des recherches de correspondance exacte sur `inspector_id_last4.unit`.

Pour obtenir de l'aide sur la configuration des balises composées, consultez [la section Configuration des balises composées](#).

Interrogez une combinaison de champs chiffrés et de champs en texte brut sur un seul index

Si vous devez interroger une combinaison de champs chiffrés et de champs en texte brut sur un seul index, créez une balise composée qui combine des balises standard individuelles et des champs de texte en clair pour former un index unique. Les champs de texte brut utilisés pour créer la balise composée doivent être marqués `SIGN_ONLY` ou figurer `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` dans vos actions [cryptographiques](#).

Après avoir configuré la balise composée, vous pouvez créer un index secondaire qui spécifie la balise composée comme clé de partition pour effectuer des requêtes de correspondance exacte ou avec une clé de tri pour effectuer des requêtes plus complexes. Les index secondaires qui spécifient la balise composée comme clé de tri peuvent effectuer des requêtes de correspondance exacte et des requêtes complexes plus personnalisées.

Exemples

Pour les exemples suivants, considérez une base de données nommée `UnitInspection` qui suit les données d'inspection d'une installation de production. Chaque enregistrement de la base de données contient des champs appelés `work_id`, `inspection_date`, `inspector_id_last4`, et `unit`. L'identifiant complet de l'inspecteur est un nombre compris entre 0 et 99 999 999. Toutefois, pour garantir une distribution uniforme de l'ensemble de données, `inspector_id_last4` seuls les quatre derniers chiffres de l'identifiant de l'inspecteur sont enregistrés. Chaque champ de la base de données est identifié par la clé primaire `work_id`. Les champs `inspector_id_last4` et `unit` sont marqués `ENCRYPT_AND_SIGN` dans les [actions cryptographiques](#).

Voici un exemple d'entrée en texte brut dans la `UnitInspection` base de données.

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

Effectuez des recherches d'égalité sur une combinaison de champs

Si vous souhaitez interroger la `UnitInspection` base de données pour les inspections effectuées par un inspecteur spécifique à une date précise, créez d'abord une balise standard

pour le `inspector_id_last4` champ. Le `inspector_id_last4` champ est marqué `ENCRYPT_AND_SIGN` dans les [actions cryptographiques](#). Toutes les parties cryptées nécessitent leur propre balise standard. Le `inspection_date` champ est marqué `SIGN_ONLY` et ne nécessite pas de balise standard. Créez ensuite une balise composée à partir du `inspection_date` champ et de la balise `inspector_id_last4` standard.

Après avoir configuré la balise composée, créez un index secondaire qui spécifie la balise composée comme clé de partition. Utilisez cet index secondaire pour rechercher dans les bases de données des enregistrements correspondant exactement à un inspecteur et à une date d'inspection spécifiques. Par exemple, vous pouvez interroger la base de données pour obtenir la liste de toutes les inspections effectuées à une date précise par l'inspecteur dont l'identifiant se termine 8744 par un identifiant.

Exécuter des requêtes complexes sur une combinaison de champs

Si vous souhaitez interroger la base de données pour les inspections effectuées dans une `inspection_date` plage, ou interroger la base de données pour les inspections effectuées sur une `inspection_date` contrainte particulière par `inspector_id_last4` ou `inspector_id_last4.unit`, créez d'abord des balises standard distinctes pour les champs `inspector_id_last4` et `unit`. Créez ensuite une balise composée à partir du `inspection_date` champ de texte brut et des deux balises standard.

Après avoir configuré la balise composée, créez un index secondaire qui spécifie la balise composée comme clé de tri. Utilisez cet index secondaire pour effectuer des requêtes pour les inspections effectuées à des dates spécifiques par un inspecteur spécifique. Par exemple, vous pouvez interroger la base de données pour obtenir la liste de toutes les unités inspectées à la même date. Vous pouvez également interroger la base de données pour obtenir une liste de toutes les inspections effectuées sur une unité spécifique entre une plage de dates d'inspection donnée.

Pour obtenir de l'aide sur la configuration des balises composées, consultez [la section Configuration des balises composées](#).

Choix de la longueur d'une balise

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Lorsque vous écrivez une nouvelle valeur dans un champ chiffré configuré pour le chiffrement consultable, le SDK de chiffrement de AWS base de données calcule un HMAC par rapport à la valeur en texte brut. Cette sortie HMAC correspond un à un (1:1) à la valeur en texte brut de ce champ. La sortie HMAC est tronquée afin que plusieurs valeurs de texte clair distinctes correspondent à la même balise HMAC tronquée. Ces collisions, ou faux positifs, limitent la capacité d'un utilisateur non autorisé à identifier des informations distinctives concernant la valeur en texte brut.

Le nombre moyen de faux positifs générés pour chaque balise est déterminé par la longueur restante de la balise après troncature. Vous devez uniquement définir la longueur des balises lors de la configuration des balises standard. Les balises composées utilisent les longueurs des balises standard à partir desquelles elles sont construites.

La balise ne modifie pas l'état chiffré du champ. Cependant, lorsque vous utilisez des balises, il existe un compromis inhérent entre l'efficacité de vos requêtes et la quantité d'informations révélées sur la distribution de vos données.

L'objectif du chiffrement consultable est de réduire les coûts de performance associés aux bases de données chiffrées côté client en utilisant des balises pour effectuer des requêtes sur des données chiffrées. Les balises sont stockées à côté des champs cryptés à partir desquels elles sont calculées. Cela signifie qu'ils peuvent révéler des informations distinctives sur la distribution de votre ensemble de données. Dans des cas extrêmes, un utilisateur non autorisé peut être en mesure d'analyser les informations révélées sur votre distribution et de les utiliser pour identifier la valeur en texte brut d'un champ. Le choix de la bonne longueur de balise peut contribuer à atténuer ces risques et à préserver la confidentialité de votre distribution.

Passez en revue votre modèle de menace pour déterminer le niveau de sécurité dont vous avez besoin. Par exemple, plus il y a de personnes qui ont accès à votre base de données, mais ne devraient pas avoir accès aux données en texte brut, plus vous souhaitez peut-être protéger la confidentialité de la distribution de votre ensemble de données. Pour accroître la confidentialité, une balise doit générer davantage de faux positifs. L'augmentation de la confidentialité entraîne une réduction des performances des requêtes.

Sécurité et performance

- Une balise trop longue produit trop peu de faux positifs et peut révéler des informations distinctives sur la distribution de votre ensemble de données.
- Une balise trop courte produit trop de faux positifs et augmente le coût des performances des requêtes car elle nécessite une analyse plus approfondie de la base de données.

Lorsque vous déterminez la longueur de balise appropriée pour votre solution, vous devez trouver une longueur qui préserve de manière adéquate la sécurité de vos données sans affecter les performances de vos requêtes plus que ce qui est absolument nécessaire. Le niveau de sécurité préservé par une balise dépend de la [distribution](#) de votre ensemble de données et de la [corrélation](#) des champs à partir desquels vos balises sont construites. Les rubriques suivantes supposent que vos balises sont distribuées de manière uniforme et ne contiennent pas de données corrélées.

Rubriques

- [Calcul de la longueur des balises](#)
- [Exemple](#)

Calcul de la longueur des balises

La longueur de la balise est définie en bits et fait référence au nombre de bits de la balise HMAC conservés après la troncature. La longueur de balise recommandée varie en fonction de la distribution du jeu de données, de la présence de valeurs corrélées et de vos exigences spécifiques en matière de sécurité et de performances. Si votre jeu de données est distribué de manière uniforme, vous pouvez utiliser les équations et procédures suivantes pour identifier la meilleure longueur de balise pour votre implémentation. Ces équations ne font qu'estimer le nombre moyen de faux positifs produits par la balise. Elles ne garantissent pas que chaque valeur unique de votre jeu de données produira un nombre spécifique de faux positifs.

Note

L'efficacité de ces équations dépend de la distribution de votre jeu de données. Si votre jeu de données n'est pas distribué de manière uniforme, consultez [Les balises sont-elles adaptées à mon ensemble de données ?](#).

En général, plus votre jeu de données est éloigné d'une distribution uniforme, plus vous devez raccourcir la longueur de votre balise.

1.

Estimer la population

La population est le nombre attendu de valeurs uniques dans le champ à partir duquel votre balise standard est construite, il ne s'agit pas du nombre total attendu de valeurs stockées dans le champ. Prenons l'exemple d'un Room champ crypté qui identifie le lieu des réunions

des employés. Le Room champ devrait stocker 100 000 valeurs au total, mais les employés ne peuvent réserver que 50 salles différentes pour les réunions. Cela signifie que la population est de 50 car seules 50 valeurs uniques peuvent être stockées Room sur le terrain.

Note

Si votre balise standard est construite à partir d'un [champ virtuel](#), la population utilisée pour calculer la longueur de la balise est le nombre de combinaisons uniques créées par le champ virtuel.

Lorsque vous estimez votre population, veillez à tenir compte de la croissance prévue de l'ensemble de données. Une fois que vous avez écrit de nouveaux enregistrements avec la balise, vous ne pouvez pas mettre à jour la longueur de la balise. Passez en revue votre modèle de menace et toutes les solutions de base de données existantes afin de créer une estimation du nombre de valeurs uniques que vous prévoyez de stocker dans ce champ au cours des cinq prochaines années.

Votre population n'a pas besoin d'être précise. Identifiez d'abord le nombre de valeurs uniques dans votre base de données actuelle ou estimez le nombre de valeurs uniques que vous comptez stocker au cours de la première année. Posez ensuite les questions suivantes pour déterminer la croissance prévue des valeurs uniques au cours des cinq prochaines années.

- Vous attendez-vous à ce que les valeurs uniques soient multipliées par 10 ?
- Vous attendez-vous à ce que les valeurs uniques soient multipliées par 100 ?
- Vous attendez-vous à ce que les valeurs uniques soient multipliées par 1000 ?

La différence entre 50 000 et 60 000 valeurs uniques n'est pas significative et elles aboutiront toutes deux à la même longueur de balise recommandée. Cependant, la différence entre 50 000 et 500 000 valeurs uniques aura un impact significatif sur la longueur de balise recommandée.

Envisagez d'examiner les données publiques sur la fréquence des types de données courants, tels que les codes postaux ou les noms de famille. Par exemple, il existe 41 707 codes postaux aux États-Unis d'Amérique. La population que vous utilisez doit être proportionnelle à votre propre base de données. Si le ZIPCode champ de votre base de données inclut des données provenant de l'ensemble des États-Unis d'Amérique, vous pouvez définir votre population comme étant 41 707, même si ZIPCode le champ ne contient pas actuellement 41 707 valeurs

uniques. Si le ZIPCode champ de votre base de données inclut uniquement les données d'un seul État, et n'inclura que les données d'un seul État, vous pouvez définir votre population comme le nombre total de codes postaux dans cet État au lieu de 41 704.

2. Calculez la plage recommandée pour le nombre de collisions attendu

Pour déterminer la longueur de balise appropriée pour un champ donné, vous devez d'abord identifier une plage appropriée pour le nombre de collisions attendu. Le nombre de collisions attendu représente le nombre moyen attendu de valeurs de texte brut uniques correspondant à une balise HMAC particulière. Le nombre attendu de faux positifs pour une valeur unique en texte brut est inférieur d'un au nombre de collisions attendu.

Nous recommandons que le nombre de collisions attendu soit supérieur ou égal à deux, et inférieur à la racine carrée de votre population. Les équations suivantes ne fonctionnent que si votre population possède 16 valeurs uniques ou plus.

$$2 \leq \text{number of collisions} < \sqrt{(\text{Population})}$$

Si le nombre de collisions est inférieur à deux, la balise produira trop peu de faux positifs. Nous recommandons deux comme nombre minimum de collisions attendues, car cela signifie qu'en moyenne, chaque valeur unique du champ générera au moins un faux positif en étant mappée à une autre valeur unique.

3. Calculez la plage recommandée pour les longueurs de balise

Après avoir identifié le nombre minimum et maximum de collisions attendues, utilisez l'équation suivante pour identifier une plage de longueurs de balise appropriées.

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

Tout d'abord, déterminez la longueur de la balise lorsque le nombre de collisions attendues est égal à deux (le nombre minimum recommandé de collisions attendues).

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

Ensuite, déterminez la longueur de la balise pour laquelle le nombre de collisions attendu est égal à la racine carrée de votre population (le nombre maximum recommandé de collisions attendues).

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

Nous recommandons d'arrondir le résultat produit par cette équation à la longueur de balise la plus courte. Par exemple, si l'équation produit une longueur de balise de 15,6, nous recommandons d'arrondir cette valeur à 15 bits au lieu d'arrondir à 16 bits.

4. Choisissez une longueur de balise

Ces équations identifient uniquement une plage recommandée de longueurs de balise pour votre champ. Nous vous recommandons d'utiliser une balise plus courte pour préserver la sécurité de votre ensemble de données dans la mesure du possible. Cependant, la longueur de la balise que vous utilisez réellement est déterminée par votre modèle de menace. Tenez compte de vos exigences en matière de performances lorsque vous examinez votre modèle de menace afin de déterminer la longueur de balise la mieux adaptée à votre champ.

L'utilisation d'une longueur de balise plus courte réduit les performances des requêtes, tandis que l'utilisation d'une longueur de balise plus longue réduit la sécurité. En général, si votre jeu de données est inégalement [distribué](#) ou si vous créez des balises distinctes à partir de champs [corrélés](#), vous devez utiliser des balises de plus courte longueur afin de minimiser la quantité d'informations révélées sur la distribution de vos ensembles de données.

Si vous examinez votre modèle de menace et que vous déterminez que les informations distinctives révélées concernant la distribution d'un champ ne constituent pas une menace pour votre sécurité globale, vous pouvez choisir d'utiliser une longueur de balise supérieure à la plage recommandée que vous avez calculée. Par exemple, si vous avez calculé la plage de longueurs de balise recommandée pour un champ entre 9 et 16 bits, vous pouvez choisir d'utiliser une longueur de balise de 24 bits pour éviter toute perte de performance.

Choisissez soigneusement la longueur de votre balise. Une fois que vous avez écrit de nouveaux enregistrements avec la balise, vous ne pouvez pas mettre à jour la longueur de la balise.

Exemple

Prenons l'exemple d'une base de données qui a marqué le `unit` champ comme `ENCRYPT_AND_SIGN` dans les [actions cryptographiques](#). Pour configurer une balise standard pour le `unit` champ, nous devons déterminer le nombre attendu de faux positifs et la longueur de la balise pour le `unit` champ.

1. Estimer la population

Après avoir examiné notre modèle de menace et notre solution de base de données actuelle, nous nous attendons à ce que le `unit` champ contienne à terme 100 000 valeurs uniques.

Cela signifie que la population est égale à 100 000 habitants.

2. Calculez la plage recommandée pour le nombre de collisions attendu.

Dans cet exemple, le nombre de collisions attendu doit être compris entre 2 et 316.

$$2 \leq \text{number of collisions} < \sqrt{(\text{Population})}$$

a. $2 \leq \text{number of collisions} < \sqrt{(100,000)}$

b. $2 \leq \text{number of collisions} < 316$

3. Calculez la plage recommandée pour la longueur de balise.

Dans cet exemple, la longueur de la balise doit être comprise entre 9 et 16 bits.

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

- a. Calculez la longueur de la balise lorsque le nombre de collisions attendu est égal au minimum identifié à l'étape 2.

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

Longueur de la balise = 15,6, soit 15 bits

- b. Calculez la longueur de la balise lorsque le nombre de collisions attendu est égal au maximum identifié à l'étape 2.

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

Longueur de la balise = 8,3, soit 8 bits

4. Déterminez la longueur de balise adaptée à vos exigences de sécurité et de performance.

Pour chaque bit inférieur à 15, le coût de performance et la sécurité doublent.

- 16 bits
 - En moyenne, chaque valeur unique correspond à 1,5 autre unité.
 - Sécurité : deux enregistrements portant la même balise HMAC tronquée ont 66 % de chances d'avoir la même valeur en texte brut.
 - Performance : une requête permet de récupérer 15 enregistrements pour 10 enregistrements que vous avez réellement demandés.
- 14 bits
 - En moyenne, chaque valeur unique correspond à 6,1 autres unités.
 - Sécurité : deux enregistrements portant la même balise HMAC tronquée ont 33 % de chances d'avoir la même valeur en texte brut.
 - Performance : une requête permet de récupérer 30 enregistrements pour 10 enregistrements que vous avez réellement demandés.

Choisir un nom de balise

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Chaque balise est identifiée par un nom de balise unique. Une fois qu'une balise est configurée, le nom de la balise est le nom que vous utilisez lorsque vous interrogez un champ chiffré. Le nom d'une balise peut être le même que celui d'un champ crypté ou d'un [champ virtuel](#), mais il ne peut pas être le même que celui d'un champ non chiffré. Deux balises différentes ne peuvent pas porter le même nom de balise.

Pour des exemples illustrant comment nommer et configurer les balises, consultez la section [Configuration des balises](#).

Nommer une balise standard

Lorsque vous nommez des balises standard, nous recommandons vivement que le nom de votre balise corresponde à la [source de la balise](#) dans la mesure du possible. Cela signifie que le nom de la balise et le nom du champ crypté ou [virtuel](#) à partir duquel votre balise standard est construite sont

identiques. Par exemple, si vous créez une balise standard pour un champ chiffré nommé `LastName`, le nom de votre balise doit également être `LastName`.

Lorsque le nom de votre balise est identique à celui de la source de la balise, vous pouvez omettre la source de la balise dans votre configuration et le SDK de chiffrement de AWS base de données utilisera automatiquement le nom de la balise comme source de balise.

Configuration des balises

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Il existe deux types de balises qui prennent en charge le chiffrement consultable. Les balises standard effectuent des recherches d'égalité. Ils constituent le moyen le plus simple d'implémenter le chiffrement consultable dans votre base de données. Les balises composées combinent des chaînes de texte en clair littérales et des balises standard pour effectuer des requêtes plus complexes.

Les balises sont conçues pour être mises en œuvre dans de nouvelles bases de données non peuplées. Toute balise configurée dans une base de données existante ne mapperà que les nouveaux enregistrements écrits dans la base de données. Les balises sont calculées à partir de la valeur en texte brut d'un champ. Une fois le champ crypté, il n'est plus possible pour la balise de mapper les données existantes. Une fois que vous avez écrit de nouveaux enregistrements avec une balise, vous ne pouvez pas mettre à jour la configuration de la balise. Cependant, vous pouvez ajouter de nouvelles balises pour les nouveaux champs que vous ajoutez à votre enregistrement.

Après avoir déterminé vos modèles d'accès, la configuration des balises doit être la deuxième étape de l'implémentation de votre base de données. Ensuite, après avoir configuré toutes vos balises, vous devez créer un jeu de [clés AWS KMS hiérarchique](#), définir la version de la balise, [configurer un index secondaire pour chaque balise](#), définir vos [actions cryptographiques](#) et configurer votre base de données et le client du SDK AWS Database Encryption. Pour plus d'informations, consultez la section [Utilisation de balises](#).

Pour faciliter la définition de la version des balises, nous vous recommandons de créer des listes pour les balises standard et composées. Ajoutez chaque balise que vous créez à la liste de balises standard ou composée correspondante au fur et à mesure de sa configuration.

Rubriques

- [Configuration des balises standard](#)
- [Configuration de balises composées](#)
- [Exemples de configuration](#)

Configuration des balises standard

Les [balises standard](#) constituent le moyen le plus simple d'implémenter un chiffrement consultable dans votre base de données. Ils ne peuvent effectuer des recherches d'égalité que pour un seul champ crypté ou virtuel.

Exemple de syntaxe de configuration

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()?,
```

Pour configurer une balise standard, fournissez les valeurs suivantes.

Nom de la balise

Le nom que vous utilisez lorsque vous interrogez un champ chiffré.

Le nom d'une balise peut être le même que celui d'un champ crypté ou d'un champ virtuel, mais il ne peut pas être le même que celui d'un champ non chiffré. Nous vous recommandons vivement d'utiliser le nom du champ crypté ou du [champ virtuel](#) à partir duquel votre balise standard est construite dans la mesure du possible. Deux balises différentes ne peuvent pas porter le même nom de balise. Pour obtenir de l'aide pour déterminer le meilleur nom de balise pour votre implémentation, consultez [Choisir un nom de balise](#).

Longueur de la balise

Nombre de bits de la valeur de hachage de la balise conservés après la troncature.

La longueur de la balise détermine le nombre moyen de faux positifs produits par une balise donnée. Pour plus d'informations et pour vous aider à déterminer la longueur de balise appropriée pour votre implémentation, consultez la section [Détermination de la longueur de balise](#).

Source de balise (facultatif)

Champ à partir duquel une balise standard est construite.

La source de la balise doit être un nom de champ ou un index faisant référence à la valeur d'un champ imbriqué. Lorsque le nom de votre balise est identique à celui de la source de la balise, vous pouvez omettre la source de la balise dans votre configuration et le SDK de chiffrement de AWS base de données utilisera automatiquement le nom de la balise comme source de balise.

Création d'un champ virtuel

Pour créer un [champ virtuel](#), vous devez fournir un nom pour le champ virtuel et une liste des champs sources. L'ordre dans lequel vous ajoutez les champs source à la liste des pièces virtuelles détermine l'ordre dans lequel ils sont concaténés pour créer le champ virtuel. L'exemple suivant concatène deux champs source dans leur intégralité pour créer un champ virtuel.

Note

Nous vous recommandons de vérifier que vos champs virtuels produisent le résultat attendu avant de remplir votre base de données. Pour plus d'informations, consultez la section [Tester les sorties des balises](#).

Java

Voir l'exemple de code complet : [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

Voir l'exemple de code complet : [VirtualBeaconSearchableEncryptionExample.cs](#)

```
var virtualPartList = new List<VirtualPart> { sourceField1, sourceField2 };

var virtualFieldName = new VirtualField
{
    Name = "virtualFieldName",
    Parts = virtualPartList
};

var virtualFieldList = new List<VirtualField> { virtualFieldName };
```

Rust

Voir l'exemple de code complet : [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];
```

Pour créer un champ virtuel avec un segment spécifique d'un champ source, vous devez définir cette transformation avant d'ajouter le champ source à votre liste de pièces virtuelles.

Considérations relatives à la sécurité des champs virtuels

Les balises ne modifient pas l'état chiffré du champ. Cependant, lorsque vous utilisez des balises, il existe un compromis inhérent entre l'efficacité de vos requêtes et la quantité d'informations révélées sur la distribution de vos données. La façon dont vous configurez votre balise détermine le niveau de sécurité préservé par cette balise.

Évitez de créer un champ virtuel dont les champs source se chevauchent avec les balises standard existantes. La création de champs virtuels incluant un champ source qui a déjà été utilisé pour créer une balise standard peut réduire le niveau de sécurité des deux balises. L'ampleur de la réduction de la sécurité dépend du niveau d'entropie ajouté par les champs source supplémentaires. Le niveau d'entropie est déterminé par la distribution de valeurs uniques dans le champ source supplémentaire et par le nombre de bits que le champ source supplémentaire contribue à la taille globale du champ virtuel.

Vous pouvez utiliser la population et la [longueur des balises](#) pour déterminer si les champs source d'un champ virtuel préservent la sécurité de votre ensemble de données. La population est le nombre attendu de valeurs uniques dans un champ. Votre population n'a pas besoin d'être précise. Pour obtenir de l'aide pour estimer la population d'un champ, voir [Estimer la population](#).

Examinez l'exemple suivant lorsque vous examinez la sécurité de vos champs virtuels.

- Beacon1 est construit à partir de FieldA FieldAa une population supérieure à $2^{(\text{longueur de Beacon1})}$.
- Beacon2 est construit à partir de VirtualField, qui est construit à partir de FieldA, FieldBFieldC, et FieldD Ensemble, FieldBFieldC, et FieldD ont une population supérieure à 2^N

Beacon2 préserve la sécurité de Beacon1 et de Beacon2 si les affirmations suivantes sont vraies :

$$N \geq (\text{Beacon1 length})/2$$

and

$$N \geq (\text{Beacon2 length})/2$$

Définition des styles de balises

Les balises standard peuvent être utilisées pour effectuer des recherches d'égalité pour un champ crypté ou virtuel. Ils peuvent également être utilisés pour créer des balises composées afin d'effectuer des opérations de base de données plus complexes. Pour vous aider à organiser et à gérer les balises standard, le SDK AWS de chiffrement de base de données fournit les styles de balises facultatifs suivants qui définissent l'utilisation prévue d'une balise standard.

Note

Pour définir les styles de balises, vous devez utiliser la version 3.2 ou ultérieure du SDK AWS Database Encryption. Déployez la nouvelle version sur tous les lecteurs avant d'ajouter des styles de balises à vos configurations de balises.

PartOnly

Une balise standard définie comme ne `PartOnly` peut être utilisée que pour définir une [partie cryptée](#) d'une balise composée. Vous ne pouvez pas interroger directement une balise `PartOnly` standard.

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

Par défaut, chaque balise standard génère une clé HMAC unique pour le calcul de la balise. Par conséquent, vous ne pouvez pas effectuer de recherche d'égalité sur les champs chiffrés à partir de deux balises standard distinctes. Une balise standard définie comme Shared utilise la clé HMAC d'une autre balise standard pour ses calculs.

Par exemple, si vous devez comparer des beacon1 champs à des beacon2 champs, beacon2 définissez-les comme une Shared balise qui utilise la clé HMAC de beacon1 pour ses calculs.

Note

Tenez compte de vos besoins en matière de sécurité et de performances avant de configurer des Shared balises. Sharedles balises peuvent augmenter la quantité d'informations statistiques pouvant être identifiées sur la distribution de votre ensemble de données. Par exemple, ils peuvent révéler quels champs partagés contiennent la même valeur en texte brut.

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

Par défaut, si la valeur d'un champ est un ensemble, le SDK AWS de chiffrement de base de données calcule une balise standard unique pour l'ensemble. Par conséquent, vous ne pouvez pas exécuter la requête `CONTAINS(a, :value)` où se `a` trouve un champ crypté. Une balise standard définie comme `AsSet` calcule des valeurs de balise standard individuelles pour chaque élément individuel de l'ensemble et stocke la valeur de balise dans l'élément sous forme d'ensemble. Cela permet au SDK AWS de chiffrement de base de données d'exécuter la requête `CONTAINS(a, :value)`.

Pour définir une balise `AsSet` standard, les éléments de l'ensemble doivent appartenir à la même population afin qu'ils puissent tous utiliser la même [longueur de balise](#). L'ensemble de balises

peut comporter moins d'éléments que le jeu de texte en clair en cas de collisions lors du calcul des valeurs des balises.

Note

Tenez compte de vos besoins en matière de sécurité et de performances avant de configurer des AsSet balises. AsSet les balises peuvent augmenter la quantité d'informations statistiques pouvant être identifiées sur la distribution de votre ensemble de données. Par exemple, ils peuvent révéler la taille de l'ensemble de texte en clair.

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

Une balise standard définie comme SharedSet combine les AsSet fonctions Shared et afin que vous puissiez effectuer des recherches d'égalité sur les valeurs cryptées d'un ensemble et d'un champ. Cela permet au SDK AWS de chiffrement de base de données d'exécuter la requête CONTAINS(*a*, *b*) lorsqu'il *a* s'agit d'un ensemble chiffré et *b* d'un champ chiffré.

Note

Tenez compte de vos besoins en matière de sécurité et de performances avant de configurer des Shared balises. SharedSet les balises peuvent augmenter la quantité d'informations statistiques pouvant être identifiées sur la distribution de votre ensemble de données. Par exemple, ils peuvent révéler la taille de l'ensemble de texte en clair ou les champs partagés contenant la même valeur en texte brut.

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()?
```

Configuration de balises composées

Les balises composées combinent des chaînes de texte brut littérales et des balises standard pour effectuer des opérations de base de données complexes, telles que l'interrogation de deux types d'enregistrement différents à partir d'un seul index ou l'interrogation d'une combinaison de champs à l'aide d'une clé de tri. Les balises composées peuvent être construites à partir de ENCRYPT_AND_SIGNSIGN_ONLY, et de SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT champs. Vous devez créer une balise standard pour chaque champ chiffré inclus dans la balise composée.

Note

Nous vous recommandons de vérifier que vos balises composées produisent le résultat attendu avant de remplir votre base de données. Pour plus d'informations, consultez la section [Tester les sorties des balises](#).

Exemple de syntaxe de configuration

Java

Configuration de balise composée

L'exemple suivant définit des listes de pièces chiffrées et signées localement dans la configuration des balises composées.

```

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

```

Définition de la version de la balise

L'exemple suivant définit les listes de pièces chiffrées et signées de manière globale dans la version balise. Pour plus d'informations sur la définition de la version des balises, consultez la section [Utilisation des balises](#).

```

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

Voir l'exemple de code complet : [BeaconConfig.cs](#)

Configuration de balise composée

L'exemple suivant définit des listes de pièces chiffrées et signées localement dans la configuration des balises composées.

```

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

```

Définition de la version de la balise

L'exemple suivant définit les listes de pièces chiffrées et signées de manière globale dans la version balise. Pour plus d'informations sur la définition de la version des balises, consultez la section [Utilisation des balises](#).

```

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

Voir l'exemple de code complet : [beacon_config.rs](#)

Configuration de balise composée

L'exemple suivant définit des listes de pièces chiffrées et signées localement dans la configuration des balises composées.

```
let compound_beacon_list = vec![
  CompoundBeacon::builder()
    .name("compound_beacon_name")
    .split(".")
    .encrypted(encrypted_parts_list)
    .signed(signed_parts_list)
    .constructors(constructor_list)
    .build()?
```

Définition de la version de la balise

L'exemple suivant définit les listes de pièces chiffrées et signées de manière globale dans la version balise. Pour plus d'informations sur la définition de la version des balises, consultez la section [Utilisation des balises](#).

```
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];
```

Vous pouvez définir vos [parties chiffrées](#) et vos [parties signées](#) dans des listes définies localement ou globalement. Nous vous recommandons de définir vos parties cryptées et signées dans une liste globale dans la [version balise](#) dans la mesure du possible. En définissant les parties chiffrées et signées de manière globale, vous pouvez définir chaque partie une seule fois, puis les réutiliser dans plusieurs configurations de balises composées. Si vous n'avez l'intention d'utiliser une pièce chiffrée ou signée qu'une seule fois, vous pouvez la définir dans une liste locale dans la configuration

des balises composées. Vous pouvez référencer des pièces locales et globales dans votre [liste de constructeurs](#).

Si vous définissez vos listes de pièces chiffrées et signées de manière globale, vous devez fournir une liste de composants du constructeur identifiant toutes les manières possibles dont la balise composée peut assembler les champs dans votre configuration de balise composée.

Note

Pour définir des listes de pièces chiffrées et signées de manière globale, vous devez utiliser la version 3.2 ou ultérieure du SDK AWS Database Encryption. Déployez la nouvelle version sur tous les lecteurs avant de définir de nouvelles pièces de manière globale.

Vous ne pouvez pas mettre à jour les configurations de balises existantes pour définir des listes de pièces chiffrées et signées dans le monde entier.

Pour configurer une balise composée, fournissez les valeurs suivantes.

Nom de la balise

Le nom que vous utilisez lorsque vous interrogez un champ chiffré.

Le nom d'une balise peut être le même que celui d'un champ crypté ou d'un champ virtuel, mais il ne peut pas être le même que celui d'un champ non chiffré. Deux balises ne peuvent pas porter le même nom de balise. Pour obtenir de l'aide pour déterminer le meilleur nom de balise pour votre implémentation, consultez [Choisir un nom de balise](#).

Caractère divisé

Le caractère utilisé pour séparer les parties qui composent votre balise composée.

Le caractère divisé ne peut apparaître dans les valeurs en texte brut d'aucun des champs à partir desquels la balise composée est construite.

Liste de pièces cryptée

Identifie les ENCRYPT_AND_SIGN champs inclus dans la balise composée.

Chaque partie doit inclure un nom et un préfixe. Le nom de pièce doit être le nom de la balise standard construite à partir du champ crypté. Le préfixe peut être n'importe quelle chaîne, mais il doit être unique. Une pièce chiffrée ne peut pas avoir le même préfixe qu'une pièce signée. Nous

recommandons d'utiliser une valeur courte qui distingue la pièce des autres parties desservies par la balise composée.

Nous vous recommandons de définir vos parties cryptées de manière globale dans la mesure du possible. Vous pouvez envisager de définir une partie chiffrée localement si vous avez l'intention de ne l'utiliser que dans une seule balise composée. Une partie cryptée définie localement ne peut pas avoir le même préfixe ou le même nom qu'une partie cryptée définie globalement.

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()?
];
```

Liste de pièces signée

Identifie les champs signés inclus dans la balise composée.

Note

Les pièces signées sont facultatives. Vous pouvez configurer une balise composée qui ne fait référence à aucune pièce signée.

Chaque partie doit inclure un nom, une source et un préfixe. La source est le `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` champ `SIGN_ONLY` ou que l'article identifie. La source doit être un nom de champ ou un index faisant référence à la valeur d'un champ imbriqué. Si le nom de votre pièce identifie la source, vous pouvez omettre la source et le SDK AWS de chiffrement de base de données utilisera automatiquement le nom comme source. Nous recommandons de spécifier la source comme nom de pièce dans la mesure du possible. Le préfixe peut être n'importe quelle chaîne, mais il doit être unique. Une pièce signée ne peut pas avoir le même préfixe qu'une pièce chiffrée. Nous recommandons d'utiliser une valeur courte qui distingue la pièce des autres parties desservies par la balise composée.

Nous vous recommandons de définir vos pièces signées de manière globale dans la mesure du possible. Vous pouvez envisager de définir une pièce signée localement si vous avez l'intention de ne l'utiliser que dans une seule balise composée. Une pièce signée définie localement ne peut pas avoir le même préfixe ou le même nom qu'une pièce signée définie globalement.

Java

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

C# / .NET

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

Rust

```
let signed_parts_list = vec![
    SignedPart::builder()
        .name("signed_field_name_1")
        .prefix("S-")
        .build()?,
    SignedPart::builder()
        .name("signed_field_name_2")
        .prefix("SF-")
        .build()?,
];
```

Liste des constructeurs

Identifie les constructeurs qui définissent les différentes manières dont les pièces cryptées et signées peuvent être assemblées par la balise composée. Vous pouvez référencer des pièces locales et globales dans votre liste de constructeurs.

Si vous créez votre balise composée à partir de pièces chiffrées et signées définies globalement, vous devez fournir une liste de constructeurs.

Si vous n'utilisez aucune pièce chiffrée ou signée définie globalement pour construire votre balise composée, la liste des constructeurs est facultative. Si vous ne spécifiez pas de liste de constructeurs, le SDK AWS Database Encryption assemble la balise composée avec le constructeur par défaut suivant.

- Toutes les pièces signées dans l'ordre dans lequel elles ont été ajoutées à la liste des pièces signées
- Toutes les pièces cryptées dans l'ordre dans lequel elles ont été ajoutées à la liste des pièces cryptées
- Toutes les pièces sont requises

Constructeurs

Chaque constructeur est une liste ordonnée de pièces du constructeur qui définit une manière dont la balise composée peut être assemblée. Les pièces du constructeur sont assemblées dans l'ordre dans lequel elles sont ajoutées à la liste, chaque partie étant séparée par le caractère divisé spécifié.

Chaque partie du constructeur nomme une partie chiffrée ou une partie signée, et définit si cette partie est obligatoire ou facultative dans le constructeur. Par exemple, si vous souhaitez

interroger une balise composée sur `Field1`, `Field1.Field2`, et `Field1.Field2.Field3`, marquer `Field2` et `Field3` comme facultatif et créer un constructeur.

Chaque constructeur doit avoir au moins une pièce requise. Nous vous recommandons de rendre obligatoire la première partie de chaque constructeur afin que vous puissiez utiliser l'opérateur `BEGINS_WITH` dans vos requêtes.

Un constructeur réussit si toutes ses pièces requises sont présentes dans l'enregistrement. Lorsque vous écrivez un nouvel enregistrement, la balise composée utilise la liste des constructeurs pour déterminer si la balise peut être assemblée à partir des valeurs fournies. Il tente d'assembler la balise dans l'ordre dans lequel les constructeurs ont été ajoutés à la liste des constructeurs, et il utilise le premier constructeur qui réussit. Si aucun constructeur ne réussit, la balise n'est pas écrite dans l'enregistrement.

Tous les lecteurs et rédacteurs doivent spécifier le même ordre de constructeurs pour s'assurer que les résultats de leurs requêtes sont corrects.

Utilisez les procédures suivantes pour spécifier votre propre liste de constructeurs.

1. Créez une partie constructeur pour chaque pièce chiffrée et pièce signée afin de définir si cette partie est requise ou non.

Le nom de la partie constructeur doit être le nom de la balise standard ou du champ signé qu'il représente.

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. Créez un constructeur pour chaque manière possible d'assembler la balise composée à l'aide des pièces du constructeur que vous avez créées à l'étape 1.

Par exemple, si vous souhaitez effectuer une requête sur `Field1.Field2.Field3` et `Field4.Field2.Field3`, vous devez créer deux constructeurs. `Field1` et `Field4` peuvent tous deux être requis car ils sont définis dans deux constructeurs distincts.

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_constructor_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_constructor_part.clone(),
        field1_constructor_part,
    ])
    .build()?;
```

3. Créez une liste de constructeurs qui inclut tous les constructeurs que vous avez créés à l'étape 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. Spécifiez le `constructorList` moment où vous créez votre balise composée.

Exemples de configuration

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Les exemples suivants montrent comment configurer des balises standard et composées. Les configurations suivantes ne fournissent pas de longueur de balise. Pour obtenir de l'aide pour déterminer la longueur de balise adaptée à votre configuration, voir [Choisir une longueur de balise](#).

Pour voir des exemples de code complets qui montrent comment configurer et utiliser des balises, consultez les exemples de chiffrement consultables [Java](#), [.NET](#) et [Rust](#) dans le référentiel `aws-database-encryption-sdk-dynamodb` sur GitHub.

Rubriques

- [Balises standard](#)
- [Balises composées](#)

Balises standard

Si vous souhaitez interroger le `inspector_id_last4` champ pour obtenir des correspondances exactes, créez une balise standard en utilisant la configuration suivante.

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];
```

Balises composées

Si vous souhaitez interroger la UnitInspection base de données sur `inspector_id_last4` et `inspector_id_last4.unit`, créez une balise composée avec la configuration suivante. Cette balise composée ne nécessite que des [parties cryptées](#).

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()?];
```

Utilisation de balises

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Les balises vous permettent de rechercher des enregistrements cryptés sans déchiffrer l'intégralité de la base de données interrogée. Les balises sont conçues pour être implémentées dans de

nouvelles bases de données non peuplées. Toute balise configurée dans une base de données existante ne mapperait que les nouveaux enregistrements écrits dans la base de données. Les balises sont calculées à partir de la valeur en texte brut d'un champ. Une fois le champ crypté, la balise ne peut plus cartographier les données existantes. Une fois que vous avez écrit de nouveaux enregistrements avec une balise, vous ne pouvez pas mettre à jour la configuration de la balise. Cependant, vous pouvez ajouter de nouvelles balises pour les nouveaux champs que vous ajoutez à votre enregistrement.

Après avoir configuré vos balises, vous devez suivre les étapes suivantes avant de commencer à remplir votre base de données et à exécuter des requêtes sur vos balises.

1. Création d'un porte-clés AWS KMS hiérarchique

Pour utiliser le chiffrement consultable, vous devez utiliser le trousseau de [clés AWS KMS hiérarchique](#) pour générer, chiffrer et déchiffrer les [clés de données](#) utilisées pour protéger vos enregistrements.

Après avoir configuré vos balises, assemblez les [prérequis relatifs au jeu de clés hiérarchique](#) et [créez votre](#) trousseau de clés hiérarchique.

Pour plus de détails sur les raisons pour lesquelles le trousseau de clés hiérarchique est requis, voir [Utilisation du trousseau de clés hiérarchique pour le chiffrement consultable](#).

2.

Définissez la version de la balise

Spécifiez votre `keyStorekeySource`, une liste de toutes les balises standard que vous avez configurées, une liste de toutes les balises composées que vous avez configurées, une liste de parties cryptées, une liste de parties signées et une version de balise. Vous devez spécifier 1 la version de la balise. Pour obtenir des conseils sur la définition de votre `keySource`, voir [Définition de la source clé de votre balise](#).

L'exemple Java suivant définit la version de balise pour une base de données à locataire unique. Pour obtenir de l'aide sur la définition de la version de balise pour une base de données mutualisée, consultez la section [Chiffrement consultable pour les bases de données mutualisées](#).

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. Configuration des index secondaires

Après avoir [configuré vos balises](#), vous devez configurer un index secondaire qui reflète chaque balise avant de pouvoir effectuer une recherche dans les champs chiffrés. Pour de plus amples informations, veuillez consulter [Configuration des index secondaires avec des balises](#).

4. Définissez vos actions [cryptographiques](#)

Tous les champs utilisés pour construire une balise standard doivent être marqués `ENCRYPT_AND_SIGN`. Tous les autres champs utilisés pour créer des balises doivent être marqués `SIGN_ONLY` ou `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

5. Configuration d'un client SDK AWS de chiffrement de base de données

Pour configurer un client du SDK AWS de chiffrement de base de données qui protège les éléments de table de votre table DynamoDB, [consultez la section Bibliothèque de chiffrement côté client Java](#) pour DynamoDB.

Interrogation de balises

Le type de balise que vous configurez détermine le type de requêtes que vous pouvez effectuer. Les balises standard utilisent des expressions de filtre pour effectuer des recherches d'égalité. Les balises composées combinent des chaînes de texte en clair littérales et des balises standard pour

effectuer des requêtes complexes. Lorsque vous interrogez des données chiffrées, vous recherchez le nom de la balise.

Vous ne pouvez pas comparer les valeurs de deux balises standard, même si elles contiennent le même texte brut sous-jacent. Les deux balises standard produiront deux balises HMAC différentes pour les mêmes valeurs en texte brut. Par conséquent, les balises standard ne peuvent pas effectuer les requêtes suivantes.

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

Les balises composées peuvent effectuer les requêtes suivantes.

- BEGINS_WITH(*a*), où *a* reflète la valeur totale du champ par lequel commence la balise composée assemblée. Vous ne pouvez pas utiliser l'opérateur BEGINS_WITH pour identifier une valeur commençant par une sous-chaîne particulière. Cependant, vous pouvez utiliser « BEGINS_WITH(*S_*) where » *S_* reflète le préfixe d'une pièce par laquelle commence la balise composée assemblée.
- CONTAINS(*a*), où *a* reflète la valeur totale d'un champ que contient la balise composée assemblée. Vous ne pouvez pas utiliser l'opérateur CONTAINS pour identifier un enregistrement contenant une sous-chaîne particulière ou une valeur au sein d'un ensemble.

Par exemple, vous ne pouvez pas exécuter une CONTAINS(*path*, "*a*") requête *a* qui reflète la valeur d'un ensemble.

- Vous pouvez comparer les [parties signées](#) de balises composées. Lorsque vous comparez des parties signées, vous pouvez éventuellement ajouter le préfixe d'une [partie chiffrée](#) à une ou plusieurs parties signées, mais vous ne pouvez inclure la valeur d'un champ crypté dans aucune requête.

Par exemple, vous pouvez comparer des pièces signées et effectuer une requête sur *signedField1* = *signedField2* ou *value* IN (*signedField1*, *signedField2*, ...).

Vous pouvez également comparer les parties signées et le préfixe d'une pièce chiffrée en effectuant une requête sur *signedField1.A_* = *signedField2.B_*.

- *field* BETWEEN *a* AND *b*, où *a* et *b* sont des pièces signées. Vous pouvez éventuellement ajouter le préfixe d'une partie chiffrée à une ou plusieurs parties signées, mais vous ne pouvez inclure la valeur d'un champ crypté dans aucune requête.

Vous devez inclure le préfixe pour chaque partie que vous incluez dans une requête sur une balise composée. Par exemple, si vous avez créé une balise composée `compoundBeacon`, à partir de deux champs, `encryptedField` et `signedField`, vous devez inclure les préfixes configurés pour ces deux parties lorsque vous interrogez la balise.

```
compoundBeacon = E_encryptedFieldValue.S_signedFieldValue
```

Chiffrement consultable pour les bases de données mutualisées

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Pour implémenter le chiffrement consultable dans votre base de données, vous devez utiliser un trousseau de [clés AWS KMS hiérarchique](#). Le trousseau de clés AWS KMS hiérarchique génère, chiffre et déchiffre les clés de données utilisées pour protéger vos dossiers. Il crée également la clé de balise utilisée pour générer des balises. Lorsque vous utilisez le trousseau de clés AWS KMS hiérarchique avec des bases de données à locataires multiples, il existe une clé de branche et une clé de balise distinctes pour chaque locataire. Pour interroger des données chiffrées dans une base de données mutualisée, vous devez identifier les éléments clés de balise utilisés pour générer la balise que vous interrogez. Pour de plus amples informations, veuillez consulter [the section called "Utilisation du trousseau de clés hiérarchique pour un chiffrement consultable"](#).

Lorsque vous définissez la [version de balise](#) pour une base de données mutualisée, spécifiez une liste de toutes les balises standard que vous avez configurées, une liste de toutes les balises composées que vous avez configurées, une version de balise et un `keySource`. Vous devez [définir votre source de clé de balise](#) comme une `MultiKeyStore` durée de vie du cache de clé de balise local et inclure une taille de cache maximale pour le cache de clé de balise local. `keyFieldName`

Si vous avez configuré des [balises signées](#), elles doivent être incluses dans votre `compoundBeaconList`. Les balises signées sont un type de balise composée qui indexe

et exécute des requêtes complexes sur des SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT champs SIGN_ONLY et des champs.

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](#) Définit le nom du champ qui stocke la clé `branch-key-id` associée à la balise utilisée pour générer des balises pour un locataire donné.

Lorsque vous écrivez de nouveaux enregistrements dans votre base de données, la clé `branch-key-id` identifiant la clé de balise utilisée pour générer des balises pour cet enregistrement est stockée dans ce champ.

Par défaut, il `keyField` s'agit d'un champ conceptuel qui n'est pas explicitement stocké dans votre base de données. Le SDK AWS de chiffrement de base de données identifie la [clé `branch-key-id` de données chiffrée contenue](#) dans la [description du matériel](#) et stocke la valeur dans le concept afin que vous `keyField` puissiez la référencer dans vos balises composées et vos balises [signées](#). Puisque la description du matériau est signée, le concept `keyField` est considéré comme une partie signée.

Vous pouvez également inclure le `keyField` dans vos actions cryptographiques sous forme de `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` champ `SIGN_ONLY` ou pour le stocker explicitement dans votre base de données. Dans ce cas, vous devez l'inclure manuellement `keyField` chaque fois que vous écrivez un enregistrement `branch-key-id` dans votre base de données.

Interrogation de balises dans une base de données mutualisée

Pour interroger une balise, vous devez l'inclure `keyField` dans votre requête afin d'identifier les éléments clés de balise appropriés nécessaires pour recalculer la balise. Vous devez spécifier le code `branch-key-id` associé à la clé de balise utilisée pour générer les balises d'un enregistrement. Vous ne pouvez pas spécifier le [nom convivial](#) qui identifie un locataire `branch-key-id` dans le fournisseur d'ID de clé de succursale. Vous pouvez les inclure `keyField` dans vos requêtes de différentes manières.

Balises composées

Que vous les stockiez explicitement ou non `keyField` dans vos dossiers, vous pouvez les inclure `keyField` directement dans vos balises composées sous forme de pièce signée. La pièce `keyField` signée doit être requise.

Par exemple, si vous souhaitez créer une balise composée `compoundBeacon`, à partir de deux champs `signedField`, `encryptedField` vous devez également l'inclure `keyField` sous forme de pièce signée. Cela vous permet d'effectuer la requête suivante sur `compoundBeacon`.

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

Balises signées

Le SDK AWS Database Encryption utilise des balises standard et composées pour fournir des solutions de chiffrement consultables. Ces balises doivent inclure au moins un champ crypté. Cependant, le SDK AWS Database Encryption prend également en charge les [balises signées](#) qui peuvent être entièrement configurées à partir de texte brut `SIGN_ONLY` et de champs `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`

Les balises signées peuvent être construites à partir d'une seule pièce. Que vous le stockiez explicitement `keyField` dans vos dossiers ou non, vous pouvez créer une balise signée à partir de celle-ci `keyField` et l'utiliser pour créer des requêtes composées combinant une requête sur

la balise `keyField` signée avec une requête sur l'une de vos autres balises. Par exemple, vous pouvez exécuter la requête suivante.

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

Pour obtenir de l'aide sur la configuration des balises signées, voir [Création de balises signées](#)

Interrogez directement sur le **keyField**

Si vous l'avez spécifié `keyField` dans vos actions cryptographiques et que vous stockez explicitement le champ dans votre enregistrement, vous pouvez créer une requête composée qui combine une requête sur votre balise avec une requête sur le `keyField`. Vous pouvez choisir d'effectuer une requête directement sur le `keyField` si vous souhaitez interroger une balise standard. Par exemple, vous pouvez exécuter la requête suivante.

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

AWS SDK de chiffrement de base de données pour DynamoDB

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

[Le SDK AWS Database Encryption pour DynamoDB est une bibliothèque logicielle qui vous permet d'inclure le chiffrement côté client dans votre conception Amazon DynamoDB.](#) Le SDK AWS de chiffrement de base de données pour DynamoDB fournit un chiffrement au niveau des attributs et vous permet de spécifier les éléments à chiffrer et ceux à inclure dans les signatures afin de garantir l'authenticité de vos données. Le chiffrement de vos données sensibles en transit et au repos permet de garantir que vos données en texte brut ne sont pas accessibles à des tiers, y compris AWS.

Note

Le SDK AWS de chiffrement de base de données ne prend pas en charge partiQL.

Dans DynamoDB, [une table est un ensemble d'](#)éléments. Chaque élément est une collection d'attributs. Chaque attribut a un nom et une valeur. Le SDK AWS de chiffrement de base de données pour DynamoDB chiffre les valeurs des attributs. Puis, il calcule une signature sur les attributs. Vous spécifiez les valeurs d'attribut à chiffrer et celles à inclure dans la signature lors des actions [cryptographiques](#).

Les rubriques de ce chapitre fournissent une vue d'ensemble du SDK de chiffrement de AWS base de données pour DynamoDB, notamment les champs chiffrés, des conseils sur l'installation et la configuration du client, ainsi que des exemples Java pour vous aider à démarrer.

Rubriques

- [Chiffrement côté client et côté serveur](#)
- [Quels sont les champs chiffrés et signés ?](#)
- [Chiffrement consultable dans DynamoDB](#)
- [Mettre à jour votre modèle de données](#)

- [AWS SDK de chiffrement de base de données pour DynamoDB, langages de programmation disponibles](#)
- [Ancien client de chiffrement DynamoDB](#)

Chiffrement côté client et côté serveur

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Le SDK AWS de chiffrement de base de données pour DynamoDB prend en charge le chiffrement côté client, dans le cadre duquel vous cryptez les données de votre table avant de les envoyer à votre base de données. DynamoDB fournit toutefois une fonctionnalité de chiffrement au repos côté serveur qui chiffre de manière transparente votre table lorsqu'elle est conservée sur le disque et la déchiffre lorsque vous accédez à la table.

Les outils que vous choisissez dépendent de la sensibilité de vos données et des exigences de sécurité de votre application. Vous pouvez utiliser à la fois le SDK AWS de chiffrement de base de données pour DynamoDB et le chiffrement au repos. Lorsque vous envoyez des éléments chiffrés et signés à DynamoDB, DynamoDB ne les reconnaît pas comme étant protégés. Il détecte simplement les éléments de table classiques avec ses valeurs d'attribut binaires.

Chiffrement côté serveur au repos

DynamoDB [prend en charge le chiffrement au](#) repos, une fonctionnalité de chiffrement côté serveur dans laquelle DynamoDB chiffre de manière transparente vos tables pour vous lorsque celles-ci sont conservées sur le disque, et les déchiffre lorsque vous accédez aux données des tables.

Lorsque vous utilisez un AWS SDK pour interagir avec DynamoDB, par défaut, vos données sont chiffrées en transit via une connexion HTTPS, déchiffrées au point de terminaison DynamoDB, puis rechiffrées avant d'être stockées dans DynamoDB.

- Chiffrement par défaut. DynamoDB chiffre et déchiffre de manière transparente toutes les tables lorsqu'elles sont écrites. Aucune option ne permet d'activer ou de désactiver le chiffrement au repos.
- DynamoDB crée et gère les clés cryptographiques. La clé unique de chaque table est protégée par un code [AWS KMS key](#) qui ne laisse jamais [AWS Key Management Service](#) (AWS KMS) non chiffré.

Par défaut, DynamoDB utilise une clé intégrée [Clé détenue par AWS](#) au service DynamoDB, mais vous pouvez choisir [Clé gérée par AWS](#) une clé [ou une clé gérée par le client dans votre compte pour protéger certaines ou](#) toutes vos tables.

- Toutes les données de la table sont cryptées sur le disque. [Lorsqu'une table chiffrée est enregistrée sur disque, DynamoDB chiffre toutes les données de la table, y compris la clé primaire et les index secondaires locaux et globaux.](#) Si votre table a une clé de tri, certaines clés de tri qui marquent les limites de plage sont stockées en texte brut dans les métadonnées de la table.
- Les objets liés aux tables sont également chiffrés. Le chiffrement au repos protège les flux [DynamoDB, les tables globales et](#) les sauvegardes chaque fois qu'ils sont écrits sur un support durable.
- Vos éléments sont déchiffrés lorsque vous y accédez. Lorsque vous accédez à la table, DynamoDB déchiffre la partie de la table qui inclut votre élément cible et vous renvoie l'élément en texte brut.

AWS SDK de chiffrement de base de données pour DynamoDB

Le chiffrement côté client end-to-end protège vos données, en transit et au repos, depuis leur source jusqu'à leur stockage dans DynamoDB. Vos données en texte brut ne sont jamais exposées à des tiers, y compris AWS. Vous pouvez utiliser le SDK AWS de chiffrement de base de données pour DynamoDB avec les nouvelles tables DynamoDB, ou vous pouvez migrer vos tables Amazon DynamoDB existantes vers la dernière version du SDK de chiffrement de base de données pour DynamoDB. AWS

- Vos données sont protégées en transit et au repos. Il n'est jamais exposé à des tiers, y compris AWS.
- Vous pouvez signer les éléments de vos tables. Vous pouvez demander au SDK AWS de chiffrement de base de données pour DynamoDB de calculer une signature sur tout ou partie d'un élément de table, y compris les attributs de clé primaire. Cette signature vous permet de détecter les modifications non autorisées sur l'élément comme un tout, y compris l'ajout ou la suppression d'attributs, ou le remplacement d'une valeur d'attribut par une autre.
- Vous déterminez comment vos données sont protégées [en sélectionnant un trousseau de clés.](#) Votre trousseau de clés détermine les clés d'encapsulation qui protègent vos clés de données et, en fin de compte, vos données. Utilisez les clés d'emballage les plus sûres et les plus pratiques pour votre tâche.
- Le SDK AWS de chiffrement de base de données pour DynamoDB ne chiffre pas l'intégralité de la table. Vous choisissez les attributs qui sont chiffrés dans vos articles. Le SDK AWS de chiffrement

de base de données pour DynamoDB ne chiffre pas un élément entier. Il ne chiffre pas les noms d'attribut, ou les noms ou valeurs des attributs de clé primaire (clé de partition et clé de tri).

AWS Encryption SDK

Si vous chiffrez des données que vous stockez dans DynamoDB, nous vous recommandons AWS le SDK de chiffrement de base de données pour DynamoDB.

Le kit [AWS Encryption SDK](#) est une bibliothèque de chiffrement côté serveur qui vous aide à chiffrer et déchiffrer les données génériques. Même s'il peut protéger tout type de données, il n'est pas conçu pour fonctionner avec des données structurées, comme les enregistrements de base de données. Contrairement au SDK AWS de chiffrement de base de données pour DynamoDB, AWS Encryption SDK il ne peut pas fournir de contrôle d'intégrité au niveau des éléments et il n'a aucune logique permettant de reconnaître les attributs ou d'empêcher le chiffrement des clés primaires.

Si vous utilisez le AWS Encryption SDK pour chiffrer un élément de votre table, n'oubliez pas qu'il n'est pas compatible avec le SDK de chiffrement de AWS base de données pour DynamoDB. Vous ne pouvez pas chiffrer avec une bibliothèque et déchiffrer avec l'autre.

Quels sont les champs chiffrés et signés ?

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Le SDK AWS Database Encryption pour DynamoDB est une bibliothèque de chiffrement côté client spécialement conçue pour les applications Amazon DynamoDB. Amazon DynamoDB stocke les données [dans](#) des tables, qui sont un ensemble d'éléments. Chaque élément est une collection d'attributs. Chaque attribut a un nom et une valeur. Le SDK AWS de chiffrement de base de données pour DynamoDB chiffre les valeurs des attributs. Puis, il calcule une signature sur les attributs. Vous pouvez spécifier les valeurs d'attribut à chiffrer et celles à inclure dans la signature.

Le chiffrement protège la confidentialité de la valeur d'attribut. La signature assure l'intégrité de tous les attributs signés et de leurs relations entre eux, et fournit l'authentification. Elle vous permet de détecter les modifications non autorisées sur l'élément comme un tout, y compris l'ajout ou la suppression d'attributs, ou le remplacement d'une valeur chiffrée par une autre.

Dans un élément chiffré, certaines données restent en texte brut, notamment le nom de la table, tous les noms d'attributs, les valeurs d'attribut que vous ne chiffrez pas, les noms et valeurs des attributs de la clé primaire (clé de partition et clé de tri) et les types d'attributs. Ne stockez pas les données sensibles dans ces champs.

Pour plus d'informations sur le fonctionnement du SDK AWS de chiffrement de base de données pour DynamoDB, consultez [Fonctionnement du SDK AWS de chiffrement de base de données](#)

Note

[Toutes les mentions d'actions d'attributs dans les rubriques du SDK AWS de chiffrement de base de données pour DynamoDB font référence aux actions cryptographiques.](#)

Rubriques

- [Chiffrement des valeurs d'attribut](#)
- [Signature de l'élément](#)

Chiffrement des valeurs d'attribut

Le SDK AWS de chiffrement de base de données pour DynamoDB chiffre les valeurs (mais pas le nom ou le type d'attribut) des attributs que vous spécifiez. Pour déterminer quelles sont les valeurs d'attribut chiffrées, utilisez les [actions d'attribut](#).

Par exemple, cet élément inclut les attributs `example` et `test`.

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

Si vous chiffrez l'attribut `example`, mais pas l'attribut `test`, les résultats se présentent comme suit. La valeur d'attribut `example` chiffrée est une donnée binaire, et non une chaîne.

```
'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'  
...
```

Les attributs de clé primaire (clé de partition et clé de tri) de chaque élément doivent rester en texte brut car DynamoDB les utilise pour rechercher l'élément dans le tableau. Ils doivent être signés, mais pas chiffrés.

Le SDK AWS de chiffrement de base de données pour DynamoDB identifie les attributs de clé primaire pour vous et garantit que leurs valeurs sont signées, mais pas chiffrées. Et, si vous identifiez votre clé primaire, puis essayez de la chiffrer, le client lève une exception.

Le client enregistre la [description du matériel](#) dans un nouvel attribut (`aws_dbe_head`) qu'il ajoute à l'article. La description du matériel décrit comment l'article a été crypté et signé. Le client utilise ces informations pour vérifier et déchiffrer l'élément. Le champ qui contient la description du matériel n'est pas crypté.

Signature de l'élément

[Après avoir chiffré les valeurs d'attribut spécifiées, le SDK de chiffrement de AWS base de données pour DynamoDB calcule les codes d'authentification des messages basés sur le hachage \(HMACs\) et une signature numérique lors de la canonisation de la description du matériel, du contexte de chiffrement et de chaque champ marqué, ou dans les actions d'attribut. `ENCRYPT_AND_SIGN` `SIGN_ONLY` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`](#) Les signatures ECDSA sont activées par défaut, mais ne sont pas obligatoires. Le client enregistre les signatures HMACs et dans un nouvel attribut (`aws_dbe_foot`) qu'il ajoute à l'élément.

Chiffrement consultable dans DynamoDB

Pour configurer vos tables Amazon DynamoDB pour le chiffrement consultable, vous devez utiliser [AWS KMS le trousseau de clés hiérarchique](#) pour générer, chiffrer et déchiffrer les clés de données utilisées pour protéger vos articles. Vous devez également inclure le [SearchConfig](#) dans la configuration de chiffrement de votre table.

Note

Si vous utilisez la bibliothèque de chiffrement côté client Java pour DynamoDB, vous devez utiliser le SDK de chiffrement de AWS base de données de bas niveau pour l'API DynamoDB pour chiffrer, signer, vérifier et déchiffrer les éléments de votre table. Le client DynamoDB amélioré et les `DynamoDBItemEncryptor` niveaux inférieurs ne prennent pas en charge le chiffrement consultable.

Rubriques

- [Configuration des index secondaires avec des balises](#)
- [Tester les sorties des balises](#)

Configuration des index secondaires avec des balises

Après avoir [configuré vos balises](#), vous devez configurer un index secondaire qui reflète chaque balise avant de pouvoir effectuer une recherche sur les attributs chiffrés.

Lorsque vous configurez une balise standard ou composée, le SDK AWS Database Encryption ajoute le `aws_dbe_b_` préfixe au nom de la balise afin que le serveur puisse facilement identifier les balises. Par exemple, si vous nommez une balise composée `compoundBeacon`, le nom complet de la balise est en fait `aws_dbe_b_compoundBeacon`. Si vous souhaitez configurer [des index secondaires](#) qui incluent une balise standard ou composée, vous devez inclure le `aws_dbe_b_` préfixe lorsque vous identifiez le nom de la balise.

Clés de partition et de tri

Vous ne pouvez pas chiffrer les valeurs des clés primaires. Vos clés de partition et de tri doivent être signées. Les valeurs de votre clé primaire ne peuvent pas être une balise standard ou composée.

Les valeurs de votre clé primaire doivent être `SIGN_ONLY`, sauf si vous spécifiez des `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` attributs, les attributs de partition et de tri doivent également l'être `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

Les valeurs de vos clés primaires peuvent être des balises signées. Si vous avez configuré des balises signées distinctes pour chacune de vos valeurs de clé primaire, vous devez spécifier le nom d'attribut qui identifie la valeur de clé primaire comme le nom de balise signé. Toutefois, le SDK AWS de chiffrement de base de données n'ajoute pas le `aws_dbe_b_` préfixe aux balises signées. Même si vous avez configuré des balises signées distinctes pour les valeurs de votre clé primaire, il vous suffit de spécifier les noms d'attribut pour les valeurs de clé primaire lorsque vous configurez un index secondaire.

Index locaux secondaires

La clé de tri d'un [index secondaire local](#) peut être une balise.

Si vous spécifiez une balise pour la clé de tri, le type doit être `String`. Si vous spécifiez une balise standard ou composée pour la clé de tri, vous devez inclure le `aws_dbe_b_` préfixe lorsque vous

spécifiez le nom de la balise. Si vous spécifiez une balise signée, spécifiez le nom de la balise sans aucun préfixe.

Index secondaires globaux

Les clés de partition et de tri d'un [index secondaire global](#) peuvent toutes deux être des balises.

Si vous spécifiez une balise pour la partition ou la clé de tri, le type doit être String. Si vous spécifiez une balise standard ou composée pour la clé de tri, vous devez inclure le `aws_dbe_b_` préfixe lorsque vous spécifiez le nom de la balise. Si vous spécifiez une balise signée, spécifiez le nom de la balise sans aucun préfixe.

Projections d'attribut

Une [projection](#) est l'ensemble d'attributs copié à partir d'une table dans un index secondaire. Les clés de partition et de tri de la table sont toujours projetées dans l'index. Vous pouvez projeter d'autres attributs en fonction des exigences de requête de votre application. DynamoDB propose trois options différentes pour les projections d'attributs `KEYS_ONLY`, `INCLUDE`, et `ALL`.

Si vous utilisez la projection d'attributs `INCLUDE` pour effectuer une recherche sur une balise, vous devez spécifier le nom de tous les attributs à partir desquels la balise est construite, ainsi que le nom de la balise avec le `aws_dbe_b_` préfixe. Par exemple, si vous avez configuré une balise composée `compoundBeaconfield1`, à partir de `field2field3`, et, vous devez spécifier `aws_dbe_b_compoundBeaconfield1,field2`, et `field3` dans la projection.

Un index secondaire global ne peut utiliser que les attributs explicitement spécifiés dans la projection, mais un index secondaire local peut utiliser n'importe quel attribut.

Tester les sorties des balises

Si vous avez [configuré des balises composées](#) ou créé vos balises à l'aide de [champs virtuels](#), nous vous recommandons de vérifier que ces balises produisent le résultat attendu avant de remplir votre table DynamoDB.

Le SDK AWS Database Encryption fournit le `DynamoDbEncryptionTransforms` service qui vous aide à résoudre les problèmes liés aux champs virtuels et aux sorties de balises composées.

Tester des champs virtuels

L'extrait suivant crée des éléments de test, définit le `DynamoDbEncryptionTransforms` service avec la [configuration de chiffrement des tables DynamoDB](#) et montre comment vérifier que le champ virtuel produit le résultat attendu. `ResolveAttributes`

Java

Voir l'exemple de code complet : [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", "CA");
assert resolveOutput.VirtualFields().equals(vf);
```

C# / .NET

Consultez l'exemple de code complet : [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

Consultez l'exemple de code complet : [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"], "CA");
```

Tester des balises composées

L'extrait suivant crée un élément de test, définit le `DynamoDbEncryptionTransforms` service avec la [configuration de chiffrement des tables DynamoDB](#) et montre comment vérifier que la balise composée produit le résultat attendu. `ResolveAttributes`

Java

Voir l'exemple de code complet : [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

Voir l'exemple de code complet : [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

Voir l'exemple de code complet : [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
```

Mettre à jour votre modèle de données

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

[Lorsque vous configurez le SDK AWS de chiffrement de base de données pour DynamoDB, vous fournissez des actions attributaires.](#) Lors du chiffrement, le SDK AWS de chiffrement de base de données utilise les actions d'attributs pour identifier les attributs à chiffrer et à signer, les attributs à signer (mais pas à chiffrer) et ceux à ignorer. Vous définissez également les [attributs non signés autorisés](#) pour indiquer explicitement au client quels attributs sont exclus des signatures. Lors du

déchiffrement, le SDK AWS de chiffrement de base de données utilise les attributs non signés autorisés que vous avez définis pour identifier les attributs qui ne sont pas inclus dans les signatures. Les actions d'attribut ne sont pas enregistrées dans l'élément chiffré et le SDK AWS de chiffrement de base de données ne met pas automatiquement à jour vos actions d'attribut.

Choisissez soigneusement vos actions d'attribut. En cas de doute, utilisez Chiffrer et signer. Une fois que vous avez utilisé le SDK AWS de chiffrement de base de données pour protéger vos éléments, vous ne pouvez pas modifier un `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` attribut ou un attribut existant `ENCRYPT_AND_SIGN` en `DO_NOTHING`. `SIGN_ONLY` Cependant, vous pouvez effectuer les modifications suivantes en toute sécurité.

- [Ajouter `ENCRYPT_AND_SIGN` de `SIGN_ONLY` nouveaux `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` attributs et](#)
- [Supprimer les attributs existants](#)
- [Remplacer un `ENCRYPT_AND_SIGN` attribut existant par `SIGN_ONLY` ou `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`](#)
- [Modifier un `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` attribut `SIGN_ONLY` ou un existant en `ENCRYPT_AND_SIGN`](#)
- [Ajouter un nouvel `DO_NOTHING` attribut](#)
- [Modifier un `SIGN_ONLY` attribut existant en `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`](#)
- [Modifier un `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` attribut existant en `SIGN_ONLY`](#)

Considérations relatives au chiffrement consultable

Avant de mettre à jour votre modèle de données, réfléchissez bien à l'impact que vos mises à jour peuvent avoir sur les [balises](#) que vous avez créées à partir des attributs. Une fois que vous avez écrit de nouveaux enregistrements avec une balise, vous ne pouvez pas mettre à jour la configuration de la balise. Vous ne pouvez pas mettre à jour les actions d'attribut associées aux attributs que vous avez utilisés pour créer des balises. Si vous supprimez un attribut existant et sa balise associée, vous ne pourrez pas interroger les enregistrements existants à l'aide de cette balise. Vous pouvez créer de nouvelles balises pour les nouveaux champs que vous ajoutez à votre enregistrement, mais vous ne pouvez pas mettre à jour les balises existantes pour inclure le nouveau champ.

Considérations relatives aux `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` attributs

Par défaut, les clés de partition et de tri sont les seuls attributs inclus dans le contexte de chiffrement. Vous pouvez envisager de définir des champs supplémentaires

`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` afin que le fournisseur d'ID de clé de branche pour votre jeu de [clés AWS KMS hiérarchique](#) puisse identifier la clé de branche requise pour le déchiffrement à partir du contexte de chiffrement. Pour plus d'informations, consultez le [fournisseur d'ID de clé de branche](#). Si vous spécifiez des `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` attributs, les attributs de partition et de tri doivent également l'être `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

Note

Pour utiliser l'action `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` cryptographique, vous devez utiliser la version 3.3 ou ultérieure du SDK AWS Database Encryption. Déployez la nouvelle version sur tous les lecteurs avant de [mettre à jour votre modèle de données](#) pour l'inclure `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

Ajouter `ENCRYPT_AND_SIGN` de `SIGN_ONLY` nouveaux `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` attributs et

Pour ajouter un nouvel attribut `ENCRYPT_AND_SIGN` `SIGN_ONLY`, ou un nouvel `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` attribut, définissez-le dans vos actions d'attribut.

Vous ne pouvez pas supprimer un `DO_NOTHING` attribut existant et le réajouter en tant qu'`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` attribut `ENCRYPT_AND_SIGN` `SIGN_ONLY`, ou.

Utilisation d'une classe de données annotée

Si vous avez défini vos actions d'attribut avec un `TableSchema`, ajoutez le nouvel attribut à votre classe de données annotée. Si vous ne spécifiez aucune annotation d'action d'attribut pour le nouvel attribut, le client chiffrera et signera le nouvel attribut par défaut (sauf si l'attribut fait partie de la clé primaire). Si vous souhaitez uniquement signer le nouvel attribut, vous devez l'ajouter avec l'`@DynamoDBEncryptionSignAndIncludeInEncryptionContext` annotation `@DynamoDBEncryptionSignOnly` or.

Utilisation d'un modèle d'objet

Si vous avez défini manuellement vos actions d'attribut, ajoutez le nouvel attribut aux actions d'attribut de votre modèle d'objet et spécifiez `ENCRYPT_AND_SIGN` `SIGN_ONLY`, ou `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` en tant qu'action d'attribut.

Supprimer les attributs existants

Si vous décidez que vous n'avez plus besoin d'un attribut, vous pouvez arrêter d'écrire des données dans cet attribut ou vous pouvez le supprimer officiellement de vos actions d'attribut. Lorsque vous arrêtez d'écrire de nouvelles données dans un attribut, celui-ci apparaît toujours dans vos actions d'attribut. Cela peut être utile si vous devez recommencer à utiliser l'attribut à l'avenir. La suppression officielle de l'attribut de vos actions d'attribut ne le supprime pas de votre ensemble de données. Votre jeu de données contiendra toujours des éléments qui incluent cet attribut.

Pour supprimer officiellement un attribut existant `ENCRYPT_AND_SIGN`, `SIGN_ONLY` ou `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, ou un `DO_NOTHING` attribut, mettez à jour vos actions d'attribut.

Si vous supprimez un `DO_NOTHING` attribut, vous ne devez pas le supprimer de vos [attributs non signés autorisés](#). Même si vous n'écrivez plus de nouvelles valeurs dans cet attribut, le client doit tout de même savoir que l'attribut n'est pas signé pour pouvoir lire les éléments existants qui le contiennent.

Utilisation d'une classe de données annotée

Si vous avez défini vos actions d'attribut avec un `TableSchema`, supprimez l'attribut de votre classe de données annotée.

Utilisation d'un modèle d'objet

Si vous avez défini manuellement vos actions d'attribut, supprimez l'attribut des actions d'attribut de votre modèle d'objet.

Remplacer un **ENCRYPT_AND_SIGN** attribut existant par **SIGN_ONLY** ou **SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT**

Pour remplacer un `ENCRYPT_AND_SIGN` attribut existant par `SIGN_ONLY` ou `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, vous devez mettre à jour vos actions d'attribut. Une fois la mise à jour déployée, le client sera en mesure de vérifier et de déchiffrer les valeurs existantes écrites dans l'attribut, mais il ne signera que les nouvelles valeurs écrites dans l'attribut.

Note

Réfléchissez bien à vos exigences de sécurité avant de remplacer un `ENCRYPT_AND_SIGN` attribut existant par `SIGN_ONLY` ou `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Tout attribut susceptible de stocker des données sensibles doit être chiffré.

Utilisation d'une classe de données annotée

Si vous avez défini vos actions d'attribut avec un `TableSchema`, mettez à jour l'attribut existant pour inclure l'annotation `@DynamoDBEncryptionSignAndIncludeInEncryptionContext` ou `@DynamoDBEncryptionSignOnly` dans votre classe de données annotée.

Utilisation d'un modèle d'objet

Si vous avez défini manuellement vos actions d'attribut, mettez à jour l'action d'attribut associée à l'attribut existant depuis `SIGN_ONLY` ou `ENCRYPT_AND_SIGN` vers `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` dans votre modèle d'objet.

Modifier un `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` attribut `SIGN_ONLY` ou un existant en `ENCRYPT_AND_SIGN`

Pour remplacer un attribut existant `SIGN_ONLY` ou par un `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` attribut `ENCRYPT_AND_SIGN`, vous devez mettre à jour vos actions d'attribut. Une fois la mise à jour déployée, le client sera en mesure de vérifier les valeurs existantes écrites dans l'attribut, puis de chiffrer et de signer les nouvelles valeurs écrites dans l'attribut.

Utilisation d'une classe de données annotée

Si vous avez défini vos actions d'attribut avec un `TableSchema`, supprimez l'annotation `@DynamoDBEncryptionSignAndIncludeInEncryptionContext` ou `@DynamoDBEncryptionSignOnly` de l'attribut existant.

Utilisation d'un modèle d'objet

Si vous avez défini manuellement vos actions d'attribut, mettez à jour l'action d'attribut associée à l'attribut depuis `SIGN_ONLY` ou `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` vers `ENCRYPT_AND_SIGN` dans votre modèle d'objet.

Ajouter un nouvel **DO_NOTHING** attribut

Pour réduire le risque d'erreur lors de l'ajout d'un nouvel `DO_NOTHING` attribut, nous vous recommandons de spécifier un préfixe distinct lorsque vous nommez vos `DO_NOTHING` attributs, puis d'utiliser ce préfixe pour définir les attributs [non signés autorisés](#).

Vous ne pouvez pas supprimer un `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` attribut ou un attribut existant `ENCRYPT_AND_SIGN` de votre classe de données annotée, puis le réintégrer en tant qu'`DO_NOTHING` attribut. `SIGN_ONLY` Vous ne pouvez ajouter que des `DO_NOTHING` attributs entièrement nouveaux.

Les étapes à suivre pour ajouter un nouvel `DO_NOTHING` attribut varient selon que vous avez défini vos attributs non signés autorisés de manière explicite dans une liste ou à l'aide d'un préfixe.

Utilisation d'un préfixe d'attributs non signés autorisé

Si vous avez défini vos actions d'attribut avec un `TableSchema`, ajoutez le nouvel `DO_NOTHING` attribut à votre classe de données annotée avec l'`@DynamoDBEncryptionDoNothing` annotation. Si vous avez défini manuellement vos actions d'attribut, mettez-les à jour pour inclure le nouvel attribut. Assurez-vous de configurer explicitement le nouvel attribut avec l'action d'`DO_NOTHING` attribut. Vous devez inclure le même préfixe distinct dans le nom du nouvel attribut.

Utilisation d'une liste d'attributs non signés autorisés

1. Ajoutez le nouvel `DO_NOTHING` attribut à votre liste d'attributs non signés autorisés et déployez la liste mise à jour.
2. Déployez la modification depuis l'étape 1.

Vous ne pouvez pas passer à l'étape 3 tant que la modification ne s'est pas propagée à tous les hôtes qui ont besoin de lire ces données.

3. Ajoutez le nouvel `DO_NOTHING` attribut à vos actions d'attribut.
 - a. Si vous avez défini vos actions d'attribut avec un `TableSchema`, ajoutez le nouvel `DO_NOTHING` attribut à votre classe de données annotée avec l'`@DynamoDBEncryptionDoNothing` annotation.
 - b. Si vous avez défini manuellement vos actions d'attribut, mettez-les à jour pour inclure le nouvel attribut. Assurez-vous de configurer explicitement le nouvel attribut avec l'action d'`DO_NOTHING` attribut.
4. Déployez la modification depuis l'étape 3.

Modifier un **SIGN_ONLY** attribut existant en **SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT**

Pour remplacer un **SIGN_ONLY** attribut existant par **SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT**, vous devez mettre à jour vos actions d'attribut. Une fois la mise à jour déployée, le client sera en mesure de vérifier les valeurs existantes écrites dans l'attribut et continuera à signer les nouvelles valeurs écrites dans l'attribut. Les nouvelles valeurs écrites dans l'attribut seront incluses dans le [contexte de chiffrement](#).

Si vous spécifiez des **SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT** attributs, les attributs de partition et de tri doivent également l'être **SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT**.

Utilisation d'une classe de données annotée

Si vous avez défini vos actions d'attribut avec un `TableSchema`, mettez à jour l'action d'attribut associée à l'attribut de `@DynamoDBEncryptionSignOnly` à `@DynamoDBEncryptionSignAndIncludeInEncryptionContext`.

Utilisation d'un modèle d'objet

Si vous avez défini manuellement vos actions d'attribut, mettez à jour l'action d'attribut associée à l'attribut de **SIGN_ONLY** à **SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT** dans votre modèle d'objet.

Modifier un **SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT** attribut existant en **SIGN_ONLY**

Pour remplacer un **SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT** attribut existant par **SIGN_ONLY**, vous devez mettre à jour vos actions d'attribut. Une fois la mise à jour déployée, le client sera en mesure de vérifier les valeurs existantes écrites dans l'attribut et continuera à signer les nouvelles valeurs écrites dans l'attribut. Les nouvelles valeurs écrites dans l'attribut ne seront pas incluses dans le [contexte de chiffrement](#).

Avant de remplacer un **SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT** attribut existant par **SIGN_ONLY**, réfléchissez bien à l'impact que vos mises à jour peuvent avoir sur les fonctionnalités de votre [fournisseur d'ID de clé de succursale](#).

Utilisation d'une classe de données annotée

Si vous avez défini vos actions d'attribut avec un `TableSchema`, mettez à jour l'action d'attribut associée à l'attribut de `@DynamoDBEncryptionSignAndIncludeInEncryptionContext` à `@DynamoDBEncryptionSignOnly`.

Utilisation d'un modèle d'objet

Si vous avez défini manuellement vos actions d'attribut, mettez à jour l'action d'attribut associée à l'attribut de `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` à `SIGN_ONLY` dans votre modèle d'objet.

AWS SDK de chiffrement de base de données pour DynamoDB, langages de programmation disponibles

Le SDK AWS de chiffrement de base de données pour DynamoDB est disponible pour les langages de programmation suivants. Les bibliothèques spécifiques au langage varient, mais les implémentations qui en résultent sont interopérables. Vous pouvez chiffrer avec une implémentation de langage et déchiffrer avec une autre. L'interopérabilité peut être soumise à des contraintes de langage. Si c'est le cas, ces contraintes sont décrites dans la rubrique relative à l'implémentation du langage.

Rubriques

- [Java](#)
- [.NET](#)
- [Rust](#)

Java

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Cette rubrique explique comment installer et utiliser la version 3. x de la bibliothèque de chiffrement côté client Java pour DynamoDB. Pour plus de détails sur la programmation avec le SDK AWS de chiffrement de base de données pour DynamoDB, consultez les exemples [Java dans aws-database-encryption-sdk le référentiel -dynamodb](#) sur GitHub

Note

Les rubriques suivantes portent sur la version 3. x de la bibliothèque de chiffrement côté client Java pour DynamoDB.

Notre bibliothèque de chiffrement côté client a été [renommée AWS Database Encryption SDK](#). Le SDK AWS Database Encryption continue de prendre en charge les [anciennes versions du client de chiffrement DynamoDB](#).

Rubriques

- [Conditions préalables](#)
- [Installation](#)
- [Utilisation de la bibliothèque de chiffrement côté client Java pour DynamoDB](#)
- [Exemples Java](#)
- [Configurer une table DynamoDB existante pour utiliser AWS le SDK de chiffrement de base de données pour DynamoDB](#)
- [Migrer vers la version 3.x de la bibliothèque de chiffrement côté client Java pour DynamoDB](#)

Conditions préalables

Avant d'installer la version 3. x de la bibliothèque de chiffrement côté client Java pour DynamoDB, assurez-vous de remplir les conditions préalables suivantes.

Environnement de développement Java

Vous aurez besoin de Java 8 ou version ultérieure. Sur le site web d'Oracle, consultez la page [Téléchargements Java SE](#), puis téléchargez et installez le kit Java SE Development (JDK).

Si vous utilisez le kit JDK Oracle, vous devez également télécharger et installer les [fichiers Java Cryptography Extension \(JCE\) Unlimited Strength Jurisdiction Policy](#).

AWS SDK for Java 2.x

Le SDK AWS de chiffrement de base de données pour DynamoDB nécessite [le module DynamoDB Enhanced Client du](#). AWS SDK for Java 2.x Vous pouvez installer la totalité du kit SDK ou le seul module.

Pour plus d'informations sur la mise à jour de votre version du AWS SDK pour Java, voir [Migration de la version 1.x vers la version 2.x](#) du AWS SDK pour Java

AWS SDK pour Java II est disponible via Apache Maven. Vous pouvez déclarer une dépendance pour l'ensemble AWS SDK pour Java ou uniquement pour le dynamodb-enhanced module.

Installez le à l' AWS SDK pour Java aide d'Apache Maven

- Pour [importer tout le AWS SDK pour Java](#) en tant que dépendance, déclarez-le dans votre fichier pom.xml.
- Pour créer une dépendance uniquement pour le module Amazon DynamoDB dans AWS SDK pour Java le, suivez les instructions [pour](#) spécifier des modules particuliers. Réglez le groupId to software.amazon.awssdk et le artifactID todynamodb-enhanced.

Note

Si vous utilisez le AWS KMS trousseau de clés ou le trousseau de clés AWS KMS hiérarchique, vous devez également créer une dépendance pour le AWS KMS module. Réglez le groupId to software.amazon.awssdk et le artifactID tokms.

Installation

Vous pouvez installer la version 3. x de la bibliothèque de chiffrement côté client Java pour DynamoDB de la manière suivante.

Utilisation d'Apache Maven

Le client de chiffrement Amazon DynamoDB pour Java est disponible [via Apache](#) Maven avec la définition de dépendance suivante.

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

Utilisation de Gradle Kotlin

Vous pouvez utiliser [Gradle](#) pour déclarer une dépendance vis-à-vis du client de chiffrement Amazon DynamoDB pour Java en ajoutant ce qui suit à la section des dépendances de votre projet Gradle.

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

Manuellement

[Pour installer la bibliothèque de chiffrement côté client Java pour DynamoDB, clonez ou téléchargez le référentiel -dynamodb. aws-database-encryption-sdk](#) GitHub

Après avoir installé le SDK, commencez par consulter l'exemple de code de ce guide et les [exemples Java du référentiel](#) aws-database-encryption-sdk -dynamodb sur. GitHub

Utilisation de la bibliothèque de chiffrement côté client Java pour DynamoDB

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Cette rubrique décrit certaines des fonctions et classes d'assistance de la version 3. x de la bibliothèque de chiffrement côté client Java pour DynamoDB.

[Pour plus de détails sur la programmation avec la bibliothèque de chiffrement côté client Java pour DynamoDB, consultez les exemples Java, les exemples Java dans le référentiel -dynamodb sur. aws-database-encryption-sdk](#) GitHub

Rubriques

- [Chiffreurs d'éléments](#)
- [Actions relatives aux attributs dans le SDK de chiffrement AWS de base de données pour DynamoDB](#)
- [Configuration du chiffrement dans le SDK de chiffrement AWS de base de données pour DynamoDB](#)
- [Mise à jour d'éléments avec le SDK AWS de chiffrement de base de données](#)

- [Déchiffrer des sets signés](#)

Chiffreurs d'éléments

À la base, le SDK de chiffrement AWS de base de données pour DynamoDB est un outil de chiffrement d'éléments. Vous pouvez utiliser la version 3. x de la bibliothèque de chiffrement côté client Java pour DynamoDB afin de chiffrer, signer, vérifier et déchiffrer les éléments de votre table DynamoDB de la manière suivante.

Le client amélioré DynamoDB

Vous pouvez configurer le client [DynamoDB amélioré](#) pour chiffrer et signer automatiquement `DynamoDbEncryptionInterceptor` les éléments côté client avec vos demandes `DynamoDB.PutItem`. Avec le client DynamoDB Enhanced, vous pouvez définir vos actions attributaires à l'aide d'[une](#) classe de données annotée. Nous vous recommandons d'utiliser le client DynamoDB amélioré dans la mesure du possible.

[Le client DynamoDB Enhanced ne prend pas en charge le chiffrement consultable.](#)

Note

Le SDK AWS de chiffrement de base de données ne prend pas en charge les annotations sur les attributs [imbriqués](#).

L'API DynamoDB de bas niveau

Vous pouvez configurer l'API [DynamoDB de bas niveau](#) pour chiffrer et signer automatiquement `DynamoDbEncryptionInterceptor` les éléments côté client avec vos demandes `DynamoDB.PutItem`.

[Vous devez utiliser l'API DynamoDB de bas niveau pour utiliser le chiffrement consultable.](#)

Le niveau inférieur **`DynamoDbItemEncryptor`**

Le niveau inférieur chiffre, signe ou déchiffre et vérifie `DynamoDbItemEncryptor` directement les éléments de votre table sans appeler DynamoDB. Il n'émet pas de `DynamoDB` ni de `PutItem` requêtes `GetItem`. Par exemple, vous pouvez utiliser le niveau inférieur `DynamoDbItemEncryptor` pour déchiffrer et vérifier directement un élément DynamoDB que vous avez déjà récupéré.

Le niveau inférieur `DynamoDbItemEncryptor` ne prend pas en charge le chiffrement [consultable](#).

Actions relatives aux attributs dans le SDK de chiffrement AWS de base de données pour DynamoDB

Les [actions d'attribut](#) déterminent quelles valeurs d'attribut sont cryptées et signées, lesquelles sont uniquement signées, lesquelles sont signées et incluses dans le contexte de chiffrement, et lesquelles sont ignorées.

Note

Pour utiliser l'action `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` cryptographique, vous devez utiliser la version 3.3 ou ultérieure du SDK AWS Database Encryption. Déployez la nouvelle version sur tous les lecteurs avant de [mettre à jour votre modèle de données](#) pour l'inclure `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

Si vous utilisez l'API DynamoDB de bas niveau ou de `DynamoDbItemEncryptor` niveau inférieur, vous devez définir manuellement vos actions attributaires. [Si vous utilisez le client DynamoDB amélioré, vous pouvez soit définir manuellement vos actions attributaires, soit utiliser une classe de données annotée pour générer un `TableSchema`](#) Pour simplifier le processus de configuration, nous vous recommandons d'utiliser une classe de données annotée. Lorsque vous utilisez une classe de données annotée, vous ne devez modéliser votre objet qu'une seule fois.

Note

Après avoir défini les actions relatives aux attributs, vous devez définir quels attributs sont exclus des signatures. Pour faciliter l'ajout de nouveaux attributs non signés à l'avenir, nous vous recommandons de choisir un préfixe distinct (tel que : « ») pour identifier vos attributs non signés. Incluez ce préfixe dans le nom d'attribut pour tous les attributs marqués lorsque vous `DO_NOTHING` définissez votre schéma DynamoDB et vos actions d'attribut.

Utiliser une classe de données annotée

Utilisez une [classe de données annotée](#) pour spécifier vos actions attributaires avec le client DynamoDB amélioré et `DynamoDbEncryptionInterceptor`. Le SDK AWS de chiffrement de base de données pour DynamoDB utilise les annotations d'attribut [DynamoDB standard qui](#)

[définissent le type d'attribut](#) afin de déterminer comment protéger un attribut. Par défaut, tous les attributs sont chiffrés et signés à l'exception des clés primaires, qui sont signées, mais pas chiffrées.

Note

Pour utiliser l'action `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` cryptographique, vous devez utiliser la version 3.3 ou ultérieure du SDK AWS Database Encryption. Déployez la nouvelle version sur tous les lecteurs avant de [mettre à jour votre modèle de données](#) pour l'inclure `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

Consultez [SimpleClass.java](#) dans le référentiel `aws-database-encryption-sdk -dynamodb` GitHub pour plus d'informations sur les annotations du client DynamoDB Enhanced.

Par défaut, les attributs de clé primaire sont signés mais pas chiffrés (`SIGN_ONLY`) et tous les autres attributs sont chiffrés et signés (`ENCRYPT_AND_SIGN`). Si vous définissez des attributs comme `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, les attributs de partition et de tri doivent également l'être `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Pour spécifier des exceptions, utilisez les annotations de chiffrement définies dans la bibliothèque de chiffrement côté client Java pour DynamoDB. Par exemple, si vous souhaitez qu'un attribut particulier soit uniquement signé, utilisez l'annotation `@DynamoDbEncryptionSignOnly`. Si vous souhaitez qu'un attribut particulier soit signé et inclus dans le contexte de chiffrement, utilisez l'annotation `@DynamoDbEncryptionSignAndIncludeInEncryptionContext`. Si vous souhaitez qu'un attribut particulier ne soit ni signé ni chiffré (`DO_NOTHING`), utilisez l'annotation `@DynamoDbEncryptionDoNothing`.

Note

Le SDK AWS de chiffrement de base de données ne prend pas en charge les annotations sur les attributs [imbriqués](#).

L'exemple suivant montre les annotations utilisées pour définir `ENCRYPT_AND_SIGN` et `DO_NOTHING` attribuer des actions. `SIGN_ONLY` Pour un exemple illustrant les annotations utilisées pour définir `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, consultez le [SimpleClassfichier 4.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;
}
}
```

Utilisez votre classe de données annotée pour créer le `TableSchema` comme indiqué dans l'extrait de code suivant.

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

Définissez manuellement les actions de vos attributs

Pour spécifier manuellement les actions d'attribut, créez un Map objet dans lequel les paires nom-valeur représentent les noms d'attributs et les actions spécifiées.

Spécifiez `ENCRYPT_AND_SIGN` le chiffrement et la signature d'un attribut.

Spécifiez `SIGN_ONLY` pour signer un attribut, mais pas pour le chiffrer. Spécifiez

`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` de signer un attribut et de l'inclure dans le contexte de chiffrement. Vous ne pouvez pas chiffrer un attribut sans le signer également. Spécifiez `DO_NOTHING` si un attribut doit être ignoré.

Les attributs de partition et de tri doivent être l'un `SIGN_ONLY` ou

l'autre `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Si vous définissez des attributs

comme `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, les attributs de partition et de tri doivent également l'être `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

Note

Pour utiliser l'action `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` cryptographique, vous devez utiliser la version 3.3 ou ultérieure du SDK AWS Database Encryption. Déployez la nouvelle version sur tous les lecteurs avant de [mettre à jour votre modèle de données](#) pour l'inclure `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);
```

Configuration du chiffrement dans le SDK de chiffrement AWS de base de données pour DynamoDB

Lorsque vous utilisez le SDK AWS Database Encryption, vous devez définir explicitement une configuration de chiffrement pour votre table DynamoDB. Les valeurs requises dans votre configuration de chiffrement varient selon que vous avez défini vos actions attributaires manuellement ou à l'aide d'une classe de données annotée.

L'extrait suivant définit une configuration de chiffrement de table DynamoDB à l'aide du client DynamoDB amélioré et autorise les attributs non signés définis par un préfixe distinct [TableSchema](#).

```
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());
```

Nom de table logique

Nom de table logique pour votre table DynamoDB.

Le nom de table logique est lié de manière cryptographique à toutes les données stockées dans la table afin de simplifier les opérations de restauration DynamoDB. Nous vous recommandons vivement de spécifier le nom de votre table DynamoDB comme nom de table logique lorsque vous définissez votre configuration de chiffrement pour la première fois. Vous devez toujours spécifier le même nom de table logique. Pour que le déchiffrement réussisse, le nom de la table logique doit correspondre au nom spécifié lors du chiffrement. Si le nom de votre table DynamoDB change après la [restauration de votre table DynamoDB à partir d'une sauvegarde, le nom logique de la table garantit que l'opération de déchiffrement](#) reconnaît toujours la table.

Attributs non signés autorisés

Les attributs marqués DO_NOTHING dans vos actions d'attributs.

Les attributs non signés autorisés indiquent au client quels attributs sont exclus des signatures. Le client suppose que tous les autres attributs sont inclus dans la signature. Ensuite, lors du déchiffrement d'un enregistrement, le client détermine les attributs qu'il doit vérifier et ceux à ignorer parmi les attributs non signés autorisés que vous avez spécifiés. Vous ne pouvez pas supprimer un attribut de vos attributs non signés autorisés.

Vous pouvez définir explicitement les attributs non signés autorisés en créant un tableau répertoriant tous vos DO_NOTHING attributs. Vous pouvez également spécifier un préfixe distinct lorsque vous nommez vos DO_NOTHING attributs et utiliser le préfixe pour indiquer au client quels attributs ne sont pas signés. Nous vous recommandons vivement de spécifier un préfixe distinct, car cela simplifie le processus d'ajout d'un nouvel DO_NOTHING attribut à l'avenir. Pour de plus amples informations, veuillez consulter [Mettre à jour votre modèle de données](#).

Si vous ne spécifiez pas de préfixe pour tous les DO_NOTHING attributs, vous pouvez configurer un `allowedUnsignedAttributes` tableau répertoriant explicitement tous les attributs que le client doit s'attendre à voir non signés lorsqu'il les rencontre lors du déchiffrement. Vous ne devez définir explicitement vos attributs non signés autorisés que si cela est absolument nécessaire.

Configuration de la recherche (facultatif)

`SearchConfig` Définit la [version de la balise](#).

Le `SearchConfig` doit être spécifié pour utiliser un [chiffrement consultable](#) ou des [balises signées](#).

Suite d'algorithmes (facultatif)

`algorithmSuiteId` Définit la suite d'algorithmes utilisée par le SDK AWS de chiffrement de base de données.

À moins que vous ne spécifiiez explicitement une suite d'algorithmes alternative, le SDK AWS de chiffrement de base de données utilise la [suite d'algorithmes par défaut](#). [La suite d'algorithmes par défaut utilise l'algorithme AES-GCM avec dérivation de clés, signatures numériques et engagement de clés](#). Bien que la suite d'algorithmes par défaut soit susceptible de convenir à la plupart des applications, vous pouvez choisir une autre suite d'algorithmes. Par exemple, certains modèles de confiance seraient satisfaits par une suite d'algorithmes sans signature numérique. Pour plus d'informations sur les suites d'algorithmes prises en charge par le SDK AWS de chiffrement de base de données, consultez [Suites d'algorithmes prises en charge dans le SDK AWS de chiffrement de base de données](#).

Pour sélectionner la [suite d'algorithmes AES-GCM sans signature numérique ECDSA](#), incluez l'extrait suivant dans votre configuration de chiffrement de table.

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

Mise à jour d'éléments avec le SDK AWS de chiffrement de base de données

Le SDK AWS de chiffrement de base de données ne prend pas en charge [ddb : UpdateItem](#) pour les éléments chiffrés ou signés. Pour mettre à jour un élément chiffré ou signé, vous devez utiliser [ddb : PutItem](#). Lorsque vous spécifiez la même clé primaire qu'un élément existant dans votre `PutItem` demande, le nouvel élément remplace complètement l'élément existant. Vous pouvez également utiliser [CLOBBER](#) pour effacer et remplacer tous les attributs lors de la sauvegarde après avoir mis à jour vos articles.

Déchiffrer des sets signés

Dans les versions 3.0.0 et 3.1.0 du SDK de chiffrement de AWS base de données, si vous définissez un attribut de [type set](#) comme `SIGN_ONLY`, les valeurs de l'ensemble sont canonisées dans l'ordre dans lequel elles sont fournies. DynamoDB ne préserve pas l'ordre des ensembles. Par conséquent, il est possible que la validation de signature de l'élément contenant l'ensemble échoue. La validation de signature échoue lorsque les valeurs de l'ensemble sont renvoyées dans un ordre différent de celui dans lequel elles ont été fournies au SDK AWS Database Encryption, même si les attributs de l'ensemble contiennent les mêmes valeurs.

Note

Les versions 3.1.1 et ultérieures du SDK de chiffrement de AWS base de données canonisent les valeurs de tous les attributs de type défini, afin que les valeurs soient lues dans le même ordre que celui dans lequel elles ont été écrites dans DynamoDB.

Si la validation de signature échoue, l'opération de déchiffrement échoue et renvoie le message d'erreur suivant.

```
software.amazon.cryptography.dbencryption.sdk.structuredencryption.model.StructuredEncryptionException: Aucune étiquette de destinataire ne correspond.
```

Si le message d'erreur ci-dessus s'affiche et que vous pensez que l'élément que vous essayez de déchiffrer inclut un ensemble signé à l'aide de la version 3.0.0 ou 3.1.0, consultez le [DecryptWithPermute](#) répertoire du dépôt `aws-database-encryption-sdk-dynamodb-java` GitHub pour savoir comment valider correctement l'ensemble.

Exemples Java

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Les exemples suivants montrent comment utiliser la bibliothèque de chiffrement côté client Java pour DynamoDB afin de protéger les éléments de table de votre application. Vous pouvez trouver d'autres exemples (et apporter les vôtres) dans les [exemples Java](#) du référentiel `aws-database-encryption-sdk-dynamodb` sur GitHub.

Les exemples suivants montrent comment configurer la bibliothèque de chiffrement côté client Java pour DynamoDB dans une nouvelle table Amazon DynamoDB non remplie. Si vous souhaitez configurer vos tables Amazon DynamoDB existantes pour le chiffrement côté client, consultez [Ajouter la version 3.x à une table existante](#).

Rubriques

- [Utilisation du client amélioré DynamoDB](#)
- [Utilisation de l'API DynamoDB de bas niveau](#)
- [Utiliser le niveau inférieur DynamoDbItemEncryptor](#)

Utilisation du client amélioré DynamoDB

L'exemple suivant montre comment utiliser le client amélioré DynamoDB

DynamoDbEncryptionInterceptor et [AWS KMS un](#) trousseau de clés pour chiffrer des éléments de table DynamoDB dans le cadre de vos appels d'API DynamoDB.

Vous pouvez utiliser n'importe quel trousseau de [clés](#) compatible avec le client DynamoDB amélioré, mais nous vous recommandons d'utiliser l'un des trousseaux de clés dans la mesure du AWS KMS possible.

Note

[Le client DynamoDB Enhanced ne prend pas en charge le chiffrement consultable.](#) Utilisez le DynamoDbEncryptionInterceptor avec l'API DynamoDB de bas niveau pour utiliser le chiffrement consultable.

Voir l'exemple de code complet : [EnhancedPutGetExample.java](#)

Étape 1 : Création du AWS KMS porte-clés

L'exemple suivant permet de `CreateAwsKmsMrkMultiKeyring` créer un AWS KMS trousseau de clés avec une clé KMS de chiffrement symétrique. La `CreateAwsKmsMrkMultiKeyring` méthode garantit que le trousseau de clés gère correctement les clés à région unique et à régions multiples.

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

Étape 2 : Création d'un schéma de table à partir de la classe de données annotée

L'exemple suivant utilise la classe de données annotée pour créer le `TableSchema`.

Cet exemple suppose que les actions de classe de données et d'attribut annotées ont été définies à l'aide du [SimpleClassfichier.java](#). Pour plus d'informations sur l'annotation des actions de vos attributs, consultez [Utiliser une classe de données annotée](#).

Note

Le SDK AWS de chiffrement de base de données ne prend pas en charge les annotations sur les attributs [imbriqués](#).

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

Étape 3 : définir les attributs exclus des signatures

L'exemple suivant suppose que tous les `DO_NOTHING` attributs partagent le préfixe distinct : « » et utilise le préfixe pour définir les attributs non signés autorisés. Le client suppose que tout nom d'attribut avec le préfixe : « » est exclu des signatures. Pour de plus amples informations, veuillez consulter [Allowed unsigned attributes](#).

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

Étape 4 : Création de la configuration de chiffrement

L'exemple suivant définit une `tableConfigs` carte qui représente la configuration de chiffrement de la table DynamoDB.

[Cet exemple indique le nom de la table DynamoDB comme nom de table logique](#). Nous vous recommandons vivement de spécifier le nom de votre table DynamoDB comme nom de table logique lorsque vous définissez votre configuration de chiffrement pour la première fois. Pour de plus amples informations, veuillez consulter [Configuration du chiffrement dans le SDK de chiffrement AWS de base de données pour DynamoDB](#).

Note

Pour utiliser le [chiffrement consultable](#) ou les [balises signées](#), vous devez également les inclure [SearchConfig](#) dans votre configuration de chiffrement.

```
final Map<String, DynamoDbEnhancedTableEncryptionConfig> tableConfigs = new
    HashMap<>();
tableConfigs.put(ddbTableName,
    DynamoDbEnhancedTableEncryptionConfig.builder()
        .logicalTableName(ddbTableName)
        .keyring(kmsKeyring)
        .allowedUnsignedAttributePrefix(unsignedAttrPrefix)
        .schemaOnEncrypt(tableSchema)
        .build());
```

Étape 5 : Crée le `DynamoDbEncryptionInterceptor`

L'exemple suivant en crée un nouveau `DynamoDbEncryptionInterceptor` à `tableConfigs` partir de l'étape 4.

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

Étape 6 : créer un nouveau client DynamoDB du AWS SDK

L'exemple suivant crée un nouveau client DynamoDB du AWS SDK à l'aide **interceptor** de l'étape 5.

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

Étape 7 : Création du client DynamoDB amélioré et création d'une table

L'exemple suivant crée le client DynamoDB amélioré à l'aide du client DynamoDB AWS SDK créé à l'étape 6 et crée une table à l'aide de la classe de données annotée.

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

Étape 8 : Chiffrer et signer un élément du tableau

L'exemple suivant place un élément dans la table DynamoDB à l'aide du client DynamoDB amélioré. L'élément est chiffré et signé côté client avant d'être envoyé à 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);
```

Utilisation de l'API DynamoDB de bas niveau

L'exemple suivant montre comment utiliser l'API DynamoDB de bas niveau avec [AWS KMS un](#) trousseau de clés pour chiffrer et signer automatiquement des éléments côté client avec vos demandes DynamoDB. PutItem

Vous pouvez utiliser n'importe quel [porte-clés](#) compatible, mais nous vous recommandons d'utiliser l'un des AWS KMS porte-clés dans la mesure du possible.

Voir l'exemple de code complet : [BasicPutGetExample.java](#)

Étape 1 : Création du AWS KMS porte-clés

L'exemple suivant permet de `CreateAwsKmsMrkMultiKeyring` créer un AWS KMS trousseau de clés avec une clé KMS de chiffrement symétrique. La `CreateAwsKmsMrkMultiKeyring`

méthode garantit que le trousseau de clés gère correctement les clés à région unique et à régions multiples.

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

Étape 2 : configurer les actions de vos attributs

L'exemple suivant définit une `attributeActionsOnEncrypt` carte qui représente des exemples d'[actions attributaires](#) pour un élément de table.

Note

L'exemple suivant ne définit aucun attribut en tant que `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Si vous spécifiez des `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` attributs, les attributs de partition et de tri doivent également l'être `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);
```

Étape 3 : définir les attributs exclus des signatures

L'exemple suivant suppose que tous les `DO_NOTHING` attributs partagent le préfixe distinct : « » et utilise le préfixe pour définir les attributs non signés autorisés. Le client suppose que tout nom d'attribut avec le préfixe : « » est exclu des signatures. Pour de plus amples informations, veuillez consulter [Allowed unsigned attributes](#).

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

Étape 4 : définir la configuration du chiffrement des tables DynamoDB

L'exemple suivant définit une `tableConfigs` carte qui représente la configuration de chiffrement pour cette table DynamoDB.

[Cet exemple indique le nom de la table DynamoDB comme nom de table logique.](#) Nous vous recommandons vivement de spécifier le nom de votre table DynamoDB comme nom de table logique lorsque vous définissez votre configuration de chiffrement pour la première fois. Pour de plus amples informations, veuillez consulter [Configuration du chiffrement dans le SDK de chiffrement AWS de base de données pour DynamoDB](#).

Note

Pour utiliser le [chiffrement consultable](#) ou les [balises signées](#), vous devez également les inclure [SearchConfig](#) dans votre configuration de chiffrement.

```
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);
```

Étape 5 : Création du **DynamoDbEncryptionInterceptor**

L'exemple suivant crée le `DynamoDbEncryptionInterceptor` à l'aide `tableConfigs` de l'étape 4.

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

Étape 6 : créer un nouveau client DynamoDB du AWS SDK

L'exemple suivant crée un nouveau client DynamoDB du AWS SDK à l'aide **interceptor** de l'étape 5.

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

Étape 7 : Chiffrer et signer un élément de table DynamoDB

L'exemple suivant définit une item carte qui représente un exemple d'élément de table et place l'élément dans la table DynamoDB. L'élément est chiffré et signé côté client avant d'être envoyé à 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);
```

Utiliser le niveau inférieur DynamoDbItemEncryptor

L'exemple suivant montre comment utiliser le niveau inférieur `DynamoDbItemEncryptor` avec un [AWS KMS trousseau de clés](#) pour chiffrer et signer directement des éléments de table. L'élément `DynamoDbItemEncryptor` n'est pas placé dans votre table DynamoDB.

Vous pouvez utiliser n'importe quel trousseau de [clés](#) compatible avec le client DynamoDB amélioré, mais nous vous recommandons d'utiliser l'un des trousseaux de clés dans la mesure du AWS KMS possible.

Note

Le niveau inférieur `DynamoDbItemEncryptor` ne prend pas en charge le chiffrement [consultable](#). Utilisez le `DynamoDbEncryptionInterceptor` avec l'API DynamoDB de bas niveau pour utiliser le chiffrement consultable.

Voir l'exemple de code complet : [ItemEncryptDecryptExample.java](#)

Étape 1 : Création du AWS KMS porte-clés

L'exemple suivant permet de `CreateAwsKmsMrkMultiKeyring` créer un AWS KMS trousseau de clés avec une clé KMS de chiffrement symétrique. La `CreateAwsKmsMrkMultiKeyring` méthode garantit que le trousseau de clés gère correctement les clés à région unique et à régions multiples.

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

Étape 2 : configurer les actions de vos attributs

L'exemple suivant définit une `attributeActionsOnEncrypt` carte qui représente des exemples d'[actions attributaires](#) pour un élément de table.

Note

L'exemple suivant ne définit aucun attribut en tant que `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Si vous spécifiez des `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` attributs, les attributs de partition et de tri doivent également l'être `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);
```

Étape 3 : définir les attributs exclus des signatures

L'exemple suivant suppose que tous les DO_NOTHING attributs partagent le préfixe distinct : « » et utilise le préfixe pour définir les attributs non signés autorisés. Le client suppose que tout nom d'attribut avec le préfixe : « » est exclu des signatures. Pour de plus amples informations, veuillez consulter [Allowed unsigned attributes](#).

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

Étape 4 : définir la **DynamoDbItemEncryptor** configuration

L'exemple suivant définit la configuration de `DynamoDbItemEncryptor`.

[Cet exemple indique le nom de la table DynamoDB comme nom de table logique](#). Nous vous recommandons vivement de spécifier le nom de votre table DynamoDB comme nom de table logique lorsque vous définissez votre configuration de chiffrement pour la première fois. Pour de plus amples informations, veuillez consulter [Configuration du chiffrement dans le SDK de chiffrement AWS de base de données pour DynamoDB](#).

```
final DynamoDbItemEncryptorConfig config = DynamoDbItemEncryptorConfig.builder()  
    .logicalTableName(ddbTableName)  
    .partitionKeyName("partition_key")  
    .sortKeyName("sort_key")  
    .attributeActionsOnEncrypt(attributeActionsOnEncrypt)  
    .keyring(kmsKeyring)  
    .allowedUnsignedAttributePrefix(unsignedAttrPrefix)  
    .build();
```

Étape 5 : Création du **DynamoDbItemEncryptor**

L'exemple suivant en crée un nouveau à `DynamoDbItemEncryptor` l'aide de l'étape 4 config à partir de l'étape 4.

```
final DynamoDbItemEncryptor itemEncryptor = DynamoDbItemEncryptor.builder()
```

```
.DynamoDbItemEncryptorConfig(config)
    .build();
```

Étape 6 : Chiffrer et signer directement un élément du tableau

L'exemple suivant chiffre et signe directement un élément à l'aide du `DynamoDbItemEncryptor`. L'élément `DynamoDbItemEncryptor` n'est pas placé dans votre table DynamoDB.

```
final Map<String, AttributeValue> originalItem = new HashMap<>();
originalItem.put("partition_key",
    AttributeValue.builder().s("ItemEncryptDecryptExample").build());
originalItem.put("sort_key", AttributeValue.builder().n("0").build());
originalItem.put("attribute1", AttributeValue.builder().s("encrypt and sign
me!").build());
originalItem.put("attribute2", AttributeValue.builder().s("sign me!").build());
originalItem.put(":attribute3", AttributeValue.builder().s("ignore me!").build());

final Map<String, AttributeValue> encryptedItem = itemEncryptor.EncryptItem(
    EncryptItemInput.builder()
        .plaintextItem(originalItem)
        .build()
    ).encryptedItem();
```

Configurer une table DynamoDB existante pour utiliser AWS le SDK de chiffrement de base de données pour DynamoDB

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Avec la version 3. x de la bibliothèque de chiffrement côté client Java pour DynamoDB, vous pouvez configurer vos tables Amazon DynamoDB existantes pour le chiffrement côté client. Cette rubrique fournit des conseils sur les trois étapes à suivre pour ajouter la version 3. x vers une table DynamoDB existante et renseignée.

Conditions préalables

Version 3. x [de la bibliothèque de chiffrement côté client Java pour DynamoDB nécessite le client DynamoDB amélioré fourni dans](#). AWS SDK for Java 2.x Si vous utilisez toujours [Dynamo](#)

[DBMapper](#), vous devez effectuer la migration AWS SDK for Java 2.x vers le client DynamoDB amélioré.

Suivez les instructions de [migration de la version 1.x vers la version 2.x](#) du AWS SDK pour Java

Suivez ensuite les instructions pour [commencer à utiliser l'API client améliorée DynamoDB](#).

[Avant de configurer votre table pour utiliser la bibliothèque de chiffrement côté client Java pour DynamoDB, vous devez générer une TableSchema classe de données annotée et créer un client amélioré.](#)

Étape 1 : Préparez-vous à lire et à écrire des éléments chiffrés

Procédez comme suit pour préparer votre client du SDK AWS Database Encryption à lire et à écrire des éléments chiffrés. Après avoir déployé les modifications suivantes, votre client continuera à lire et à écrire des éléments en texte brut. Il ne chiffrera ni ne signera aucun nouvel élément inscrit dans la table, mais il sera en mesure de déchiffrer les éléments chiffrés dès leur apparition. Ces modifications préparent le client à commencer à [chiffrer de nouveaux éléments](#). Les modifications suivantes doivent être déployées sur chaque lecteur avant de passer à l'étape suivante.

1. Définissez vos [actions attributaires](#)

Mettez à jour votre classe de données annotée pour inclure des actions d'attribut qui définissent les valeurs d'attribut qui seront chiffrées et signées, celles qui seront uniquement signées et celles qui seront ignorées.

Consultez le [SimpleClassfichier.java](#) dans le référentiel `aws-database-encryption-sdk-dynamodb` GitHub pour plus d'informations sur les annotations du client DynamoDB Enhanced.

Par défaut, les attributs de clé primaire sont signés mais pas chiffrés (`SIGN_ONLY`) et tous les autres attributs sont chiffrés et signés (`ENCRYPT_AND_SIGN`). Pour spécifier des exceptions, utilisez les annotations de chiffrement définies dans la bibliothèque de chiffrement côté client Java pour DynamoDB. Par exemple, si vous souhaitez qu'un attribut particulier soit un signe, utilisez uniquement l'`@DynamoDbEncryptionSignOnly` annotation. Si vous souhaitez qu'un attribut particulier soit signé et inclus dans le contexte de chiffrement, utilisez l'`@DynamoDbEncryptionSignAndIncludeInEncryptionContext` annotation. Si vous souhaitez qu'un attribut particulier ne soit ni signé ni chiffré (`DO_NOTHING`), utilisez l'`@DynamoDbEncryptionDoNothing` annotation.

Note

Si vous spécifiez des `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` attributs, les attributs de partition et de tri doivent également l'être `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Pour un exemple illustrant les annotations utilisées pour définir `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, consultez le [SimpleClassfichier 4.java](#).

Pour des exemples d'annotations, voir [Utiliser une classe de données annotée](#).

2. Définissez les attributs qui seront exclus des signatures

L'exemple suivant suppose que tous les `DO_NOTHING` attributs partagent le préfixe distinct : « » et utilise le préfixe pour définir les attributs non signés autorisés. Le client supposera que tout nom d'attribut avec le préfixe : « » est exclu des signatures. Pour de plus amples informations, veuillez consulter [Allowed unsigned attributes](#).

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

3. Créez un [porte-clés](#)

L'exemple suivant crée un [AWS KMS trousseau de clés](#). Le AWS KMS trousseau de clés utilise le chiffrement symétrique ou le RSA asymétrique AWS KMS keys pour générer, chiffrer et déchiffrer les clés de données.

Cet exemple permet `CreateMrkMultiKeyring` de créer un AWS KMS trousseau de clés avec une clé KMS de chiffrement symétrique. La `CreateAwsKmsMrkMultiKeyring` méthode garantit que le trousseau de clés gère correctement les clés à région unique et à régions multiples.

```
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. Définition de la configuration du chiffrement des tables DynamoDB

L'exemple suivant définit une `tableConfigs` carte qui représente la configuration de chiffrement pour cette table DynamoDB.

[Cet exemple indique le nom de la table DynamoDB comme nom de table logique](#). Nous vous recommandons vivement de spécifier le nom de votre table DynamoDB comme nom de table logique lorsque vous définissez votre configuration de chiffrement pour la première fois. Pour de plus amples informations, veuillez consulter [Configuration du chiffrement dans le SDK de chiffrement AWS de base de données pour DynamoDB](#).

Vous devez le spécifier `FORCE_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT` comme remplacement en texte brut. Cette politique continue de lire et d'écrire des éléments en texte brut, de lire des éléments chiffrés et de préparer le client à écrire des éléments chiffrés.

```
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. Créer le `DynamoDbEncryptionInterceptor`

L'exemple suivant crée le `DynamoDbEncryptionInterceptor` à l'aide `tableConfigs` de l'étape 3.

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

Étape 2 : Écrire des éléments chiffrés et signés

Mettez à jour la politique de texte brut dans votre `DynamoDbEncryptionInterceptor` configuration pour permettre au client d'écrire des éléments chiffrés et signés. Après avoir déployé la modification suivante, le client chiffre et signe les nouveaux éléments en fonction des actions d'attribut que vous avez configurées à l'étape 1. Le client pourra lire les éléments en texte brut et les éléments chiffrés et signés.

Avant de passer à l'[étape 3](#), vous devez chiffrer et signer tous les éléments en texte brut existants dans votre tableau. Il n'existe pas de métrique ou de requête unique que vous puissiez exécuter pour chiffrer rapidement vos éléments en texte brut existants. Utilisez le processus le mieux adapté à votre système. Par exemple, vous pouvez utiliser un processus asynchrone qui analyse lentement la table et réécrit les éléments à l'aide des actions attributaires et de la configuration de chiffrement que vous avez définies. Pour identifier les éléments en texte brut de votre tableau, nous vous recommandons de rechercher tous les éléments qui ne contiennent pas les `aws_dbe_foot` attributs `aws_dbe_head` et que le SDK de chiffrement de AWS base de données ajoute aux éléments lorsqu'ils sont chiffrés et signés.

L'exemple suivant met à jour la configuration de chiffrement des tables depuis l'étape 1. Vous devez mettre à jour le remplacement en texte brut avec.

`FORBID_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT` Cette politique continue de lire les éléments en texte brut, mais également de lire et d'écrire des éléments chiffrés. Créez-en un nouveau `DynamoDbEncryptionInterceptor` à l'aide de la mise à jour `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);
```

Étape 3 : lire uniquement les éléments chiffrés et signés

Une fois que vous avez chiffré et signé tous vos éléments, mettez à jour la dérogation en texte brut dans votre `DynamoDbEncryptionInterceptor` configuration pour autoriser uniquement le client à lire et écrire des éléments chiffrés et signés. Après avoir déployé la modification suivante, le client chiffre et signe les nouveaux éléments en fonction des actions d'attribut que vous avez configurées à l'étape 1. Le client ne pourra lire que les éléments chiffrés et signés.

L'exemple suivant met à jour la configuration de chiffrement des tables depuis l'étape 2. Vous pouvez soit mettre à jour la règle de remplacement en texte brut, `FORBID_WRITE_PLAINTEXT_FORBID_READ_PLAINTEXT` soit la supprimer de votre configuration. Le client lit et écrit uniquement les éléments chiffrés et signés par défaut. Créez-en un nouveau `DynamoDbEncryptionInterceptor` à l'aide de la mise à jour `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);
```

Migrer vers la version 3.x de la bibliothèque de chiffrement côté client Java pour DynamoDB

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Version 3. x de la bibliothèque de chiffrement côté client Java pour DynamoDB est une réécriture majeure du 2. base de code x. Il inclut de nombreuses mises à jour, telles qu'un nouveau format de données structuré, une prise en charge améliorée de la mutualisation, des modifications de schéma

fluides et la prise en charge du chiffrement consultable. Cette rubrique fournit des conseils sur la façon de migrer votre code vers la version 3. x.

Migration de la version 1.x vers la version 2.x

Migrez vers la version 2. x avant de migrer vers la version 3. x. Version 2. x a changé le symbole du fournisseur le plus récent de `MostRecentProvider` à `CachingMostRecentProvider`. Si vous utilisez actuellement la version 1. x de la bibliothèque de chiffrement côté client Java pour DynamoDB avec le `MostRecentProvider` symbole, vous devez mettre à jour le nom du symbole dans votre code en `CachingMostRecentProvider`. Pour plus d'informations, voir [Mises à jour du fournisseur le plus récent](#).

Migration de la version 2.x vers la version 3.x

Les procédures suivantes décrivent comment migrer votre code depuis la version 2. x vers la version 3. x de la bibliothèque de chiffrement côté client Java pour DynamoDB.

Étape 1. Préparez-vous à lire les éléments dans le nouveau format

Procédez comme suit pour préparer votre client AWS Database Encryption SDK à lire les éléments dans le nouveau format. Après avoir déployé les modifications suivantes, votre client continuera à se comporter de la même manière que dans la version 2. x. Votre client continuera à lire et à écrire des éléments dans la version 2. format x, mais ces modifications préparent le client à [lire les éléments dans le nouveau format](#).

Mettez à jour votre AWS SDK pour Java version 2.x

Version 3. x [de la bibliothèque de chiffrement côté client Java pour DynamoDB nécessite le client DynamoDB amélioré](#). Le client DynamoDB amélioré remplace le `DBMapper Dynamo` utilisé [dans les versions](#) précédentes. Pour utiliser le client amélioré, vous devez utiliser le AWS SDK for Java 2.x.

Suivez les instructions de [migration de la version 1.x vers la version 2.x](#) du. AWS SDK pour Java

Pour plus d'informations sur les AWS SDK for Java 2.x modules requis, consultez [Conditions préalables](#).

Configurez votre client pour lire les éléments chiffrés par les anciennes versions

Les procédures suivantes fournissent une vue d'ensemble des étapes illustrées dans l'exemple de code ci-dessous.

1. Créez un [porte-clés](#).

Les porte-clés et les [gestionnaires de matériel cryptographique](#) remplacent les fournisseurs de matériel cryptographique utilisés dans les versions précédentes de la bibliothèque de chiffrement côté client Java pour DynamoDB.

 Important


Les clés d'encapsulation que vous spécifiez lors de la création d'un jeu de clés doivent être les mêmes que celles que vous avez utilisées avec votre fournisseur de matériel cryptographique dans la version 2. x.

2. Créez un schéma de table au-dessus de votre classe annotée.

Cette étape définit les actions d'attribut qui seront utilisées lorsque vous commencerez à écrire des éléments dans le nouveau format.

Pour obtenir des conseils sur l'utilisation du nouveau client DynamoDB amélioré, reportez-vous à la section [Generate TableSchema a du](#) guide du développeur.AWS SDK pour Java

L'exemple suivant suppose que vous avez mis à jour votre classe annotée à partir de la version 2. x en utilisant les nouvelles annotations d'actions d'attribut. Pour plus d'informations sur l'annotation des actions de vos attributs, consultez [Utiliser une classe de données annotée.](#)

 Note

Si vous spécifiez des `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` attributs, les attributs de partition et de tri doivent également l'être `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Pour un exemple illustrant les annotations utilisées pour définir `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, consultez [SimpleClass4.java](#).

3. Définissez les [attributs exclus de la signature](#).
4. Configurez une carte explicite des actions d'attributs configurées dans votre classe modélisée de la version 2.x.

Cette étape définit les actions d'attribut utilisées pour écrire des éléments dans l'ancien format.

5. Configurez le `DynamoDBEncryptor` que vous avez utilisé dans la version 2. x de la bibliothèque de chiffrement côté client Java pour DynamoDB.
6. Configurez le comportement existant.
7. Créez un `DynamoDbEncryptionInterceptor`.
8. Créez un nouveau client AWS DynamoDB SDK.
9. Créez le `DynamoDBEnhancedClient` et créez un tableau avec votre classe modélisée.

Pour plus d'informations sur le client DynamoDB amélioré, [voir créer un](#) client amélioré.

```
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);
    }
}
```

Étape 2. Écrire des éléments dans le nouveau format

Après avoir déployé les modifications apportées à l'étape 1 sur tous les lecteurs, effectuez les étapes suivantes pour configurer votre client AWS Database Encryption SDK afin d'écrire des éléments dans le nouveau format. Après avoir déployé les modifications suivantes, votre client continuera à lire les éléments dans l'ancien format et commencera à écrire et à lire des éléments dans le nouveau format.

Les procédures suivantes fournissent une vue d'ensemble des étapes illustrées dans l'exemple de code ci-dessous.

1. Continuez à configurer votre trousseau de clés, votre schéma de table, vos anciennes actions attributaires `allowedUnsignedAttributes`, `DynamoDBEncryptor` comme vous l'avez fait à l'[étape 1](#).
2. Mettez à jour votre ancien comportement pour n'écrire que les nouveaux éléments en utilisant le nouveau format.
3. Créer une `DynamoDbEncryptionInterceptor`
4. Créez un nouveau client AWS DynamoDB SDK.
5. Créez le `DynamoDBEnhancedClient` et créez un tableau avec votre classe modélisée.

Pour plus d'informations sur le client DynamoDB amélioré, [voir créer un](#) client amélioré.

```
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);
}
}
```

Après avoir déployé les modifications de l'étape 2, vous devez rechiffrer tous les anciens éléments de votre tableau avec le nouveau format avant de [passer à l'étape 3](#). Il n'existe pas de métrique ou de requête unique que vous puissiez exécuter pour chiffrer rapidement vos éléments existants. Utilisez le processus le mieux adapté à votre système. Par exemple, vous pouvez utiliser un processus asynchrone qui analyse lentement la table et réécrit les éléments à l'aide des nouvelles actions attributaires et de la nouvelle configuration de chiffrement que vous avez définies.

Étape 3. Lisez et écrivez uniquement les éléments dans le nouveau format

Après avoir rechiffré tous les éléments de votre tableau avec le nouveau format, vous pouvez supprimer le comportement existant de votre configuration. Procédez comme suit pour configurer votre client de manière à ce qu'il lise et écrive uniquement les éléments dans le nouveau format.

Les procédures suivantes fournissent une vue d'ensemble des étapes illustrées dans l'exemple de code ci-dessous.

1. Continuez à configurer votre trousseau de clés, votre schéma de table, `allowedUnsignedAttributes` comme vous l'avez fait à [l'étape 1](#). Supprimez les anciennes actions attributaires et `DynamoDBEncryptor` supprimez-les de votre configuration.
2. Créez un `DynamoDbEncryptionInterceptor`.
3. Créez un nouveau client AWS DynamoDB SDK.
4. Créez le `DynamoDBEnhancedClient` et créez un tableau avec votre classe modélisée.

Pour plus d'informations sur le client DynamoDB amélioré, [voir créer un](#) client amélioré.

```
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

Cette rubrique explique comment installer et utiliser la version 3. x de la bibliothèque de chiffrement côté client .NET pour DynamoDB. Pour plus de détails sur la programmation avec le SDK AWS de chiffrement de base de données pour DynamoDB, consultez les exemples [.NET dans aws-database-encryption-sdk le référentiel -dynamodb](#) sur GitHub

La bibliothèque de chiffrement côté client .NET pour DynamoDB est destinée aux développeurs qui écrivent des applications en C# et dans d'autres langages de programmation .NET. Elle est prise en charge sur Windows, macOS et Linux.

Toutes les implémentations en [langage de programmation](#) du SDK de chiffrement de AWS base de données pour DynamoDB sont interopérables. Toutefois, les valeurs vides SDK pour .NET ne sont pas prises en charge pour les types de données de liste ou de carte. Cela signifie que si vous utilisez la bibliothèque de chiffrement côté client Java pour DynamoDB pour écrire un élément contenant des valeurs vides pour un type de données de liste ou de carte, vous ne pouvez pas déchiffrer et lire cet élément à l'aide de la bibliothèque de chiffrement côté client .NET pour DynamoDB.

Rubriques

- [Installation de la bibliothèque de chiffrement côté client .NET pour DynamoDB](#)
- [Débogage avec .NET](#)
- [Utilisation de la bibliothèque de chiffrement côté client .NET pour DynamoDB](#)
- [Exemples .NET](#)
- [Configurer une table DynamoDB existante pour utiliser AWS le SDK de chiffrement de base de données pour DynamoDB](#)

Installation de la bibliothèque de chiffrement côté client .NET pour DynamoDB

La bibliothèque de chiffrement côté client .NET pour DynamoDB est disponible sous le nom de [AWS.Cryptography.DbEncryptionSDK.DynamoDb](#) dans NuGet. Pour plus de détails sur l'installation et la création de la bibliothèque, consultez le fichier [.NET README.md](#) dans le aws-database-encryption-sdk référentiel -dynamodb. La bibliothèque de chiffrement côté client .NET pour DynamoDB nécessite les clés SDK pour .NET même si vous n'utilisez pas (). AWS Key Management Service AWS KMS Le SDK pour .NET est installé avec le NuGet package.

Version 3. x de la bibliothèque de chiffrement côté client .NET pour DynamoDB prend en charge .NET 6.0 et .NET Framework net48 et versions ultérieures.

Débogage avec .NET

La bibliothèque de chiffrement côté client .NET pour DynamoDB ne génère aucun journal. Les exceptions de la bibliothèque de chiffrement côté client .NET pour DynamoDB génèrent un message d'exception, mais aucune trace de pile.

Pour vous aider à déboguer, veillez à activer la connexion au SDK pour .NET. Les journaux et les messages d'erreur du SDK pour .NET peuvent vous aider à distinguer les erreurs survenant dans la bibliothèque de chiffrement côté client .NET pour DynamoDB SDK pour .NET de celles qui se produisent dans la bibliothèque de chiffrement .NET. Pour obtenir de l'aide SDK pour .NET concernant la journalisation, consultez [AWSLogging](#) le guide du AWS SDK pour .NET développeur. (Pour consulter le sujet, développez la section Ouvrir pour afficher le contenu du .NET Framework.)

Utilisation de la bibliothèque de chiffrement côté client .NET pour DynamoDB

Cette rubrique décrit certaines des fonctions et classes d'assistance de la version 3. x de la bibliothèque de chiffrement côté client .NET pour DynamoDB.

Pour plus de détails sur la programmation avec la bibliothèque de chiffrement côté client .NET pour DynamoDB, consultez les [exemples .NET](#) dans le référentiel -dynamodb sur. aws-database-encryption-sdk GitHub

Rubriques

- [Chiffreurs d'éléments](#)
- [Actions relatives aux attributs dans le SDK de chiffrement AWS de base de données pour DynamoDB](#)

- [Configuration du chiffrement dans le SDK de chiffrement AWS de base de données pour DynamoDB](#)
- [Mise à jour d'éléments avec le SDK AWS de chiffrement de base de données](#)

Chiffreurs d'éléments

À la base, le SDK de chiffrement AWS de base de données pour DynamoDB est un outil de chiffrement d'éléments. Vous pouvez utiliser la version 3. x de la bibliothèque de chiffrement côté client .NET pour DynamoDB afin de chiffrer, signer, vérifier et déchiffrer les éléments de votre table DynamoDB de la manière suivante.

Le SDK de chiffrement de AWS base de données de bas niveau pour l'API DynamoDB

Vous pouvez utiliser votre [configuration de chiffrement de table](#) pour créer un client DynamoDB qui chiffre et signe automatiquement les éléments côté client avec vos requêtes DynamoDB. `PutItem` Vous pouvez utiliser ce client directement ou créer un modèle de [document ou un modèle](#) de [persistance d'objet](#).

[Vous devez utiliser le SDK de chiffrement de AWS base de données de bas niveau pour l'API DynamoDB afin d'utiliser le chiffrement consultable.](#)

Le niveau inférieur `DynamoDbItemEncryptor`

Le niveau inférieur chiffre, signe ou déchiffre et vérifie `DynamoDbItemEncryptor` directement les éléments de votre table sans appeler DynamoDB. Il n'émet pas de DynamoDB ni de `PutItem` requêtes `GetItem`. Par exemple, vous pouvez utiliser le niveau inférieur `DynamoDbItemEncryptor` pour déchiffrer et vérifier directement un élément DynamoDB que vous avez déjà récupéré. Si vous utilisez le niveau inférieur `DynamoDbItemEncryptor`, nous vous recommandons d'utiliser le [modèle de programmation de bas niveau](#) SDK pour .NET fourni pour communiquer avec DynamoDB.

Le niveau inférieur `DynamoDbItemEncryptor` ne prend pas en charge le chiffrement [consultable](#).

Actions relatives aux attributs dans le SDK de chiffrement AWS de base de données pour DynamoDB

Les [actions d'attribut](#) déterminent quelles valeurs d'attribut sont cryptées et signées, lesquelles sont uniquement signées, lesquelles sont signées et incluses dans le contexte de chiffrement, et lesquelles sont ignorées.

Pour spécifier des actions d'attribut avec le client .NET, définissez manuellement les actions d'attribut à l'aide d'un modèle d'objet. Spécifiez vos actions d'attribut en créant un `Dictionary` objet dans lequel les paires nom-valeur représentent les noms d'attributs et les actions spécifiées.

Spécifiez `ENCRYPT_AND_SIGN` le chiffrement et la signature d'un attribut.

Spécifiez `SIGN_ONLY` pour signer un attribut, mais pas pour le chiffrer. Spécifiez

`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` de signer un attribut et de l'inclure dans le contexte de chiffrement. Vous ne pouvez pas chiffrer un attribut sans le signer également. Spécifiez

`DO_NOTHING` pour ignorer un attribut.

Les attributs de partition et de tri doivent être `SIGN_ONLY`

soit `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Si vous définissez des attributs

comme `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, les attributs de partition et de tri doivent également l'être `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

Note

Après avoir défini vos actions d'attribut, vous devez définir quels attributs sont exclus des signatures. Pour faciliter l'ajout de nouveaux attributs non signés à l'avenir, nous vous recommandons de choisir un préfixe distinct (tel que : « ») pour identifier vos attributs non signés. Incluez ce préfixe dans le nom de tous les attributs marqués lorsque vous `DO_NOTHING` définissez votre schéma DynamoDB et vos actions d'attribut.

Le modèle d'objet suivant montre comment spécifier `ENCRYPT_AND_SIGN`,

`SIGN_ONLY` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, et `DO_NOTHING` attribuer des actions avec le client .NET. Cet exemple utilise le préfixe « : » pour identifier les `DO_NOTHING` attributs.

Note

Pour utiliser l'action `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` cryptographique, vous devez utiliser la version 3.3 ou ultérieure du SDK AWS Database Encryption. Déployez la nouvelle version sur tous les lecteurs avant de [mettre à jour votre modèle de données](#) pour l'inclure `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
};
```

Configuration du chiffrement dans le SDK de chiffrement AWS de base de données pour DynamoDB

Lorsque vous utilisez le SDK AWS Database Encryption, vous devez définir explicitement une configuration de chiffrement pour votre table DynamoDB. Les valeurs requises dans votre configuration de chiffrement varient selon que vous avez défini vos actions attributaires manuellement ou à l'aide d'une classe de données annotée.

L'extrait suivant définit une configuration de chiffrement de table DynamoDB à l'aide du SDK de chiffrement de AWS base de données de bas niveau pour l'API DynamoDB et des attributs non signés autorisés définis par un préfixe distinct.

```
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);
```

Nom de table logique

Nom de table logique pour votre table DynamoDB.

Le nom de table logique est lié de manière cryptographique à toutes les données stockées dans la table afin de simplifier les opérations de restauration DynamoDB. Nous vous recommandons vivement de spécifier le nom de votre table DynamoDB comme nom de table logique lorsque vous définissez votre configuration de chiffrement pour la première fois. Vous devez toujours spécifier le même nom de table logique. Pour que le déchiffrement réussisse, le nom de la table logique doit correspondre au nom spécifié lors du chiffrement. Si le nom de votre table DynamoDB change après la [restauration de votre table DynamoDB à partir d'une sauvegarde, le nom logique de la table garantit que l'opération de déchiffrement](#) reconnaît toujours la table.

Attributs non signés autorisés

Les attributs marqués DO_NOTHING dans vos actions d'attributs.

Les attributs non signés autorisés indiquent au client quels attributs sont exclus des signatures. Le client suppose que tous les autres attributs sont inclus dans la signature. Ensuite, lors du déchiffrement d'un enregistrement, le client détermine les attributs qu'il doit vérifier et ceux à ignorer parmi les attributs non signés autorisés que vous avez spécifiés. Vous ne pouvez pas supprimer un attribut de vos attributs non signés autorisés.

Vous pouvez définir explicitement les attributs non signés autorisés en créant un tableau répertoriant tous vos DO_NOTHING attributs. Vous pouvez également spécifier un préfixe distinct lorsque vous nommez vos DO_NOTHING attributs et utiliser le préfixe pour indiquer au client quels attributs ne sont pas signés. Nous vous recommandons vivement de spécifier un préfixe distinct, car cela simplifie le processus d'ajout d'un nouvel DO_NOTHING attribut à l'avenir. Pour de plus amples informations, veuillez consulter [Mettre à jour votre modèle de données](#).

Si vous ne spécifiez pas de préfixe pour tous les DO_NOTHING attributs, vous pouvez configurer un `allowedUnsignedAttributes` tableau répertoriant explicitement tous les attributs que le client doit s'attendre à voir non signés lorsqu'il les rencontre lors du déchiffrement. Vous ne devez définir explicitement vos attributs non signés autorisés que si cela est absolument nécessaire.

Configuration de la recherche (facultatif)

`SearchConfig` Définit la [version de la balise](#).

Le `SearchConfig` doit être spécifié pour utiliser un [chiffrement consultable](#) ou des [balises signées](#).

Suite d'algorithmes (facultatif)

`algorithmSuiteId` Définit la suite d'algorithmes utilisée par le SDK AWS de chiffrement de base de données.

À moins que vous ne spécifiez explicitement une suite d'algorithmes alternative, le SDK AWS de chiffrement de base de données utilise la [suite d'algorithmes par défaut](#). [La suite d'algorithmes par défaut utilise l'algorithme AES-GCM avec dérivation de clés, signatures numériques et engagement de clés](#). Bien que la suite d'algorithmes par défaut soit susceptible de convenir à la plupart des applications, vous pouvez choisir une autre suite d'algorithmes. Par exemple, certains modèles de confiance seraient satisfaits par une suite d'algorithmes sans signature numérique. Pour plus d'informations sur les suites d'algorithmes prises en charge par le SDK AWS de chiffrement de base de données, consultez [Suites d'algorithmes prises en charge dans le SDK AWS de chiffrement de base de données](#).

Pour sélectionner la [suite d'algorithmes AES-GCM sans signature numérique ECDSA](#), incluez l'extrait suivant dans votre configuration de chiffrement de table.

```
AlgorithmSuiteId =  
DBEAlgorithmSuiteId.ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY_SYMSIG_HMAC_SHA384
```

Mise à jour d'éléments avec le SDK AWS de chiffrement de base de données

Le SDK AWS de chiffrement de base de données ne prend pas en charge [ddb : UpdateItem](#) pour les éléments qui incluent des attributs chiffrés ou signés. Pour mettre à jour un attribut chiffré ou signé, vous devez utiliser [ddb : PutItem](#). Lorsque vous spécifiez la même clé primaire qu'un élément existant dans votre `PutItem` demande, le nouvel article remplace complètement l'élément existant. Vous pouvez également utiliser [CLOBBER](#) pour effacer et remplacer tous les attributs lors de la sauvegarde après avoir mis à jour vos articles.

Exemples .NET

Les exemples suivants montrent comment utiliser la bibliothèque de chiffrement côté client .NET pour DynamoDB afin de protéger les éléments de table de votre application. Pour trouver d'autres exemples (et apporter les vôtres), consultez les [exemples .NET](#) dans le référentiel `aws-database-encryption-sdk -dynamodb` sur GitHub.

Les exemples suivants montrent comment configurer la bibliothèque de chiffrement côté client .NET pour DynamoDB dans une nouvelle table Amazon DynamoDB non remplie. Si vous souhaitez

configurer vos tables Amazon DynamoDB existantes pour le chiffrement côté client, consultez.

[Ajouter la version 3.x à une table existante](#)

Rubriques

- [Utilisation du SDK de chiffrement de AWS base de données de bas niveau pour l'API DynamoDB](#)
- [Utiliser le niveau inférieur DynamoDbItemEncryptor](#)

Utilisation du SDK de chiffrement de AWS base de données de bas niveau pour l'API DynamoDB

L'exemple suivant montre comment utiliser le SDK de chiffrement de AWS base de données de bas niveau pour l'API DynamoDB avec un jeu de [AWS KMS clés](#) afin de chiffrer et de signer automatiquement des éléments côté client avec vos demandes DynamoDB. PutItem

Vous pouvez utiliser n'importe quel [porte-clés](#) compatible, mais nous vous recommandons d'utiliser l'un des AWS KMS porte-clés dans la mesure du possible.

Voir l'exemple de code complet : [BasicPutGetExample.cs](#)

Étape 1 : Création du AWS KMS porte-clés

L'exemple suivant permet de `CreateAwsKmsMrkMultiKeyring` créer un AWS KMS trousseau de clés avec une clé KMS de chiffrement symétrique. Le `CreateAwsKmsMrkMultiKeyring` procédé garantit que le trousseau de clés gère correctement les clés à région unique et à zones multiples.

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());
var keyringInput = new CreateAwsKmsMrkMultiKeyringInput { Generator = kmsKeyId };
var kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

Étape 2 : configurer les actions de vos attributs

L'exemple suivant définit un `attributeActionsOnEncrypt` dictionnaire qui représente des exemples d'[actions attributaires](#) pour un élément de table.

Note

L'exemple suivant ne définit aucun attribut en tant que `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Si vous spécifiez des

`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` attributs, les attributs de partition et de tri doivent également l'être `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
};
```

Étape 3 : définir les attributs exclus des signatures

L'exemple suivant suppose que tous les `DO_NOTHING` attributs partagent le préfixe distinct : « » et utilise le préfixe pour définir les attributs non signés autorisés. Le client suppose que tout nom d'attribut avec le préfixe : « » est exclu des signatures. Pour de plus amples informations, veuillez consulter [Allowed unsigned attributes](#).

```
const String unsignAttrPrefix = ":";
```

Étape 4 : définir la configuration du chiffrement des tables DynamoDB

L'exemple suivant définit une `tableConfigs` carte qui représente la configuration de chiffrement pour cette table DynamoDB.

[Cet exemple indique le nom de la table DynamoDB comme nom de table logique](#). Nous vous recommandons vivement de spécifier le nom de votre table DynamoDB comme nom de table logique lorsque vous définissez votre configuration de chiffrement pour la première fois. Pour de plus amples informations, veuillez consulter [Configuration du chiffrement dans le SDK de chiffrement AWS de base de données pour DynamoDB](#).

Note

Pour utiliser le [chiffrement consultable](#) ou les [balises signées](#), vous devez également les inclure [SearchConfig](#) dans votre configuration de chiffrement.

```
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 = unsignedAttrPrefix
};
tableConfigs.Add(ddbTableName, config);
```

Étape 5 : créer un nouveau client DynamoDB du AWS SDK

L'exemple suivant crée un nouveau client DynamoDB du AWS SDK à l'aide **TableEncryptionConfigs** de l'étape 4.

```
var ddb = new Client.DynamoDbClient(
    new DynamoDbTablesEncryptionConfig { TableEncryptionConfigs = tableConfigs });
```

Étape 6 : Chiffrer et signer un élément de table DynamoDB

L'exemple suivant définit un item dictionnaire qui représente un exemple d'élément de table et place cet élément dans la table DynamoDB. L'élément est chiffré et signé côté client avant d'être envoyé à 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);
```

Utiliser le niveau inférieur **DynamoDbItemEncryptor**

L'exemple suivant montre comment utiliser le niveau inférieur `DynamoDbItemEncryptor` avec un [AWS KMS trousseau de clés](#) pour chiffrer et signer directement des éléments de table. L'élément `DynamoDbItemEncryptor` n'est pas placé dans votre table DynamoDB.

Vous pouvez utiliser n'importe quel trousseau de [clés](#) compatible avec le client DynamoDB amélioré, mais nous vous recommandons d'utiliser l'un des trousseaux de clés dans la mesure du AWS KMS possible.

Note

Le niveau inférieur `DynamoDbItemEncryptor` ne prend pas en charge le chiffrement [consultable](#). Utilisez le SDK de chiffrement de AWS base de données de bas niveau pour l'API DynamoDB afin d'utiliser le chiffrement consultable.

Voir l'exemple de code complet : [ItemEncryptDecryptExample.cs](#)

Étape 1 : Création du AWS KMS porte-clés

L'exemple suivant permet de `CreateAwsKmsMrkMultiKeyring` créer un AWS KMS trousseau de clés avec une clé KMS de chiffrement symétrique. Le `CreateAwsKmsMrkMultiKeyring` procédé garantit que le trousseau de clés gère correctement les clés à région unique et à zones multiples.

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());  
var keyringInput = new CreateAwsKmsMrkMultiKeyringInput { Generator = kmsKeyId };  
var kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

Étape 2 : configurer les actions de vos attributs

L'exemple suivant définit un `attributeActionsOnEncrypt` dictionnaire qui représente des exemples d'[actions attributaires](#) pour un élément de table.

Note

L'exemple suivant ne définit aucun attribut en tant que `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Si vous spécifiez des `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` attributs, les attributs de partition et de tri doivent également l'être `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
};
```

Étape 3 : définir les attributs exclus des signatures

L'exemple suivant suppose que tous les `DO_NOTHING` attributs partagent le préfixe distinct : « » et utilise le préfixe pour définir les attributs non signés autorisés. Le client suppose que tout nom d'attribut avec le préfixe : « » est exclu des signatures. Pour de plus amples informations, veuillez consulter [Allowed unsigned attributes](#).

```
String unsignAttrPrefix = ":";
```

Étape 4 : Définition de la `DynamoDbItemEncryptor` configuration

L'exemple suivant définit la configuration de `DynamoDbItemEncryptor`.

[Cet exemple indique le nom de la table DynamoDB comme nom de table logique](#). Nous vous recommandons vivement de spécifier le nom de votre table DynamoDB comme nom de table logique lorsque vous définissez votre configuration de chiffrement pour la première fois. Pour de plus amples informations, veuillez consulter [Configuration du chiffrement dans le SDK de chiffrement AWS de base de données pour DynamoDB](#).

```
var config = new DynamoDbItemEncryptorConfig
{
```

```
LogicalTableName = ddbTableName,  
PartitionKeyName = "partition_key",  
SortKeyName = "sort_key",  
AttributeActionsOnEncrypt = attributeActionsOnEncrypt,  
Keyring = kmsKeyring,  
AllowedUnsignedAttributePrefix = unsignAttrPrefix  
};
```

Étape 5 : Création du `DynamoDbItemEncryptor`

L'exemple suivant en crée un nouveau à `DynamoDbItemEncryptor` l'aide de l'étape 4 config à partir de l'étape 4.

```
var itemEncryptor = new DynamoDbItemEncryptor(config);
```

Étape 6 : Chiffrer et signer directement un élément du tableau

L'exemple suivant chiffre et signe directement un élément à l'aide du `DynamoDbItemEncryptor`. L'élément `DynamoDbItemEncryptor` n'est pas placé dans votre table DynamoDB.

```
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;
```

Configurer une table DynamoDB existante pour utiliser AWS le SDK de chiffrement de base de données pour DynamoDB

Avec la version 3. x de la bibliothèque de chiffrement côté client .NET pour DynamoDB, vous pouvez configurer vos tables Amazon DynamoDB existantes pour le chiffrement côté client. Cette rubrique fournit des conseils sur les trois étapes à suivre pour ajouter la version 3. x vers une table DynamoDB existante et renseignée.

Étape 1 : Préparez-vous à lire et à écrire des éléments chiffrés

Procédez comme suit pour préparer votre client du SDK AWS Database Encryption à lire et à écrire des éléments chiffrés. Après avoir déployé les modifications suivantes, votre client continuera à lire et à écrire des éléments en texte brut. Il ne chiffrera ni ne signera aucun nouvel élément inscrit dans la table, mais il sera en mesure de déchiffrer les éléments chiffrés dès leur apparition. Ces modifications préparent le client à commencer à [chiffrer de nouveaux éléments](#). Les modifications suivantes doivent être déployées sur chaque lecteur avant de passer à l'étape suivante.

1. Définissez vos [actions attributaires](#)

Créez un modèle d'objet pour définir les valeurs d'attribut qui seront cryptées et signées, celles qui seront uniquement signées et celles qui seront ignorées.

Par défaut, les attributs de clé primaire sont signés mais pas chiffrés (SIGN_ONLY) et tous les autres attributs sont chiffrés et signés (ENCRYPT_AND_SIGN).

Spécifiez ENCRYPT_AND_SIGN le chiffrement et la signature d'un attribut.

Spécifiez SIGN_ONLY pour signer un attribut, mais pas pour le chiffrer. Spécifiez

SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT à signer et à attribuer et à inclure dans le contexte de chiffrement. Vous ne pouvez pas chiffrer un attribut sans le signer également.

Spécifiez DO_NOTHING si un attribut doit être ignoré. Pour de plus amples informations, veuillez consulter [Actions relatives aux attributs dans le SDK de chiffrement AWS de base de données pour DynamoDB](#).

Note

Si vous spécifiez des SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT attributs, les attributs de partition et de tri doivent également l'être 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. Définissez les attributs qui seront exclus des signatures

L'exemple suivant suppose que tous les `DO_NOTHING` attributs partagent le préfixe distinct : « » et utilise le préfixe pour définir les attributs non signés autorisés. Le client supposera que tout nom d'attribut avec le préfixe : « » est exclu des signatures. Pour de plus amples informations, veuillez consulter [Allowed unsigned attributes](#).

```
const String unsignAttrPrefix = ":";
```

3. Créez un [porte-clés](#)

L'exemple suivant crée un [AWS KMS trousseau de clés](#). Le AWS KMS trousseau de clés utilise le chiffrement symétrique ou le RSA asymétrique AWS KMS keys pour générer, chiffrer et déchiffrer les clés de données.

Cet exemple permet `CreateMrkMultiKeyring` de créer un AWS KMS trousseau de clés avec une clé KMS de chiffrement symétrique. La `CreateAwsKmsMrkMultiKeyring` méthode garantit que le trousseau de clés gère correctement les clés à région unique et à régions multiples.

```
var matProv = new MaterialProviders(new MaterialProvidersConfig());  
var keyringInput = new CreateAwsKmsMrkMultiKeyringInput { Generator = kmsKeyId };  
var kmsKeyring = matProv.CreateAwsKmsMrkMultiKeyring(keyringInput);
```

4. Définition de la configuration du chiffrement des tables DynamoDB

L'exemple suivant définit une `tableConfigs` carte qui représente la configuration de chiffrement pour cette table DynamoDB.

[Cet exemple indique le nom de la table DynamoDB comme nom de table logique](#). Nous vous recommandons vivement de spécifier le nom de votre table DynamoDB comme nom de table logique lorsque vous définissez votre configuration de chiffrement pour la première fois.

Vous devez le spécifier `FORCE_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT` comme remplacement en texte brut. Cette politique continue de lire et d'écrire des éléments en texte brut, de lire des éléments chiffrés et de préparer le client à écrire des éléments chiffrés.

Pour plus d'informations sur les valeurs incluses dans la configuration du chiffrement des tables, consultez [Configuration du chiffrement dans le SDK de chiffrement AWS de base de données pour 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 = unsignedAttrPrefix,
    PlaintextOverride = FORCE_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT
};
tableConfigs.Add(ddbTableName, config);
```

5. Création d'un nouveau client AWS DynamoDB SDK

L'exemple suivant crée un nouveau client DynamoDB AWS SDK à l'aide **TableEncryptionConfigs** de l'étape 4.

```
var ddb = new Client.DynamoDbClient(
    new DynamoDbTablesEncryptionConfig { TableEncryptionConfigs = tableConfigs });
```

Étape 2 : Écrire des éléments chiffrés et signés

Mettez à jour la politique de texte brut dans votre configuration de chiffrement de table pour permettre au client d'écrire des éléments chiffrés et signés. Après avoir déployé la modification suivante, le client chiffre et signe les nouveaux éléments en fonction des actions d'attribut que vous avez configurées à l'étape 1. Le client pourra lire les éléments en texte brut et les éléments chiffrés et signés.

Avant de passer à l'[étape 3](#), vous devez chiffrer et signer tous les éléments en texte brut existants dans votre tableau. Il n'existe pas de métrique ou de requête unique que vous puissiez exécuter pour chiffrer rapidement vos éléments en texte brut existants. Utilisez le processus le mieux adapté à votre système. Par exemple, vous pouvez utiliser un processus asynchrone qui analyse lentement la table et réécrit les éléments à l'aide des actions attributaires et de la configuration de chiffrement que vous avez définies. Pour identifier les éléments en texte brut de votre tableau, nous vous recommandons de rechercher tous les éléments qui ne contiennent pas les `aws_dbe_foot` attributs `aws_dbe_head` et que le SDK de chiffrement de AWS base de données ajoute aux éléments lorsqu'ils sont chiffrés et signés.

L'exemple suivant met à jour la configuration de chiffrement des tables depuis l'étape 1. Vous devez mettre à jour le remplacement en texte brut avec.

`FORBID_WRITE_PLAINTEXT_ALLOW_READ_PLAINTEXT` Cette politique continue de lire les éléments en texte brut, mais également de lire et d'écrire des éléments chiffrés. Créez un nouveau client DynamoDB du AWS SDK à l'aide de la version mise à jour. `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);
```

Étape 3 : lire uniquement les éléments chiffrés et signés

Une fois que vous avez chiffré et signé tous vos éléments, mettez à jour la dérogation en texte brut dans votre configuration de chiffrement de table pour autoriser uniquement le client à lire et écrire des éléments chiffrés et signés. Après avoir déployé la modification suivante, le client chiffre et signe les nouveaux éléments en fonction des actions d'attribut que vous avez configurées à l'étape 1. Le client ne pourra lire que les éléments chiffrés et signés.

L'exemple suivant met à jour la configuration de chiffrement des tables depuis l'étape 2. Vous pouvez soit mettre à jour la règle de remplacement en texte brut, `FORBID_WRITE_PLAINTEXT_FORBID_READ_PLAINTEXT` soit la supprimer de votre configuration. Le client lit et écrit uniquement les éléments chiffrés et signés par défaut. Créez un nouveau client DynamoDB du AWS SDK à l'aide de la version mise à jour. `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

Cette rubrique explique comment installer et utiliser la version 1. x de la bibliothèque de chiffrement côté client Rust pour DynamoDB. Pour plus de détails sur la programmation avec le SDK AWS de chiffrement de base de données pour DynamoDB, consultez les exemples [Rust dans aws-database-encryption-sdk le référentiel -dynamodb](#) sur. GitHub

Toutes les implémentations en langage de programmation du SDK de chiffrement de AWS base de données pour DynamoDB sont interopérables.

Rubriques

- [Conditions préalables](#)
- [Installation](#)
- [Utilisation de la bibliothèque de chiffrement côté client Rust pour DynamoDB](#)

Conditions préalables

Avant d'installer la bibliothèque de chiffrement côté client Rust pour DynamoDB, assurez-vous de remplir les conditions préalables suivantes.

Installez Rust and Cargo

Installez la version stable actuelle de [Rust](#) en utilisant [rustup](#).

Pour plus d'informations sur le téléchargement et l'installation de rustup, consultez les [procédures d'installation dans The Cargo Book](#).

Installation

La bibliothèque de chiffrement Rust côté client pour DynamoDB est disponible sous forme de caisse sur Crates.io. [aws-db-esdk](#) Pour plus de détails sur l'installation et la création de la bibliothèque, consultez le fichier [README.md](#) dans le aws-database-encryption-sdk référentiel -dynamodb. GitHub

Manuellement

[Pour installer la bibliothèque de chiffrement côté client Rust pour DynamoDB, clonez ou téléchargez le référentiel -dynamodb. aws-database-encryption-sdk](#) GitHub

Pour installer la dernière version

Exécutez la commande Cargo suivante dans le répertoire de votre projet :

```
cargo add aws-db-esdk
```

Ou ajoutez la ligne suivante à votre Cargo.toml :

```
aws-db-esdk = "<version>"
```

Utilisation de la bibliothèque de chiffrement côté client Rust pour DynamoDB

Cette rubrique décrit certaines des fonctions et classes d'assistance de la version 1. x de la bibliothèque de chiffrement côté client Rust pour DynamoDB.

Pour plus de détails sur la programmation avec la bibliothèque de chiffrement côté client Rust pour DynamoDB, consultez les [exemples Rust](#) dans le référentiel -dynamodb sur. aws-database-encryption-sdk GitHub

Rubriques

- [Chiffreurs d'éléments](#)
- [Actions relatives aux attributs dans le SDK de chiffrement AWS de base de données pour DynamoDB](#)
- [Configuration du chiffrement dans le SDK de chiffrement AWS de base de données pour DynamoDB](#)
- [Mise à jour d'éléments avec le SDK AWS de chiffrement de base de données](#)

Chiffreurs d'éléments

À la base, le SDK de chiffrement AWS de base de données pour DynamoDB est un outil de chiffrement d'éléments. Vous pouvez utiliser la version 1. x de la bibliothèque de chiffrement côté client Rust pour DynamoDB pour chiffrer, signer, vérifier et déchiffrer les éléments de votre table DynamoDB de la manière suivante.

Le SDK de chiffrement de AWS base de données de bas niveau pour l'API DynamoDB

Vous pouvez utiliser votre [configuration de chiffrement de table](#) pour créer un client DynamoDB qui chiffre et signe automatiquement les éléments côté client avec vos requêtes DynamoDB.

PutItem

[Vous devez utiliser le SDK de chiffrement de AWS base de données de bas niveau pour l'API DynamoDB afin d'utiliser le chiffrement consultable.](#)

Pour un exemple illustrant comment utiliser le SDK de chiffrement de AWS base de données de bas niveau pour l'API DynamoDB, consultez [basic_get_put_example.rs](#) dans le référentiel - dynamodb sur. aws-database-encryption-sdk GitHub

Le niveau inférieur **DynamoDbItemEncryptor**

Le niveau inférieur chiffre, signe ou déchiffre et vérifie `DynamoDbItemEncryptor` directement les éléments de votre table sans appeler DynamoDB. Il n'émet pas de DynamoDB ni de `PutItem` requêtes `GetItem`. Par exemple, vous pouvez utiliser le niveau inférieur `DynamoDbItemEncryptor` pour déchiffrer et vérifier directement un élément DynamoDB que vous avez déjà récupéré.

Le niveau inférieur `DynamoDbItemEncryptor` ne prend pas en charge le chiffrement [consultable](#).

Pour un exemple illustrant comment utiliser le niveau inférieur `DynamoDbItemEncryptor`, consultez [item_encrypt_decrypt.rs](#) dans le référentiel -dynamodb sur. aws-database-encryption-sdk GitHub

Actions relatives aux attributs dans le SDK de chiffrement AWS de base de données pour DynamoDB

Les [actions d'attribut](#) déterminent quelles valeurs d'attribut sont cryptées et signées, lesquelles sont uniquement signées, lesquelles sont signées et incluses dans le contexte de chiffrement, et lesquelles sont ignorées.

Pour spécifier des actions d'attribut avec le client Rust, définissez manuellement les actions d'attribut à l'aide d'un modèle d'objet. Spécifiez vos actions d'attribut en créant un HashMap objet dans lequel les paires nom-valeur représentent les noms d'attributs et les actions spécifiées.

Spécifiez `ENCRYPT_AND_SIGN` le chiffrement et la signature d'un attribut.

Spécifiez `SIGN_ONLY` pour signer un attribut, mais pas pour le chiffrer. Spécifiez

`SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT` de signer un attribut et de l'inclure dans le

contexte de chiffrement. Vous ne pouvez pas chiffrer un attribut sans le signer également. Spécifiez

`DO_NOTHING` si un attribut doit être ignoré.

Les attributs de partition et de tri doivent être l'un `SIGN_ONLY` ou

l'autre `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`. Si vous définissez des attributs

comme `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, les attributs de partition et de tri doivent

également l'être `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`.

Note

Après avoir défini vos actions d'attribut, vous devez définir quels attributs sont exclus des signatures. Pour faciliter l'ajout de nouveaux attributs non signés à l'avenir, nous vous recommandons de choisir un préfixe distinct (tel que : « ») pour identifier vos attributs non signés. Incluez ce préfixe dans le nom d'attribut pour tous les attributs marqués lorsque vous `DO_NOTHING` définissez votre schéma DynamoDB et vos actions d'attribut.

Le modèle d'objet suivant montre comment spécifier `ENCRYPT_AND_SIGN`,

`SIGN_ONLY` `SIGN_AND_INCLUDE_IN_ENCRYPTION_CONTEXT`, et `DO_NOTHING` attribuer des

actions avec le client Rust. Cet exemple utilise le préfixe « : » pour identifier les `DO_NOTHING` attributs.

```
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),
]);
```

Configuration du chiffrement dans le SDK de chiffrement AWS de base de données pour DynamoDB

Lorsque vous utilisez le SDK AWS Database Encryption, vous devez définir explicitement une configuration de chiffrement pour votre table DynamoDB. Les valeurs requises dans votre configuration de chiffrement varient selon que vous avez défini vos actions attributaires manuellement ou à l'aide d'une classe de données annotée.

L'extrait suivant définit une configuration de chiffrement de table DynamoDB à l'aide du SDK de chiffrement de AWS base de données de bas niveau pour l'API DynamoDB et des attributs non signés autorisés définis par un préfixe distinct.

```
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()?;
```

Nom de table logique

Nom de table logique pour votre table DynamoDB.

Le nom de table logique est lié de manière cryptographique à toutes les données stockées dans la table afin de simplifier les opérations de restauration DynamoDB. Nous vous recommandons vivement de spécifier le nom de votre table DynamoDB comme nom de table logique lorsque vous définissez votre configuration de chiffrement pour la première fois. Vous devez toujours spécifier le même nom de table logique. Pour que le déchiffrement réussisse, le nom de la table logique doit correspondre au nom spécifié lors du chiffrement. Si le nom de votre table DynamoDB change après la [restauration de votre table DynamoDB à partir d'une sauvegarde, le nom logique de la table garantit que l'opération de déchiffrement](#) reconnaît toujours la table.

Attributs non signés autorisés

Les attributs marqués `DO_NOTHING` dans vos actions d'attributs.

Les attributs non signés autorisés indiquent au client quels attributs sont exclus des signatures. Le client suppose que tous les autres attributs sont inclus dans la signature. Ensuite, lors du déchiffrement d'un enregistrement, le client détermine les attributs qu'il doit vérifier et ceux à ignorer parmi les attributs non signés autorisés que vous avez spécifiés. Vous ne pouvez pas supprimer un attribut de vos attributs non signés autorisés.

Vous pouvez définir explicitement les attributs non signés autorisés en créant un tableau répertoriant tous vos `DO_NOTHING` attributs. Vous pouvez également spécifier un préfixe distinct lorsque vous nommez vos `DO_NOTHING` attributs et utiliser le préfixe pour indiquer au client quels attributs ne sont pas signés. Nous vous recommandons vivement de spécifier un préfixe distinct, car cela simplifie le processus d'ajout d'un nouvel `DO_NOTHING` attribut à l'avenir. Pour de plus amples informations, veuillez consulter [Mettre à jour votre modèle de données](#).

Si vous ne spécifiez pas de préfixe pour tous les `DO_NOTHING` attributs, vous pouvez configurer un `allowedUnsignedAttributes` tableau répertoriant explicitement tous les attributs que le client doit s'attendre à voir non signés lorsqu'il les rencontre lors du déchiffrement. Vous ne devez définir explicitement vos attributs non signés autorisés que si cela est absolument nécessaire.

Configuration de la recherche (facultatif)

`SearchConfig` Définit la [version de la balise](#).

Le `SearchConfig` doit être spécifié pour utiliser un [chiffrement consultable](#) ou des [balises signées](#).

Suite d'algorithmes (facultatif)

`algorithmSuiteId` Définit la suite d'algorithmes utilisée par le SDK AWS de chiffrement de base de données.

À moins que vous ne spécifiez explicitement une suite d'algorithmes alternative, le SDK AWS de chiffrement de base de données utilise la [suite d'algorithmes par défaut](#). [La suite d'algorithmes par défaut utilise l'algorithme AES-GCM avec dérivation de clés, signatures numériques et engagement de clés](#). Bien que la suite d'algorithmes par défaut soit susceptible de convenir à la plupart des applications, vous pouvez choisir une autre suite d'algorithmes. Par exemple, certains modèles de confiance seraient satisfaits par une suite d'algorithmes sans signature

numérique. Pour plus d'informations sur les suites d'algorithmes prises en charge par le SDK AWS de chiffrement de base de données, consultez [Suites d'algorithmes prises en charge dans le SDK AWS de chiffrement de base de données](#).

Pour sélectionner la [suite d'algorithmes AES-GCM sans signature numérique ECDSA](#), incluez l'extrait suivant dans votre configuration de chiffrement de table.

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

Mise à jour d'éléments avec le SDK AWS de chiffrement de base de données

Le SDK AWS de chiffrement de base de données ne prend pas en charge [ddb : UpdateItem](#) pour les éléments qui incluent des attributs chiffrés ou signés. Pour mettre à jour un attribut chiffré ou signé, vous devez utiliser [ddb : PutItem](#). Lorsque vous spécifiez la même clé primaire qu'un élément existant dans votre PutItem demande, le nouvel élément remplace complètement l'élément existant.

Ancien client de chiffrement DynamoDB

Le 9 juin 2023, notre bibliothèque de chiffrement côté client a été renommée AWS Database Encryption SDK. Le SDK AWS de chiffrement de base de données continue de prendre en charge les anciennes versions du client de chiffrement DynamoDB. Pour plus d'informations sur les différentes parties de la bibliothèque de chiffrement côté client qui ont changé avec le changement de nom, consultez [Changement du nom du client de chiffrement Amazon DynamoDB](#)

Pour migrer vers la dernière version de la bibliothèque de chiffrement côté client Java pour DynamoDB, consultez [Migrer vers la version 3.x](#)

Rubriques

- [AWS SDK de chiffrement de base de données pour la prise en charge des versions DynamoDB](#)
- [Fonctionnement du client de chiffrement DynamoDB](#)
- [Concepts du client de chiffrement Amazon DynamoDB](#)
- [Fournisseur de matériel cryptographique](#)
- [Langages de programmation disponibles pour le client de chiffrement Amazon DynamoDB](#)
- [Modification de votre modèle de données](#)

- [Résolution des problèmes liés à votre application cliente de chiffrement DynamoDB](#)

AWS SDK de chiffrement de base de données pour la prise en charge des versions DynamoDB

Les rubriques du chapitre Legacy fournissent des informations sur les versions 1. x —2. x du client de chiffrement DynamoDB pour Java et versions 1. x —3. x du client de chiffrement DynamoDB pour Python.

Le tableau suivant répertorie les langues et les versions qui prennent en charge le chiffrement côté client dans Amazon DynamoDB.

Langage de programmation	Version	Phase du cycle de vie de la version majeure du SDK
Java	Versions 1. x	End-of-Support phase , en vigueur en juillet 2022
Java	Versions 2. x	Disponibilité générale (GA)
Java	Version 3. x	Disponibilité générale (GA)
Python	Versions 1. x	End-of-Support phase , en vigueur en juillet 2022
Python	Versions 2. x	End-of-Support phase , en vigueur en juillet 2022
Python	Versions 3. x	Disponibilité générale (GA)

Fonctionnement du client de chiffrement DynamoDB

Note

Notre bibliothèque de chiffrement côté client a été [renommée AWS Database Encryption SDK](#). La rubrique suivante fournit des informations sur les versions 1. x —2. x du client de chiffrement DynamoDB pour Java et versions 1. x —3. x du client de chiffrement DynamoDB pour Python.

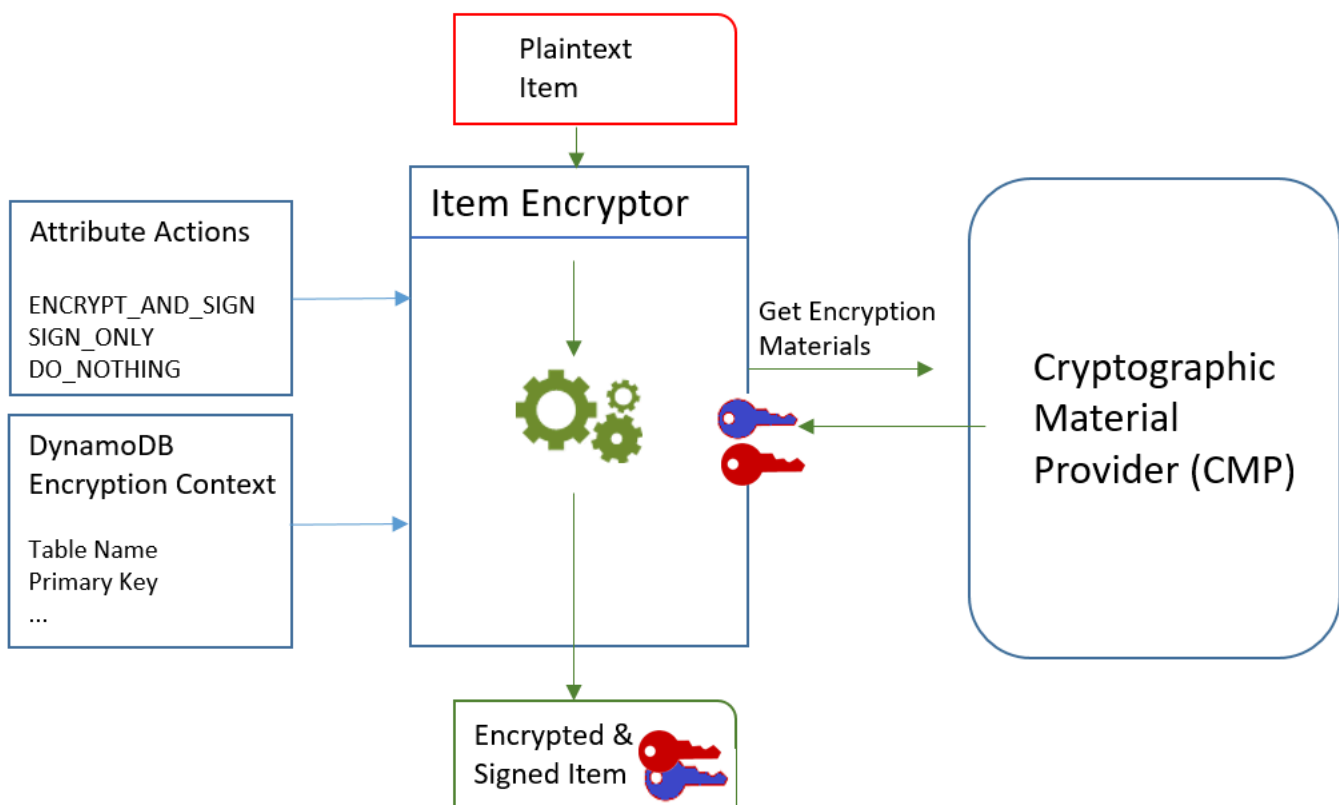
pour Python. Pour plus d'informations, consultez la section [SDK AWS de chiffrement de base de données pour la prise en charge des versions DynamoDB](#).

Le client de chiffrement DynamoDB est spécialement conçu pour protéger les données que vous stockez dans DynamoDB. Les bibliothèques incluent les implémentations sécurisées que vous pouvez étendre ou utiliser inchangées. Et, la plupart des éléments sont représentés par des éléments abstraits afin que vous puissiez créer et utiliser des composants personnalisés compatibles.

Chiffrement et signature des éléments de table


Au cœur du client de chiffrement DynamoDB se trouve un outil de chiffrement d'éléments qui chiffre, signe, vérifie et déchiffre les éléments de table. Il prend les informations sur les éléments de table et les instructions sur les éléments à chiffrer et signer. Il obtient les matériaux de chiffrement et les instructions sur leur utilisation auprès d'un [fournisseur CMP](#) que vous sélectionnez et configurez.

Le schéma suivant illustre un aperçu de haut niveau du processus.



Pour chiffrer et signer un élément de table, le client de chiffrement DynamoDB a besoin des éléments suivants :

- Informations sur le tableau. Il obtient des informations sur la table à partir d'un contexte de [chiffrement DynamoDB](#) que vous fournissez. Certains assistants obtiennent les informations requises auprès de DynamoDB et créent le contexte de chiffrement DynamoDB pour vous.

 Note

Le contexte de chiffrement DynamoDB dans le client de chiffrement DynamoDB n'est pas lié au contexte de chiffrement dans () et le. AWS Key Management Service AWS KMS AWS Encryption SDK

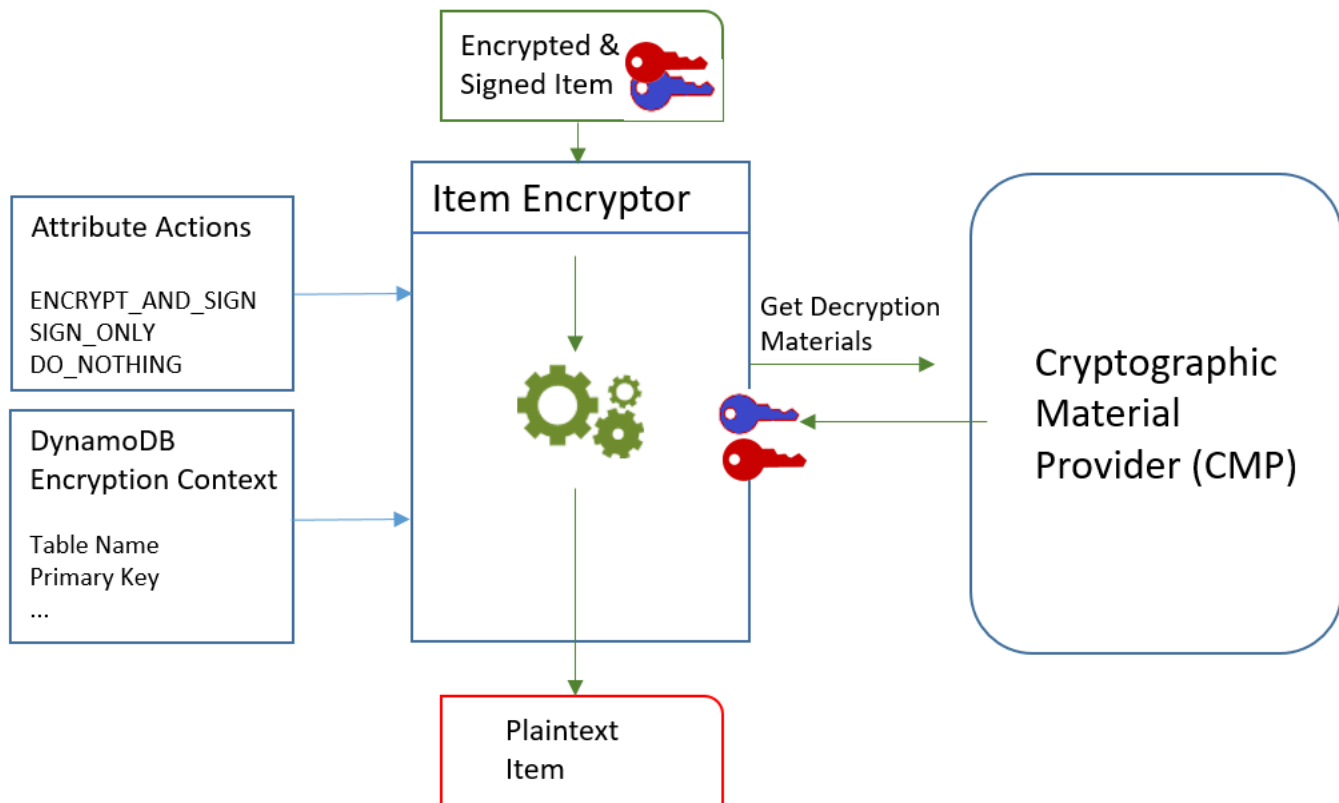
- Attributs à chiffrer et signer. Il obtient ces informations à partir des [actions d'attribut](#) que vous fournissez.
- Matériaux de chiffrement, clés de chiffrement et de signature incluses. Il obtient ces informations auprès d'un [fournisseur CMP](#) que vous sélectionnez et configurez.
- Instructions pour le chiffrement et la signature de l'élément. Le fournisseur CMP ajoute les instructions sur l'utilisation des matériaux de chiffrement, algorithmes de chiffrement et de signature inclus, à la [description du matériau réel](#).

Le [chiffreur d'élément](#) utilise tous ces éléments pour chiffrer et signer l'élément. Le chiffreur d'élément ajoute aussi deux attributs à l'élément : un [attribut de description de matériau](#) qui contient les instructions de chiffrement et de signature (description du matériau réel), et un attribut qui contient la signature. Vous pouvez interagir directement avec le chiffreur d'élément, ou utilisez les fonctions d'annotation qui interagissent avec le chiffreur d'élément pour que vous implémentiez le comportement par défaut sécurisé.

Il en résulte un élément DynamoDB contenant les données chiffrées et signées.

Vérification et déchiffrement des éléments de table

Ces composants fonctionnent aussi ensemble pour vérifier et déchiffrer votre élément, comme illustré dans le schéma suivant.



Pour vérifier et déchiffrer un élément, le client de chiffrement DynamoDB a besoin des mêmes composants, de composants présentant la même configuration ou de composants spécialement conçus pour déchiffrer les éléments, comme suit :

- Informations relatives à la table issues du contexte de [chiffrement DynamoDB](#).
- Attributs à vérifier et à déchiffrer. Il obtient ces informations à partir des [actions d'attribut](#).
- Matériaux de déchiffrement, clés de vérification et de déchiffrement incluses, depuis le [fournisseur CMP](#) que vous sélectionnez et configurez.

L'élément chiffré n'inclut pas d'enregistrement du CMP qui a été utilisé pour le chiffrer. Vous devez fournir le même CMP, un CMP avec la même configuration ou un CMP qui a été conçu pour déchiffrer les éléments.

- Informations sur la façon dont l'élément a été chiffré et signé, algorithmes de chiffrement et de signature inclus. Le client obtient ces informations à partir de l'[attribut de description du matériau](#) de l'élément.

Le [chiffreur d'élément](#) utilise tous ces éléments pour vérifier et déchiffrer l'élément. Il supprime aussi les attribut de description de matériau et de signature. Le résultat est un élément DynamoDB en texte brut.

Concepts du client de chiffrement Amazon DynamoDB

Note

Notre bibliothèque de chiffrement côté client a été [renommée AWS Database Encryption SDK](#). La rubrique suivante fournit des informations sur les versions 1. x —2. x du client de chiffrement DynamoDB pour Java et versions 1. x —3. x du client de chiffrement DynamoDB pour Python. Pour plus d'informations, consultez la section [SDK AWS de chiffrement de base de données pour la prise en charge des versions DynamoDB](#).

Cette rubrique explique les concepts et la terminologie utilisés dans le client de chiffrement Amazon DynamoDB.

Pour savoir comment les composants du client de chiffrement DynamoDB interagissent, consultez [Fonctionnement du client de chiffrement DynamoDB](#)

Rubriques

- [Fournisseur CMP \(Cryptographic Materials Provider\)](#)
- [Chiffreurs d'éléments](#)
- [Actions d'attribut](#)
- [Description du matériau](#)
- [Client de chiffrement DynamoDB](#)
- [Magasin de fournisseur](#)

Fournisseur CMP (Cryptographic Materials Provider)

Lorsque vous implémentez le client de chiffrement DynamoDB, l'une de vos premières tâches consiste [à sélectionner un fournisseur de matériel cryptographique](#) (CMP) (également appelé fournisseur de matériel de chiffrement). Votre choix détermine une grande part du reste de l'implémentation.

Un fournisseur CMP recueille, assemble et retourne les matériaux de chiffrement que le [chiffreur d'élément](#) utilise pour chiffrer et signer vos éléments de table. Le CMP détermine les algorithmes de chiffrement à utiliser, ainsi que la façon de générer et de protéger le chiffrement et les clés de signature.

Le fournisseur CMP interagit avec le chiffreur d'élément. Celui-ci demande les matériaux de chiffrement ou de déchiffrement au CMP, et le CMP les retourne au chiffreur d'élément. Puis, celui-ci utilise les matériaux de chiffrement pour chiffrer et signer, ou vérifier et déchiffrer, l'élément.

Vous spécifiez le CMP lorsque vous configurez le client. Vous pouvez créer un CMP personnalisé compatible ou utiliser l'un des nombreux modèles de CMPs la bibliothèque. La plupart CMPs sont disponibles pour plusieurs langages de programmation.

Chiffreurs d'éléments

Le chiffreur d'éléments est un composant de niveau inférieur qui effectue des opérations de chiffrement pour le client de chiffrement DynamoDB. Il demande les matériaux de chiffrement auprès d'un [fournisseur CMP](#), puis utilise les matériaux retournés par le CMP pour chiffrer et signer, ou vérifier et déchiffrer, votre élément de table.

Vous pouvez interagir directement avec le chiffreur d'élément ou utiliser les annotations fournis par votre bibliothèque. Par exemple, le client de chiffrement DynamoDB pour Java inclut `AttributeEncryptor` une classe d'assistance que vous pouvez utiliser avec `DynamoDBMapper` le chiffreur d'éléments, au lieu d'interagir directement avec celui-ci. `DynamoDBEncryptor` La bibliothèque Python inclut les classes d'annotations `EncryptedTable`, `EncryptedClient` et `EncryptedResource` qui interagissent avec le chiffreur d'éléments pour vous.

Actions d'attribut

Les actions d'attribut informent le chiffreur d'élément des actions à exécuter sur chaque attribut de l'élément.

L'action d'attribut peut avoir l'une des valeurs suivantes :

- Chiffrer et signer — Chiffrez la valeur de l'attribut. Incluez l'attribut (nom et valeur) dans la signature de l'élément.
- Signe uniquement — Incluez l'attribut dans la signature de l'article.
- Ne rien faire : ne chiffrez ni ne signez l'attribut.

Pour tout attribut pouvant stocker des données sensibles, utilisez Chiffrer et signer. Pour les attributs de clé primaire (clé de partition et clé de tri), utilisez Signer uniquement. L'[attribut de description de matériau](#) et l'attribut de signature ne sont pas signés ou chiffrés. Vous n'avez pas besoin de spécifier les actions d'attribut pour ces attributs.

Choisissez soigneusement vos actions d'attribut. En cas de doute, utilisez Chiffrer et signer. Une fois que vous avez utilisé le client de chiffrement DynamoDB pour protéger les éléments de votre table, vous ne pouvez pas modifier l'action d'un attribut sans risquer une erreur de validation de signature. Pour en savoir plus, consultez [Modification de votre modèle de données](#).

Warning

Ne chiffrez pas les attributs de la clé primaire. Ils doivent rester en texte brut pour que DynamoDB puisse trouver l'élément sans exécuter une analyse complète du tableau.

Si le contexte de [chiffrement DynamoDB](#) identifie les attributs de votre clé primaire, le client génère une erreur si vous essayez de les chiffrer.

La technique que vous utilisez pour spécifier les actions d'attribut dépend du langage de programmation que vous utilisez. Elle peut aussi être spécifique aux classes d'annotations que vous utilisez.

Pour plus d'informations, consultez la documentation de votre langage de programmation.

- [Python](#)
- [Java](#)

Description du matériau

La description du matériau pour un élément de table chiffré se compose d'informations, telles que les algorithmes de chiffrement, sur la façon dont l'élément de table est chiffré et signé. Le [fournisseur CMP](#) enregistre la description du matériau tandis qu'il rassemble les matériaux de chiffrement pour le chiffrement et la signature. Par la suite, quand il doit rassembler les matériaux de chiffrement pour vérifier et chiffrer l'élément, il utilise la description du matériau comme guide.

Dans le client de chiffrement DynamoDB, la description matérielle fait référence à trois éléments connexes :

Description du matériau demandé

Certains [fournisseurs de matériel cryptographique](#) (CMPs) vous permettent de définir des options avancées, telles qu'un algorithme de chiffrement. Pour indiquer vos choix, vous ajoutez des paires nom-valeur à la propriété de description matérielle du [contexte de chiffrement DynamoDB dans votre demande de chiffrement](#) d'un élément de table. L'élément est appelé description du matériau demandé. Les valeurs autorisées pour la description du matériau demandé sont définies par le fournisseur CMP que vous choisissez.

Note

Comme la description du matériau peut remplacer les valeurs par défaut sécurisées, il est recommandé d'omettre la description du matériau demandé à moins que vous n'ayez une excellente raison de l'utiliser.

Description du matériau réel

La description matérielle renvoyée par les [fournisseurs de matériel cryptographique](#) (CMPs) est connue sous le nom de description matérielle réelle. Elle décrit les valeurs réelles que le CMP a utilisées quand il a rassemblé les matériaux de chiffrement. Elle se compose généralement de la description du matériau demandé, le cas échéant, avec les ajouts et les modifications.

Attribut de description du matériau

Le client enregistre la description du matériau réel dans l'attribut de description du matériau de l'élément chiffré. Le nom de l'attribut de description du matériau est `amzn-ddb-map-desc` et sa valeur est la description du matériau réel. Le client utilise les valeurs de l'attribut de description du matériau pour vérifier et déchiffrer l'élément.

Client de chiffrement DynamoDB

Le contexte de chiffrement DynamoDB fournit des informations sur la table et l'élément au fournisseur de [matériel cryptographique](#) (CMP). [Dans les implémentations avancées, le contexte de chiffrement DynamoDB peut inclure une description matérielle demandée.](#)

Lorsque vous chiffrez des éléments de table, le contexte de chiffrement DynamoDB est lié cryptographiquement aux valeurs d'attributs chiffrées. Lorsque vous déchiffrez, si le contexte de chiffrement DynamoDB ne correspond pas exactement, en distinguant majuscules et minuscules, au contexte de chiffrement DynamoDB utilisé pour le chiffrement, l'opération de déchiffrement échoue.

Si vous interagissez directement avec le [chiffreur d'éléments](#), vous devez fournir un contexte de chiffrement DynamoDB lorsque vous appelez une méthode de chiffrement ou de déchiffrement. La plupart des assistants créent le contexte de chiffrement DynamoDB pour vous.

Note

Le contexte de chiffrement DynamoDB dans le client de chiffrement DynamoDB n'est pas lié au contexte de chiffrement dans () et le. AWS Key Management Service AWS KMS AWS Encryption SDK

Le contexte de chiffrement DynamoDB peut inclure les champs suivants. Tous les champs et valeurs sont facultatifs.

- Nom de la table
- Nom de la clé de partition
- Nom de la clé de tri
- Paires nom-valeur des attributs
- [Description du matériau demandé](#)

Magasin de fournisseur

Une boutique de fournisseurs est un composant qui renvoie des [fournisseurs de matériel cryptographique](#) (CMPs). Le magasin du fournisseur peut les créer CMPs ou les obtenir à partir d'une autre source, telle qu'un autre magasin du fournisseur. Le magasin du fournisseur enregistre les versions CMPs qu'il crée dans un stockage persistant dans lequel chaque CMP stocké est identifié par le nom matériel du demandeur et le numéro de version.

Le [fournisseur le plus récent](#) du client de chiffrement DynamoDB l'obtient CMPs auprès d'un magasin du fournisseur, mais vous pouvez utiliser le magasin du fournisseur pour CMPs approvisionner n'importe quel composant. Chaque fournisseur le plus récent est associé à une boutique de fournisseurs, mais une boutique de fournisseurs peut fournir des CMPs informations à de nombreux demandeurs sur plusieurs hôtes.

Le magasin du fournisseur crée de nouvelles versions CMPs de On Demand et renvoie les versions nouvelles et existantes. Il retourne aussi le dernier numéro de version d'un nom de matériau donné. Le demandeur peut ainsi savoir lorsque le magasin de fournisseur a une nouvelle version de son fournisseur, puis la demander.

Le client de chiffrement DynamoDB inclut [MetaStore](#) un magasin fournisseur qui crée des clés CMPs Wrapped qui sont stockées dans DynamoDB et chiffrées à l'aide d'un client de chiffrement DynamoDB interne.

En savoir plus :

- Magasin de fournisseur : [Java](#), [Python](#)
- MetaStore: [Java](#), [Python](#)

Fournisseur de matériel cryptographique

Note

Notre bibliothèque de chiffrement côté client a été [renommée AWS Database Encryption SDK](#). La rubrique suivante fournit des informations sur les versions 1. x —2. x du client de chiffrement DynamoDB pour Java et versions 1. x —3. x du client de chiffrement DynamoDB pour Python. Pour plus d'informations, consultez la section [SDK AWS de chiffrement de base de données pour la prise en charge des versions DynamoDB](#).

L'une des décisions les plus importantes que vous prenez lorsque vous utilisez le client de chiffrement DynamoDB est de sélectionner [un fournisseur de matériel cryptographique](#) (CMP). Le fournisseur CMP assemble et retourne les matériaux de déchiffrement au chiffreur d'élément. Il détermine aussi la façon dont les clés de chiffrement et les clés de signature sont générées, que les nouveaux matériaux de clés soient générés pour chaque élément ou soient réutilisés, et les algorithmes de chiffrement et de signature qui sont utilisés.

Vous pouvez choisir un CMP parmi les implémentations fournies dans les bibliothèques du client de chiffrement DynamoDB ou créer un CMP personnalisé compatible. Le choix du fournisseur CMP peut aussi dépendre du [langage de programmation](#) que vous utilisez.

Cette rubrique décrit les plus courantes CMPs et propose quelques conseils pour vous aider à choisir celle qui convient le mieux à votre application.

Fournisseur de matériaux KMS direct

Le fournisseur de matériel KMS direct protège les articles de votre table sous un format [AWS KMS key](#) qui ne laisse jamais [AWS Key Management Service](#) (AWS KMS) non chiffré. Votre application n'a pas à générer ou gérer des matériaux de chiffrement. Comme il utilise le AWS

KMS key pour générer des clés de chiffrement et de signature uniques pour chaque élément, ce fournisseur appelle AWS KMS chaque fois qu'il chiffre ou déchiffre un élément.

Si vous utilisez AWS KMS et qu'un AWS KMS appel par transaction est pratique pour votre application, ce fournisseur est un bon choix.

Pour en savoir plus, consultez [Fournisseur de matériaux KMS direct](#).

Fournisseur CMP encapsulé

Le Wrapped Materials Provider (Wrapped CMP) vous permet de générer et de gérer vos clés d'encapsulation et de signature en dehors du client de chiffrement DynamoDB.

Le fournisseur CMP encapsulé génère une clé de chiffrement unique pour chaque élément. Puis, il utilise les clés d'encapsulation (ou de désencapsulation) et de signature que vous fournissez. En tant que tel, vous déterminez de quelle façon les clés d'encapsulation et de signature sont générées, et si elles sont propres à chaque élément ou sont réutilisées. Le Wrapped CMP est une alternative sécurisée au [fournisseur Direct KMS](#) pour les applications qui n'utilisent pas de matériel cryptographique AWS KMS et qui peuvent le gérer en toute sécurité.

Pour en savoir plus, consultez [Fournisseur de matériaux encapsulé](#).

À propos du fournisseur le plus récent

Le fournisseur le plus récent est un [fournisseur CMP](#) conçu pour travailler avec un [magasin de fournisseur](#). Il CMPs provient du magasin du fournisseur et obtient le matériel cryptographique qu'il renvoie depuis le CMPs. Le fournisseur le plus récent utilise généralement chaque fournisseur CMP pour satisfaire plusieurs demandes de matériaux de chiffrement, mais vous pouvez utiliser les fonctions du magasin de fournisseur pour contrôler l'étendue à laquelle les matériaux sont réutilisés, déterminer à quelle fréquence a lieu la rotation des fournisseurs CMP et, même, modifier le type de fournisseur CMP utilisé sans modifier le fournisseur le plus récent.

Vous pouvez utiliser le fournisseur le plus récent avec n'importe quel magasin de fournisseur compatible. Le client de chiffrement DynamoDB inclut MetaStore un, qui est un magasin fournisseur qui renvoie Wrapped. CMPs

Le fournisseur le plus récent constitue un bon choix pour les applications qui doivent minimiser les appels à leur source de chiffrement, et pour les applications qui peuvent réutiliser certains matériaux de chiffrement sans enfreindre leurs exigences de sécurité. Par exemple, il vous permet de protéger votre matériel cryptographique avec un [AWS KMS key](#) in [AWS Key Management Service](#) (AWS KMS) sans avoir à appeler AWS KMS chaque fois que vous chiffrez ou déchiffrez un élément.

Pour en savoir plus, consultez [À propos du fournisseur le plus récent](#).

Fournisseur de matériaux statique

Le fournisseur de matériaux statiques est conçu pour les tests, les proof-of-concept démonstrations et la compatibilité existante. Il ne génère pas de matériaux de chiffrement uniques pour chaque élément. Il retourne les mêmes clés de chiffrement et de signature que vous fournissez, et ces clés sont utilisées directement pour chiffrer, déchiffrer et signer vos éléments de table.

Note

Le [fournisseur statique asymétrique](#) de la bibliothèque Java n'est pas un fournisseur statique. Il fournit juste d'autres constructeurs au [fournisseur CMP encapsulé](#). Il est sûr pour une utilisation en production, mais vous devez utiliser directement le CMP encapsulé chaque fois que possible.

Rubriques

- [Fournisseur de matériaux KMS direct](#)
- [Fournisseur de matériaux encapsulé](#)
- [À propos du fournisseur le plus récent](#)
- [Fournisseur de matériaux statique](#)

Fournisseur de matériaux KMS direct

Note

Notre bibliothèque de chiffrement côté client a été [renommée AWS Database Encryption SDK](#). La rubrique suivante fournit des informations sur les versions 1. x —2. x du client de chiffrement DynamoDB pour Java et versions 1. x —3. x du client de chiffrement DynamoDB pour Python. Pour plus d'informations, consultez la section [SDK AWS de chiffrement de base de données pour la prise en charge des versions DynamoDB](#).

Le fournisseur de matériel Direct KMS (fournisseur direct KMS) protège les éléments de votre table sous un format [AWS KMS key](#) qui ne laisse jamais [AWS Key Management Service](#) (AWS KMS)

non chiffré. Le [fournisseur CMP](#) retourne une clé de chiffrement et une clé de signature uniques pour chaque élément de table. Pour ce faire, il appelle AWS KMS chaque fois que vous chiffrez ou déchiffrez un élément.

Si vous traitez des éléments DynamoDB à une fréquence élevée et à grande échelle, vous risquez de dépasser les limites, ce qui entraîne des AWS KMS [requests-per-secondretards](#) de traitement. Si vous devez dépasser une limite, créez un dossier dans le [AWS Support Centre](#). Vous pouvez également envisager de faire appel à un fournisseur de matériel cryptographique avec une réutilisation limitée des clés, tel que le [fournisseur le plus récent](#).

Pour utiliser le fournisseur Direct KMS, l'appelant doit avoir, au moins [un Compte AWS](#) AWS KMS key, et l'autorisation d'appeler les opérations [GenerateDataKey](#) et de [déchiffrer](#) sur le. AWS KMS key AWS KMS key Il doit s'agir d'une clé de chiffrement symétrique ; le client de chiffrement DynamoDB ne prend pas en charge le chiffrement asymétrique. [Si vous utilisez une table globale DynamoDB, vous souhaitez peut-être spécifier AWS KMS une clé multirégionale.](#) Pour en savoir plus, consultez [Comment l'utiliser](#).

Note

Lorsque vous utilisez le fournisseur Direct KMS, les noms et les valeurs de vos attributs de clé primaire apparaissent en texte clair dans le [contexte de AWS KMS chiffrement](#) et dans AWS CloudTrail les journaux des AWS KMS opérations associées. Toutefois, le client de chiffrement DynamoDB n'expose jamais le texte en clair des valeurs d'attributs chiffrés.

Le fournisseur Direct KMS est l'un des nombreux [fournisseurs de matériel cryptographique](#) (CMPs) pris en charge par le client de chiffrement DynamoDB. Pour plus d'informations sur l'autre CMPs, voir [Fournisseur de matériel cryptographique](#).

Pour obtenir un exemple de code, consultez :

- Java: [AwsKmsEncryptedItem](#)
- Python : [aws-kms-encrypted-table](#), [aws-kms-encrypted-item](#)

Rubriques

- [Comment l'utiliser](#)
- [Comment ça marche](#)

Comment l'utiliser

Pour créer un fournisseur KMS direct, utilisez le paramètre Key ID pour spécifier une [clé KMS](#) de chiffrement symétrique dans votre compte. La valeur du paramètre Key ID peut être l'ID de clé, l'ARN de clé, le nom d'alias ou l'ARN d'alias du AWS KMS key. Pour plus de détails sur les identifiants de clé, consultez la section [Identifiants de clé](#) dans le guide du AWS Key Management Service développeur.

Le fournisseur Direct KMS nécessite une clé KMS de chiffrement symétrique. Vous ne pouvez pas utiliser de clé KMS asymétrique. Toutefois, vous pouvez utiliser une clé KMS multirégionale, une clé KMS contenant des éléments clés importés ou une clé KMS dans un magasin de clés personnalisé. Vous devez disposer des autorisations [kms : GenerateDataKey](#) et [KMS:Decrypt](#) sur la clé KMS. Vous devez donc utiliser une clé gérée par le client, et non une clé KMS AWS gérée ou AWS détenue.

Le client de chiffrement DynamoDB pour Python détermine la région à AWS KMS appeler depuis la région dans la valeur du paramètre ID clé, s'il en inclut une. Sinon, il utilise la région du AWS KMS client, si vous en spécifiez une, ou la région que vous configurez dans le AWS SDK pour Python (Boto3). Pour plus d'informations sur la sélection de régions en Python, consultez [Configuration](#) dans le manuel de AWS référence de l'API SDK for Python (Boto3).

Le client de chiffrement DynamoDB pour Java détermine la région à AWS KMS appeler depuis la région du client, si AWS KMS le client que vous spécifiez inclut une région. Dans le cas contraire, il utilise la région que vous configurez dans le AWS SDK pour Java. Pour plus d'informations sur la sélection des régions dans le AWS SDK pour Java, voir [Région AWS sélection](#) dans le guide du AWS SDK pour Java développeur.

Java

```
// Replace the example key ARN and Region with valid values for your application
final String keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
final String region = 'us-west-2'

final AWSKMS kms = AWSKMSClientBuilder.standard().withRegion(region).build();
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, keyArn);
```

Python

L'exemple suivant utilise la clé ARN pour spécifier le AWS KMS key. Si votre identifiant de clé n'inclut pas de Région AWS, le client de chiffrement DynamoDB obtient la région à partir de la session Botocore configurée, s'il en existe une, ou à partir des paramètres par défaut de Boto.

```
# Replace the example key ID with a valid value
kms_key = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key)
```

Si vous utilisez des tables [globales Amazon DynamoDB](#), nous vous recommandons de chiffrer vos données sous une clé multirégionale. AWS KMS Les clés multirégionales sont AWS KMS keys différentes Régions AWS et peuvent être utilisées de manière interchangeable car elles ont le même identifiant de clé et le même matériau de clé. Pour plus de détails, consultez la section [Utilisation de clés multirégionales](#) dans le Guide du AWS Key Management Service développeur.

Note

Si vous utilisez la [version 2017.11.29](#) des tables globales, vous devez définir des actions attributaires afin que les champs de réplication réservés ne soient ni chiffrés ni signés. Pour en savoir plus, consultez [Problèmes liés aux anciennes versions des tables globales](#).

Pour utiliser une clé multirégionale avec le client de chiffrement DynamoDB, créez une clé multirégionale et répliquez-la dans les régions dans lesquelles votre application s'exécute. Configurez ensuite le fournisseur Direct KMS pour utiliser la clé multirégionale dans la région dans laquelle le client de chiffrement DynamoDB appelle. AWS KMS

L'exemple suivant configure le client de chiffrement DynamoDB pour chiffrer les données dans la région USA Est (Virginie du Nord) (us-east-1) et les déchiffrer dans la région USA Ouest (Oregon) (us-west-2) à l'aide d'une clé multirégionale.

Java

Dans cet exemple, le client de chiffrement DynamoDB obtient la région à AWS KMS appeler depuis la région du client. AWS KMS La `keyArn` valeur identifie une clé multirégionale dans la même région.

```
// 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

Dans cet exemple, le client de chiffrement DynamoDB obtient la région à AWS KMS appeler depuis la région dans l'ARN de la clé.

```
# 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)
```

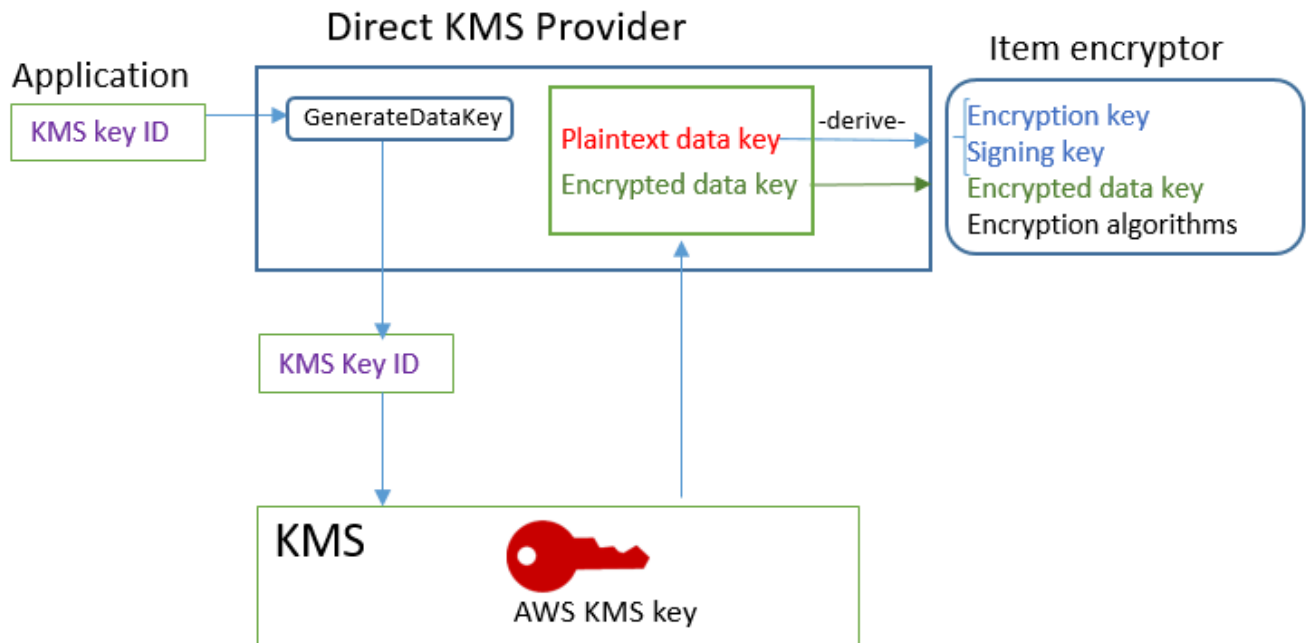
```
# Decrypt in us-west-2

# 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)
```

Comment ça marche

Le fournisseur Direct KMS renvoie des clés de chiffrement et de signature protégées par un AWS KMS key code que vous spécifiez, comme indiqué dans le schéma suivant.

Direct KMS Provider



- Pour générer du matériel de chiffrement, le fournisseur Direct KMS demande de AWS KMS [générer une clé de données unique](#) pour chaque élément à l'aide d'une clé AWS KMS key que vous spécifiez. Il dérive les clés de chiffrement et de signature de l'élément depuis la copie en texte brut de la [clé de données](#), puis retourne les clés de chiffrement et de signature, ainsi que la clé des données chiffrées, qui est stockée dans l'[attribut de description de matériau](#) de l'élément.

Le chiffreur d'élément utilise les clés de chiffrement et de signature, et les supprime de la mémoire dès que possible. Seule la copie chiffrée de la clé de données à partir de laquelle elles ont été dérivées est enregistrée dans l'élément chiffré.

- Pour générer du matériel de déchiffrement, le fournisseur KMS direct demande de AWS KMS déchiffrer la clé de données chiffrée. Puis, il dérive les clés de vérification et de signature de la clé de données en texte brut, et les retourne au chiffreur d'élément.

Le chiffreur d'élément vérifie l'élément et, si la vérification aboutit, déchiffre les valeurs chiffrées. Puis, il supprime les clés de la mémoire dès que possible.

Obtention des matériaux de chiffrement

Cette section décrit en détail les entrées, les sorties et le traitement du fournisseur KMS direct quand il reçoit une demande de matériaux de chiffrement de la part du [chiffreur d'élément](#).

Entrée (depuis l'application)

- L'identifiant clé d'un AWS KMS key.

Entrée (depuis le chiffreur d'élément)

- [Contexte de chiffrement DynamoDB](#)

Sortie (vers le chiffreur d'élément)

- Clé de chiffrement (texte brut)
- Clé de signature
- Dans [Description du matériau réel](#) : ces valeurs sont enregistrées dans l'attribut de description du matériau que le client ajoute à l'élément.
 - amzn-ddb-env-key: clé de données codée en Base64 cryptée par le AWS KMS key
 - amzn-ddb-env-alg: algorithme de chiffrement, par défaut [AES/256](#)
 - amzn-ddb-sig-alg: algorithme de signature, par défaut, [Hmac /256 SHA256](#)
 - amzn-ddb-wrap-alg: km

Traitement

1. Le fournisseur Direct KMS envoie AWS KMS une demande d'utilisation de la clé spécifiée AWS KMS key pour [générer une clé de données unique](#) pour l'article. L'opération retourne une clé en texte brut et une copie chiffrée sous la AWS KMS key. Ce matériau est appelé matériau de clé initial.

La demande inclut les valeurs suivantes en texte brut dans le [contexte de chiffrement AWS KMS](#). Ces valeurs non secrètes sont liées en termes de chiffrement à l'objet chiffré, si bien que le même contexte de chiffrement est requis au déchiffrement. Vous pouvez utiliser ces valeurs pour identifier l'appel AWS KMS dans les [AWS CloudTrail journaux](#).

- amzn-ddb-env-alg — Algorithme de chiffrement, par défaut AES/256

- `amzn-ddb-sig-alg` — Algorithme de signature, par défaut Hmac /256 SHA256
- (Facultatif) `aws-kms-table` — *table name*
- (Facultatif) *partition key name* — *partition key value* (les valeurs binaires sont codées en Base64)
- (Facultatif) *sort key name* — *sort key value* (les valeurs binaires sont codées en Base64)

Le fournisseur Direct KMS obtient les valeurs du contexte de AWS KMS chiffrement à partir du contexte de chiffrement [DynamoDB de l'élément](#). Si le contexte de chiffrement DynamoDB n'inclut aucune valeur, telle que le nom de la table, cette paire nom-valeur est omise du contexte de chiffrement. AWS KMS

2. Le fournisseur KMS direct dérive une clé de chiffrement symétrique et une clé de signature à partir de la clé de données. Par défaut, il utilise l'[algorithme de hachage sécurisé \(SHA\) 256](#) et la [fonction de dérivation de clé RFC5869 basée sur HMAC pour dériver une clé](#) de chiffrement symétrique AES 256 bits et une clé de signature HMAC-SHA-256 bits.
3. Le fournisseur KMS direct retourne la sortie du chiffreur d'élément.
4. Le chiffreur d'élément utilise la clé de chiffrement pour chiffrer les attributs spécifiés et la clé de signature pour les signer, à l'aide des algorithmes spécifiés dans la description du matériau réel. Il supprime les clés en texte brut de la mémoire dès que possible.

Obtention des matériaux de déchiffrement

Cette section décrit en détail les entrées, les sorties et le traitement du fournisseur KMS direct quand il reçoit une demande de déchiffrement de matériaux de la part du [chiffreur d'élément](#).

Entrée (depuis l'application)

- L'identifiant clé d'un AWS KMS key.

La valeur de l'ID de clé peut être l'ID de clé, l'ARN de clé, le nom d'alias ou l'ARN d'alias du AWS KMS key. Toutes les valeurs qui ne sont pas incluses dans l'ID de clé, telles que la région, doivent être disponibles dans le [profil AWS nommé](#). La clé ARN fournit toutes les valeurs AWS KMS nécessaires.

Entrée (depuis le chiffreur d'élément)

- Copie du contexte de [chiffrement DynamoDB](#) qui contient le contenu de l'attribut de description du matériau.

Sortie (vers le chiffreur d'élément)

- Clé de chiffrement (texte brut)
- Clé de signature

Traitement

1. Le fournisseur Direct KMS obtient la clé de données chiffrée à partir de l'attribut de description du matériau contenu dans l'élément chiffré.
2. Il demande AWS KMS d'utiliser le paramètre spécifié AWS KMS key pour [déchiffrer](#) la clé de données cryptée. L'opération retourne une clé en texte brut.

Cette demande doit utiliser le même [contexte de chiffrement AWS KMS](#) que celui utilisé pour générer et chiffrer la clé de données.

- `aws-kms-table` – *table name*
 - *partition key name* — *partition key value* (les valeurs binaires sont codées en Base64)
 - (Facultatif) *sort key name* — *sort key value* (les valeurs binaires sont codées en Base64)
 - `amzn-ddb-env-alg` — Algorithme de chiffrement, par défaut AES/256
 - `amzn-ddb-sig-alg` — Algorithme de signature, par défaut Hmac /256 SHA256
3. Le fournisseur Direct KMS utilise l'[algorithme de hachage sécurisé \(SHA\) 256](#) et une [fonction de dérivation de clé RFC5869 basée sur HMAC pour dériver une clé](#) de chiffrement symétrique AES 256 bits et une clé de signature HMAC-SHA-256 bits à partir de la clé de données.
 4. Le fournisseur KMS direct retourne la sortie du chiffreur d'élément.
 5. Le chiffreur d'élément utilise la clé de signature pour vérifier l'élément. S'il réussit, il utilise la clé de chiffrement symétrique pour déchiffrer les valeurs d'attribut chiffrées. Ces opérations utilisent les algorithmes de chiffrement et de signature spécifiés dans la description du matériau réel. Le chiffreur d'élément supprime les clés en texte brut de la mémoire dès que possible.

Fournisseur de matériaux encapsulé

Note

Notre bibliothèque de chiffrement côté client a été [renommée AWS Database Encryption SDK](#). La rubrique suivante fournit des informations sur les versions 1. x —2. x du client de chiffrement DynamoDB pour Java et versions 1. x —3. x du client de chiffrement DynamoDB pour Python. Pour plus d'informations, consultez la section [SDK AWS de chiffrement de base de données pour la prise en charge des versions DynamoDB](#).

Le Wrapped Materials Provider (Wrapped CMP) vous permet d'utiliser des clés d'encapsulation et de signature provenant de n'importe quelle source avec le client de chiffrement DynamoDB. Le CMP Wrapped ne dépend d'aucun AWS service. Cependant, vous devez générer et gérer vos clés d'encapsulation et de signature en dehors du client, y compris la fourniture des clés appropriées pour vérifier et déchiffrer l'élément.

Le fournisseur CMP encapsulé génère une clé de chiffrement d'élément unique pour chaque élément. Il encapsule la clé de chiffrement d'élément avec la clé d'encapsulation que vous fournissez et enregistre la clé de chiffrement d'élément encapsulée dans l'[attribut de description de matériau](#) de l'élément. Comme vous fournissez les clés d'encapsulation et de signature, vous déterminez de quelle façon les clés d'encapsulation et de signature sont générées, et si elles sont propres à chaque élément ou sont réutilisées.

Le fournisseur CMP encapsulé constitue une implémentation sécurisée et un bon choix pour les applications qui peuvent gérer les matériaux de chiffrement.

Le Wrapped CMP est l'un des nombreux [fournisseurs de matériel cryptographique](#) (CMPs) pris en charge par le client de chiffrement DynamoDB. Pour plus d'informations sur l'autre CMPs, voir [Fournisseur de matériel cryptographique](#).

Pour obtenir un exemple de code, consultez :

- Java: [AsymmetricEncryptedItem](#)
- Python : [wrapped-rsa-encrypted-table](#), [wrapped-symmetric-encrypted-table](#)

Rubriques

- [Comment l'utiliser](#)

- [Comment ça marche](#)

Comment l'utiliser

Pour créer un fournisseur CMP encapsulé, spécifiez une clé d'encapsulation (requis au chiffrement), une clé de désencapsulation (requis au déchiffrement) et une clé de signature. Vous devez fournir les clés lorsque vous chiffrez et déchiffrez les éléments.

Les clés d'encapsulation, de désencapsulation et de signature peuvent être des clés symétriques ou des paires de clés asymétriques.

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);
```

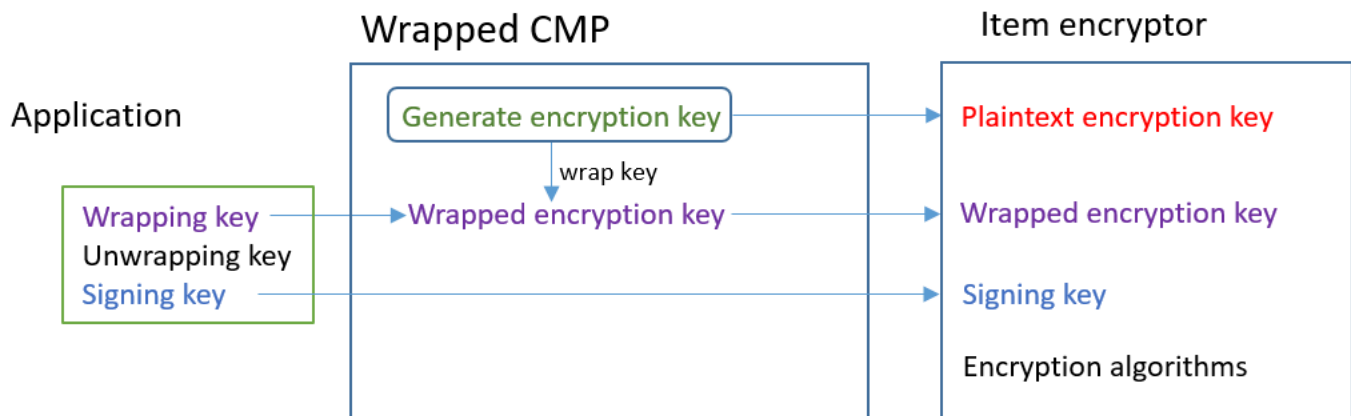
Python

```
# 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
)
```

Comment ça marche

Le fournisseur CMP encapsulé génère une nouvelle clé de chiffrement d'élément pour chaque élément. Il utilise les clés d'encapsulation, de désencapsulation et de signature que vous fournissez, comme illustré dans le schéma suivant.



Obtention des matériaux de chiffrement

Cette section décrit en détail les entrées, les sorties et le traitement du fournisseur CMP encapsulé quand il reçoit une demande de matériaux de chiffrement.

Entrée (depuis l'application)

- Clé d'encapsulation : clé symétrique [Advanced Encryption Standard](#) (AES) ou clé publique [RSA](#). Obligatoire si les valeurs d'attribut sont chiffrées. Sinon, elle est facultative et ignorée.
- Clé de désencapsulation : facultative et ignorée.
- Clé de signature

Entrée (depuis le chiffreur d'élément)

- [Contexte de chiffrement DynamoDB](#)

Sortie (vers le chiffreur d'élément) :

- Clé de chiffrement d'élément en texte brut
- Clé de signature (inchangée)
- [Description du matériau réel](#) : ces valeurs sont enregistrées dans l'[attribut de description du matériau](#) que le client ajoute à l'élément.
 - `amzn-ddb-env-key` : clé de chiffrement d'élément codée en base64
 - `amzn-ddb-env-alg` : algorithme de chiffrement utilisé pour chiffrer l'élément. La valeur par défaut est AES-256-CBC.

- `amzn-ddb-wrap-alg` : algorithme d'encapsulation utilisé par le fournisseur CMP encapsulé pour encapsuler la clé de chiffrement d'élément. Si la clé d'encapsulation est une clé AES, la clé est encapsulée à l'aide de `AES-Keywrap` non complétée, comme défini dans [RFC 3394](#). Si la clé d'encapsulation est une clé RSA, elle est chiffrée à l'aide de RSA OAEP avec rembourrage. MGF1

Traitement

Lorsque vous chiffrez un élément, vous transmettez une clé d'encapsulation et une clé de signature. Une clé de désencapsulation est facultative et ignorée.

1. Le fournisseur CMP encapsulé génère une clé de chiffrement d'élément symétrique unique pour l'élément de table.
2. Il utilise la clé d'encapsulation que vous spécifiez pour encapsuler la clé de chiffrement d'élément. Puis, il la supprime de la mémoire dès que possible.
3. Il retourne la clé de chiffrement d'élément en texte brut, la clé de signature que vous avez fournie et une [description de matériel réel](#) qui inclut la clé de chiffrement d'élément encapsulé, ainsi que les algorithmes de chiffrement et d'encapsulation.
4. Le chiffreur d'élément utilise la clé de chiffrement en texte brut pour chiffrer l'élément. Il utilise la clé de signature que vous avez fournie pour signer l'élément. Puis, il supprime les clés en texte brut de la mémoire dès que possible. Il copie les champs de la description du matériel réel, y compris la clé de chiffrement encapsulée (`amzn-ddb-env-key`), dans l'attribut de la description de matériel de l'élément.

Obtention des matériaux de déchiffrement

Cette section décrit en détail les entrées, les sorties et le traitement du fournisseur CMP encapsulé quand il reçoit une demande de matériaux de déchiffrement.

Entrée (depuis l'application)

- Clé d'encapsulation : facultative et ignorée.
- Clé de désencapsulation : la même clé symétrique [Advanced Encryption Standard](#) (AES) ou la même clé privée [RSA](#) qui correspond à la clé publique RSA utilisée pour chiffrer. Obligatoire si les valeurs d'attribut sont chiffrées. Sinon, elle est facultative et ignorée.
- Clé de signature

Entrée (depuis le chiffreur d'élément)

- Copie du contexte de [chiffrement DynamoDB](#) qui contient le contenu de l'attribut de description du matériau.

Sortie (vers le chiffreur d'élément)

- Clé de chiffrement d'élément en texte brut
- Clé de signature (inchangée)

Traitement

Lorsque vous déchiffrez un élément, vous transmettez une clé de désencapsulation et une clé de signature. Une clé d'encapsulation est facultative et ignorée.

1. Le fournisseur CMP encapsulé obtient la clé de chiffrement d'élément encapsulé depuis l'attribut de description du matériau de l'élément.
2. Il utilise la clé et l'algorithme de désencapsulation pour désencapsuler la clé de chiffrement d'élément.
3. Il retourne la clé de chiffrement de l'élément en texte brut, la clé de signature, et les algorithmes de chiffrement et de signature au chiffreur d'élément.
4. Le chiffreur d'élément utilise la clé de signature pour vérifier l'élément. S'il réussit, il utilise la clé de chiffrement d'élément pour déchiffrer l'élément. Puis, il supprime les clés en texte brut de la mémoire dès que possible.

À propos du fournisseur le plus récent

Note

Notre bibliothèque de chiffrement côté client a été [renommée AWS Database Encryption SDK](#). La rubrique suivante fournit des informations sur les versions 1. x —2. x du client de chiffrement DynamoDB pour Java et versions 1. x —3. x du client de chiffrement DynamoDB pour Python. Pour plus d'informations, consultez la section [SDK AWS de chiffrement de base de données pour la prise en charge des versions DynamoDB](#).

Le fournisseur le plus récent est un [fournisseur CMP](#) conçu pour travailler avec un [magasin de fournisseur](#). Il CMPs provient du magasin du fournisseur et obtient le matériel cryptographique qu'il renvoie depuis le CMPs. Il utilise généralement chaque CMP pour répondre à plusieurs demandes de matériaux de chiffrement. Cependant, vous pouvez utiliser les fonctions de son magasin de fournisseur pour contrôler jusqu'à quelle mesure les matériaux sont réutilisés, déterminer la fréquence de rotation de son fournisseur CMP et, même, modifier le type de fournisseur CMP utilisé sans modifier le fournisseur le plus récent.

Note

Le code associé au `MostRecentProvider` symbole du fournisseur le plus récent peut stocker du matériel cryptographique en mémoire pendant toute la durée de vie du processus. Cela peut permettre à un appelant d'utiliser des clés qu'il n'est plus autorisé à utiliser. Le `MostRecentProvider` symbole est obsolète dans les anciennes versions prises en charge du client de chiffrement DynamoDB et supprimé de la version 2.0.0. Il est remplacé par le `CachingMostRecentProvider` symbole. Pour en savoir plus, consultez [Mises à jour du fournisseur le plus récent](#).

Le fournisseur le plus récent constitue un bon choix pour les applications qui doivent minimiser les appels au magasin de fournisseur et à sa source de chiffrement, et pour les applications qui peuvent réutiliser certains matériaux de chiffrement sans enfreindre leurs exigences de sécurité. Par exemple, il vous permet de protéger votre matériel cryptographique sous un [AWS KMS key](#) in [AWS Key Management Service](#) (AWS KMS) sans appeler AWS KMS chaque fois que vous cryptez ou décryptez un élément.

Le magasin de fournisseurs que vous choisissez détermine le type de magasin utilisé par le fournisseur le plus récent et la fréquence à CMPs laquelle il reçoit un nouveau CMP. Vous pouvez utiliser tout magasin de fournisseur compatible avec le fournisseur le plus récent, y compris les magasins de fournisseur personnalisés que vous concevez.

Le client de chiffrement DynamoDB inclut `MetaStore` un qui crée et [renvoie des fournisseurs de matériaux encapsulés \(encapsulés\)](#). CMPs Il `MetaStore` enregistre plusieurs versions du `Wrapped` CMPs qu'il génère dans une table DynamoDB interne et les protège par un chiffrement côté client par une instance interne du client de chiffrement DynamoDB.

Vous pouvez configurer le `MetaStore` pour utiliser n'importe quel type de CMP interne pour protéger le contenu du tableau, y compris un [fournisseur KMS direct](#) qui génère du matériel cryptographique

protégé par vos soins AWS KMS key, un CMP encapsulé qui utilise les clés d'encapsulation et de signature que vous fournissez, ou un CMP personnalisé compatible que vous concevez.

Pour obtenir un exemple de code, consultez :

- Java: [MostRecentEncryptedItem](#)
- Python : [most_recent_provider_encrypted_table](#)

Rubriques

- [Comment l'utiliser](#)
- [Comment ça marche](#)
- [Mises à jour du fournisseur le plus récent](#)

Comment l'utiliser

Pour créer un fournisseur le plus récent, vous devez créer et configurer un magasin de fournisseur, puis créer un fournisseur le plus récent qui utilise le magasin de fournisseur.

[Les exemples suivants montrent comment créer un fournisseur le plus récent qui utilise un MetaStore et protège les versions de sa table DynamoDB interne avec du matériel cryptographique provenant d'un fournisseur KMS direct.](#) Ces exemples utilisent le [CachingMostRecentProviders](#) symbole.

Chaque fournisseur le plus récent possède un nom qui l'identifie CMPs dans le MetaStore tableau, un paramètre [time-to-live](#)(TTL) et un paramètre de taille de cache qui détermine le nombre d'entrées que le cache peut contenir. Ces exemples définissent la taille du cache à 1 000 entrées et à un TTL de 60 secondes.

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
)
```

Comment ça marche

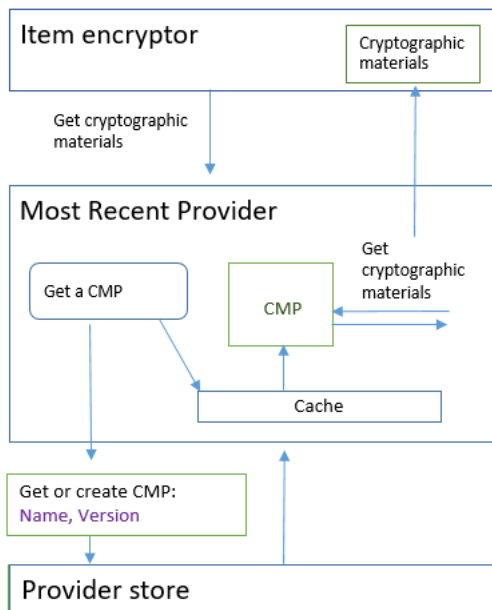
Le fournisseur le plus récent CMPs provient d'un magasin de fournisseurs. Puis, il utilise le fournisseur CMP pour générer les matériaux de chiffrement qu'il retourne au chiffreur d'élément.

À propos du fournisseur le plus récent

Le fournisseur le plus récent obtient un [fournisseur CMP](#) à partir d'un [magasin de fournisseur](#). Puis, il utilise le fournisseur CMP pour générer les matériaux de chiffrement qu'il retourne. Chaque fournisseur le plus récent est associé à une boutique de fournisseurs, mais une boutique de fournisseurs peut CMPs approvisionner plusieurs fournisseurs sur plusieurs hôtes.

Le fournisseur le plus récent peut travailler avec n'importe quel fournisseur CMP compatible d'un magasin de fournisseur. Il demande du matériel de chiffrement ou de déchiffrement au CMP et renvoie le résultat au chiffreur d'articles. Il n'effectue pas d'opération de chiffrement.

Pour demander un fournisseur CMP auprès de son magasin de fournisseur, le fournisseur le plus récent fournit son nom de matériau et la version d'un fournisseur CMP existant qu'il veut utiliser. Pour les matériaux de chiffrement, le fournisseur le plus récent demande toujours la version maximale (la « plus récente »). Pour les matériaux de déchiffrement, il demande la version du fournisseur CMP qui a été utilisée pour créer les matériaux de chiffrement, comme illustré dans le diagramme suivant.



Le fournisseur le plus récent enregistre les versions CMPs renvoyées par le magasin du fournisseur dans un cache local le moins récemment utilisé (LRU) en mémoire. Le cache permet au fournisseur le plus récent d'obtenir CMPs ce dont il a besoin sans avoir à appeler le magasin du fournisseur pour chaque élément. Vous pouvez effacer le cache à la demande.

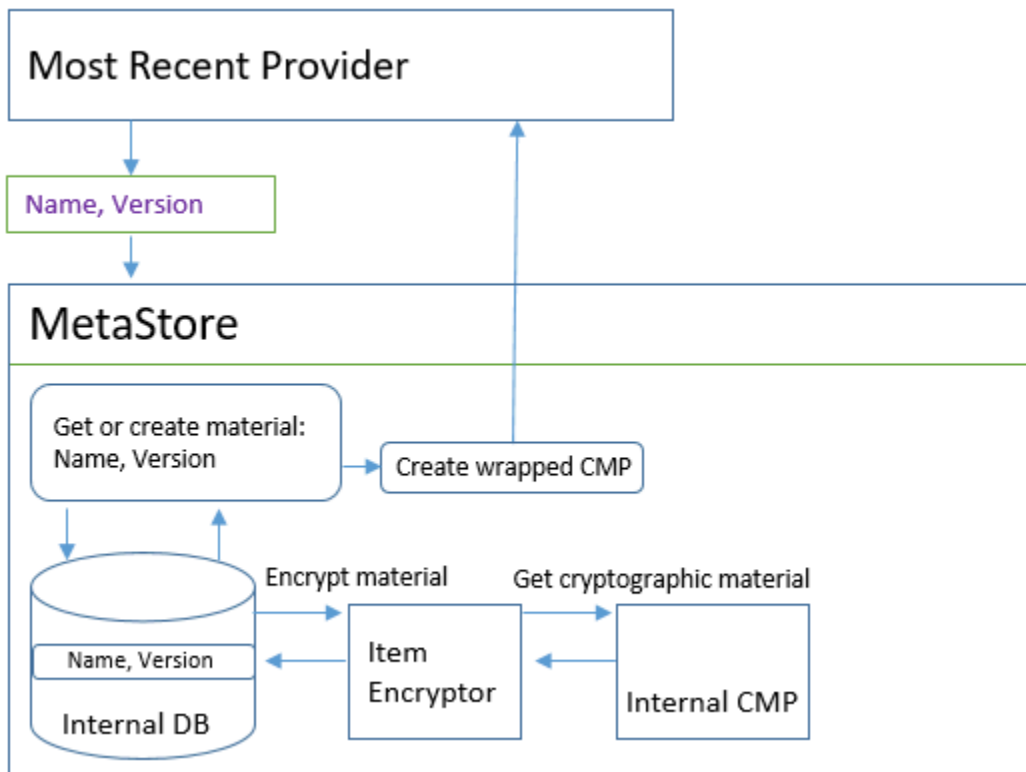
Le fournisseur le plus récent utilise une [time-to-live valeur](#) configurable que vous pouvez ajuster en fonction des caractéristiques de votre application.

À propos de MetaStore

Vous pouvez utiliser un fournisseur le plus récent avec n'importe quel magasin de fournisseur, y compris un magasin de fournisseur personnalisé compatible. Le client de chiffrement DynamoDB inclut MetaStore une implémentation sécurisée que vous pouvez configurer et personnaliser.

A MetaStore est un [magasin fournisseur](#) qui crée et renvoie des [Wrapped CMPs](#) configurés avec la clé d'encapsulation, la clé de déballage et la clé de signature requises par Wrapped CMPs . A MetaStore est une option sécurisée pour le fournisseur le plus récent, car Wrapped génère CMPs toujours des clés de chiffrement uniques pour chaque article. Seule la clé d'encapsulation qui protège la clé de chiffrement d'élément et la clé de signature sont réutilisées.

Le schéma suivant montre les composants du fournisseur le plus récent MetaStore et la manière dont il interagit avec celui-ci.



Il MetaStore génère le Wrapped CMPs, puis le stocke (sous forme cryptée) dans une table DynamoDB interne. La clé de partition est le nom du matériel du fournisseur le plus récent ; la clé de tri son numéro de version. Les éléments de la table sont protégés par un client de chiffrement DynamoDB interne, comprenant un crypteur d'éléments et un fournisseur [interne de matériel cryptographique](#) (CMP).

Vous pouvez utiliser n'importe quel type de CMP interne MetaStore, y compris un [fournisseur KMS direct](#), un CMP encapsulé avec du matériel cryptographique que vous fournissez ou un CMP personnalisé compatible. Si le CMP interne de votre ordinateur MetaStore est un fournisseur KMS direct, vos clés d'emballage et de signature réutilisables sont protégées par un [AWS KMS key](#) in [AWS Key Management Service](#) (AWS KMS). Les MetaStore appels AWS KMS chaque fois qu'il ajoute une nouvelle version CMP à sa table interne ou qu'il obtient une version CMP à partir de sa table interne.

Définition d'une time-to-live valeur

Vous pouvez définir une valeur time-to-live (TTL) pour chaque fournisseur le plus récent que vous créez. En général, utilisez la valeur TTL la plus basse adaptée à votre application.

L'utilisation de la valeur TTL est modifiée dans le `CachingMostRecentProvider` symbole du fournisseur le plus récent.

Note

Le `MostRecentProvider` symbole du fournisseur le plus récent est obsolète dans les anciennes versions prises en charge du client de chiffrement DynamoDB et supprimé de la version 2.0.0. Il est remplacé par le `CachingMostRecentProvider` symbole. Nous vous recommandons de mettre à jour votre code dès que possible. Pour en savoir plus, consultez [Mises à jour du fournisseur le plus récent](#).

CachingMostRecentProvider

`CachingMostRecentProvider` utilise la valeur TTL de deux manières différentes.

- Le TTL détermine la fréquence à laquelle le fournisseur le plus récent vérifie la présence d'une nouvelle version du CMP dans le magasin du fournisseur. Si une nouvelle version est disponible, le fournisseur le plus récent remplace son CMP et actualise son matériel cryptographique. Dans le cas contraire, il continue d'utiliser son CMP et son matériel cryptographique actuels.
- Le TTL détermine la durée pendant laquelle le cache peut être utilisé. Avant d'utiliser un CMP mis en cache pour le chiffrement, le fournisseur le plus récent évalue le temps passé dans le cache. Si le temps de cache du CMP dépasse le TTL, le CMP est expulsé du cache et le fournisseur le plus récent obtient une nouvelle version de CMP dans le magasin de son fournisseur.

MostRecentProvider

Dans le `MostRecentProvider`, le TTL détermine la fréquence à laquelle le fournisseur le plus récent vérifie la présence d'une nouvelle version du CMP dans le magasin du fournisseur. Si une nouvelle version est disponible, le fournisseur le plus récent remplace son CMP et actualise son matériel cryptographique. Dans le cas contraire, il continue d'utiliser son CMP et son matériel cryptographique actuels.

Le TTL ne détermine pas la fréquence à laquelle une nouvelle version du CMP est créée. Vous créez de nouvelles versions CMP en faisant [pivoter les matériaux cryptographiques](#).

La valeur TTL idéale varie en fonction de l'application et de ses objectifs de latence et de disponibilité. Un TTL inférieur améliore votre profil de sécurité en réduisant le temps pendant lequel les documents cryptographiques sont stockés en mémoire. En outre, un TTL inférieur actualise les

informations critiques plus fréquemment. Par exemple, si votre CMP interne est un [fournisseur KMS direct](#), il vérifie plus fréquemment que l'appelant est toujours autorisé à utiliser un. AWS KMS key

Toutefois, si le TTL est trop bref, les appels fréquents vers le magasin du fournisseur peuvent augmenter vos coûts et obliger ce dernier à limiter les demandes provenant de votre application et d'autres applications partageant votre compte de service. Il peut également être avantageux de coordonner le TTL avec la vitesse à laquelle vous faites pivoter les documents cryptographiques.

Pendant les tests, modifiez le TTL et la taille du cache en fonction des différentes charges de travail jusqu'à ce que vous trouviez une configuration adaptée à votre application et à vos normes de sécurité et de performance.

Rotation des matériaux de chiffrement

Lorsqu'un fournisseur le plus récent a besoin de matériel de chiffrement, il utilise toujours la version la plus récente de son CMP dont il a connaissance. La fréquence à laquelle il vérifie la présence d'une version plus récente est déterminée par la valeur [time-to-live](#)(TTL) que vous définissez lorsque vous configurez le fournisseur le plus récent.

Lorsque le TTL expire, le fournisseur le plus récent vérifie la version la plus récente du CMP dans le magasin du fournisseur. Si un CMP est disponible, le fournisseur le plus récent l'obtient et remplace le CMP dans son cache. Il utilise ce CMP et son matériel cryptographique jusqu'à ce qu'il découvre que le magasin du fournisseur dispose d'une version plus récente.

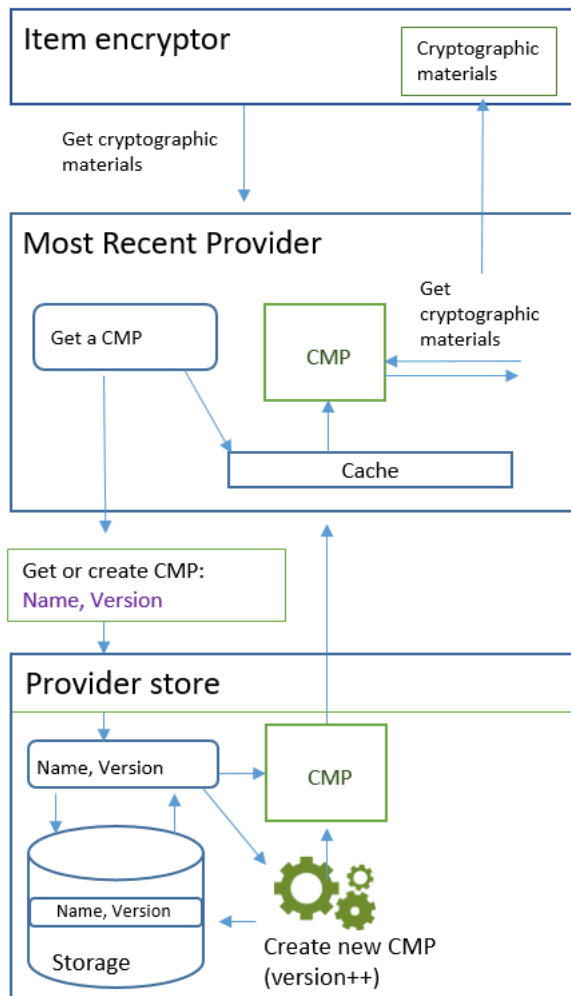
Pour demander au magasin de fournisseur de créer une version d'un CMP pour un fournisseur le plus récent, appelez l'opération Create New Provider du magasin de fournisseur avec le nom de matériau du fournisseur le plus récent. Le magasin de fournisseur crée un CMP et enregistre une copie chiffrée dans son stockage interne avec un numéro de version supérieur. (Il retourne aussi un CMP, mais vous pouvez l'ignorer.) Par conséquent, la prochaine fois que le fournisseur le plus récent demandera au magasin du fournisseur le numéro de version maximal de celui-ci CMPs, il obtiendra le nouveau numéro de version supérieur et l'utilisera dans les demandes ultérieures adressées au magasin pour voir si une nouvelle version du CMP a été créée.

Vous pouvez planifier vos appels Create New Provider en fonction de l'heure, du nombre d'éléments ou d'attributs traités, ou de toute autre métrique qui revêt un sens pour votre application.

Obtention des matériaux de chiffrement

Le fournisseur le plus récent utilise le processus suivant, illustré dans le diagramme, pour obtenir les matériaux de chiffrement qu'il retourne au chiffreur d'élément. La sortie dépend du type de

fournisseur CMP que le magasin de fournisseur retourne. Le fournisseur le plus récent peut utiliser n'importe quel magasin de fournisseurs compatible, y compris MetaStore celui inclus dans le client de chiffrement DynamoDB.



Lorsque vous créez un fournisseur le plus récent à l'aide du [CachingMostRecentProvidersymbole](#), vous spécifiez un magasin de fournisseurs, le nom du fournisseur le plus récent et une valeur [time-to-live](#)(TTL). Vous pouvez également éventuellement spécifier une taille de cache, qui détermine le nombre maximal de documents cryptographiques pouvant exister dans le cache.

Quand le chiffreur d'élément demande au fournisseur le plus récent les matériaux de chiffrement, le fournisseur le plus récent commence par chercher dans le cache le dernier numéro de version de ses fournisseurs CMP.

- S'il trouve la dernière version du CMP dans son cache et que le CMP n'a pas dépassé la valeur TTL, le fournisseur le plus récent utilise le CMP pour générer du matériel de chiffrement. Puis,

il retourne les matériaux de chiffrement au chiffreur d'élément. Cette opération ne requiert pas d'appel au magasin de fournisseur.

- Si la dernière version du CMP n'est pas dans son cache, ou si elle est dans le cache mais a dépassé sa valeur TTL, le fournisseur le plus récent demande un CMP à son magasin de fournisseurs. La demande inclut le nom de matériau du fournisseur le plus récent et le numéro de version maximal qu'il connaît.
 1. Le magasin de fournisseur retourne un fournisseur CMP à partir de son stockage permanent. Si le magasin du fournisseur est un MetaStore, il obtient un CMP encapsulé chiffré à partir de sa table DynamoDB interne en utilisant le nom du matériau du fournisseur le plus récent comme clé de partition et le numéro de version comme clé de tri. Il MetaStore utilise son crypteur d'éléments interne et son CMP interne pour déchiffrer le CMP encapsulé. Ensuite, il retourne le fournisseur CMP en texte brut au fournisseur le plus récent. Si le CMP interne est un [fournisseur KMS direct](#), cette étape inclut un appel de [AWS Key Management Service](#) (AWS KMS).
 2. Le fournisseur CMP ajoute le champ `amzn-ddb-meta-id` à la [description du matériau réel](#). Sa valeur est le nom de matériau et la version du CMP dans sa table interne. Le magasin de fournisseur retourne le fournisseur CMP au fournisseur le plus récent.
 3. Le fournisseur le plus récent met en cache mémoire le fournisseur CMP.
 4. Le fournisseur le plus récent utilise le fournisseur CMP pour générer les matériaux de chiffrement. Puis, il retourne les matériaux de chiffrement au chiffreur d'élément.

Obtention des matériaux de déchiffrement

Quand le chiffreur d'élément demande au fournisseur le plus récent les matériaux de chiffrement, le fournisseur le plus récent utilise le processus suivant pour les obtenir et les retourner.

1. Le fournisseur le plus récent demande au magasin de fournisseur le numéro de version des matériaux de chiffrement utilisés pour chiffrer l'élément. Il transmet la description du matériau réel depuis l'[attribut de description du matériau](#) de l'élément.
2. Le magasin de fournisseur obtient le numéro de version du CMP en charge du chiffrement auprès du champ `amzn-ddb-meta-id` de la description du matériau réel et le retourne au fournisseur le plus récent.
3. Le fournisseur le plus récent recherche dans son cache le numéro de version du CMP utilisé pour chiffrer et signer l'élément.

- S'il trouve que la version correspondante du CMP se trouve dans son cache et que le CMP n'a pas dépassé la [valeur time-to-live \(TTL\)](#), le fournisseur le plus récent utilise le CMP pour générer du matériel de déchiffrement. Puis, il retourne les matériaux de déchiffrement au chiffreur d'élément. Cette opération ne requiert pas d'appel au magasin de fournisseur ou à un autre fournisseur CMP.
- Si la version correspondante du CMP ne se trouve pas dans son cache, ou si le cache AWS KMS key a dépassé sa valeur TTL, le fournisseur le plus récent demande un CMP à son magasin de fournisseurs. Il envoie le nom de matériau et le numéro de version du CMP de chiffrement dans la demande.
 1. Le magasin de fournisseur recherche dans le stockage permanent le fournisseur CMP en utilisant le nom du fournisseur le plus récent comme clé de partition et le numéro de version comme clé de tri.
 - Si le nom et le numéro de version ne sont pas dans le stockage permanent, le magasin de fournisseur lève une exception. Si le magasin de fournisseur a été utilisé pour générer le CMP, celui-ci doit être stocké dans son stockage permanent, à moins qu'il ne soit intentionnellement supprimé.
 - Si le CMP avec le nom et le numéro de version correspondants se trouve dans le stockage permanent du magasin de fournisseur, celui-ci retourne le CMP spécifié au fournisseur le plus récent.

Si le magasin du fournisseur est un MetaStore, il obtient le CMP chiffré à partir de sa table DynamoDB. Puis, il utilise les matériaux de chiffrement de son fournisseur CMP interne pour déchiffrer le CMP chiffré avant de retourner le fournisseur CMP au fournisseur le plus récent. Si le CMP interne est un [fournisseur KMS direct](#), cette étape inclut un appel de [AWS Key Management Service](#) (AWS KMS).

2. Le fournisseur le plus récent met en cache mémoire le fournisseur CMP.
3. Le fournisseur le plus récent utilise le fournisseur CMP pour générer les matériaux de déchiffrement. Puis, il retourne les matériaux de déchiffrement au chiffreur d'élément.

Mises à jour du fournisseur le plus récent

Le symbole du fournisseur le plus récent est remplacé par `MostRecentProviderCachingMostRecentProvider`.

Note

Le `MostRecentProvider` symbole, qui représente le fournisseur le plus récent, est obsolète dans la version 1.15 du client de chiffrement DynamoDB pour Java et dans la version 1.3 du client de chiffrement DynamoDB pour Python et supprimé des versions 2.0.0 du client de chiffrement DynamoDB dans les deux implémentations linguistiques. Utilisez plutôt `leCachingMostRecentProvider`.

`CachingMostRecentProvider` implémente les modifications suivantes :

- Supprime `CachingMostRecentProvider` périodiquement les documents cryptographiques de la mémoire lorsque leur durée en mémoire dépasse la valeur configurée [time-to-live \(TTL\)](#).

Ils `MostRecentProvider` peuvent stocker du matériel cryptographique en mémoire pendant toute la durée de vie du processus. Par conséquent, le fournisseur le plus récent n'est peut-être pas au courant des modifications d'autorisation. Il peut utiliser des clés de chiffrement une fois que les autorisations de l'appelant à les utiliser ont été révoquées.

Si vous ne pouvez pas effectuer la mise à jour vers cette nouvelle version, vous pouvez obtenir un effet similaire en appelant régulièrement la `clear()` méthode dans le cache. Cette méthode vide manuellement le contenu du cache et oblige le fournisseur le plus récent à demander un nouveau CMP et de nouveaux matériaux cryptographiques.

- `CachingMostRecentProvider` inclut également un paramètre de taille de cache qui vous permet de mieux contrôler le cache.

Pour effectuer la mise à jour vers `leCachingMostRecentProvider`, vous devez modifier le nom du symbole dans votre code. À tous les autres égards, le `CachingMostRecentProvider` est entièrement rétrocompatible avec le `MostRecentProvider`. Il n'est pas nécessaire de chiffrer à nouveau les éléments du tableau.

Cependant, cela `CachingMostRecentProvider` génère davantage d'appels vers l'infrastructure clé sous-jacente. Il appelle le magasin du fournisseur au moins une fois par intervalle `time-to-live (TTL)`. Les applications avec de nombreuses applications actives CMPs (en raison de rotations fréquentes) ou les applications avec de grands flottes sont les plus susceptibles d'être sensibles à ce changement.

Avant de publier votre code mis à jour, testez-le minutieusement pour vous assurer que les appels les plus fréquents n'altèrent pas votre application ou n'entraînent pas de ralentissement de la part des services dont dépend votre fournisseur, tels que AWS Key Management Service (AWS KMS) ou Amazon DynamoDB. Pour atténuer les problèmes de performances, ajustez la taille du cache et celle time-to-live du en `CachingMostRecentProvider` fonction des caractéristiques de performance que vous observez. Pour de plus amples informations, consultez [Définition d'une time-to-live valeur](#).

Fournisseur de matériaux statique

Note

Notre bibliothèque de chiffrement côté client a été [renommée AWS Database Encryption SDK](#). La rubrique suivante fournit des informations sur les versions 1. x —2. x du client de chiffrement DynamoDB pour Java et versions 1. x —3. x du client de chiffrement DynamoDB pour Python. Pour plus d'informations, consultez la section [SDK AWS de chiffrement de base de données pour la prise en charge des versions DynamoDB](#).

Le fournisseur de matériaux statiques (Static CMP) est un [fournisseur de matériaux cryptographiques](#) (CMP) très simple destiné aux tests, aux proof-of-concept démonstrations et à la compatibilité existante.

Pour utiliser le fournisseur CMP statique afin de chiffrer un élément de table, vous fournissez une clé de chiffrement symétrique [AES \(Advanced Encryption Standard\)](#) et une clé ou paire de clés de signature. Vous devez fournir les mêmes clés pour déchiffrer l'élément chiffré. Le fournisseur CMP statique n'assure aucune opération de chiffrement. Au lieu de cela, il transmet inchangées les clés de chiffrement que vous fournissez au chiffreur d'élément. Le chiffreur d'élément chiffre les éléments directement sous la clé de chiffrement. Puis, il utilise directement la clé de signature pour les signer.

Comme le fournisseur CMP statique ne génère pas de matériau de chiffrement unique, tous les éléments de table que vous traitez sont chiffrés avec la même clé de chiffrement et signés par la même clé de signature. Lorsque vous utilisez la même clé pour chiffrer les valeurs d'attribut de nombreux éléments, ou que vous utilisez la même clé ou paire de clés pour signer tous les éléments, vous risquez de dépasser les limites de chiffrement des clés.

Note

Le [fournisseur statique asymétrique](#) de la bibliothèque Java n'est pas un fournisseur statique. Il fournit juste d'autres constructeurs au [fournisseur CMP encapsulé](#). Il est sûr pour une

utilisation en production, mais vous devez utiliser directement le CMP encapsulé chaque fois que possible.

Le CMP statique est l'un des nombreux [fournisseurs de matériel cryptographique](#) (CMPs) pris en charge par le client de chiffrement DynamoDB. Pour plus d'informations sur l'autre CMPs, voir [Fournisseur de matériel cryptographique](#).

Pour obtenir un exemple de code, consultez :

- Java: [SymmetricEncryptedItem](#)

Rubriques

- [Comment l'utiliser](#)
- [Comment ça marche](#)

Comment l'utiliser

Pour créer un fournisseur statique, fournissez une clé ou paire de clés de chiffrement, et une clé ou paire de clés de signature. Vous devez fournir le matériau de clé pour chiffrer et déchiffrer les éléments de table.

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 = ...
```

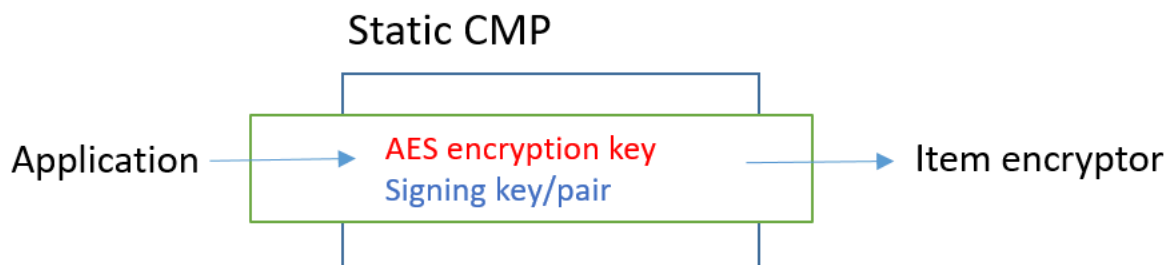
```
)

decrypt_keys = DecryptionMaterials(
    decryption_key = ...,
    verification_key = ...
)

static_cmp = StaticCryptographicMaterialsProvider(
    encryption_materials=encrypt_keys
    decryption_materials=decrypt_keys
)
```

Comment ça marche

Le fournisseur statique transmet les clés de chiffrement et de signature que vous fournissez au chiffreur d'élément, où elles sont utilisées directement pour chiffrer et signer vos éléments de table. À moins que vous ne fournissiez différentes clés pour chaque élément, les mêmes clés sont utilisées pour chaque élément.



Obtention des matériaux de chiffrement

Cette section décrit en détail les entrées, les sorties et le traitement du fournisseur CMP statique quand il reçoit une demande de matériaux de chiffrement.

Entrée (depuis l'application)

- Une clé de chiffrement — Il doit s'agir d'une clé symétrique, telle qu'une clé AES ([Advanced Encryption Standard](#)).
- Une clé de signature : il peut s'agir d'une clé symétrique ou d'une paire de clés asymétrique.

Entrée (depuis le chiffreur d'élément)

- [Contexte de chiffrement DynamoDB](#)

Sortie (vers le chiffreur d'élément)

- Clé de chiffrement transmise comme entrée.
- Clé de signature transmise comme entrée.
- Description de matériau réel : [description de matériau demandé](#), le cas échéant, inchangé.

Obtention des matériaux de déchiffrement

Cette section décrit en détail les entrées, les sorties et le traitement du fournisseur CMP statique quand il reçoit une demande de matériaux de déchiffrement.

Même s'il comporte des méthodes distinctes pour l'obtention des matériaux de chiffrement et celle des matériaux de déchiffrement, le comportement est le même.

Entrée (depuis l'application)

- Une clé de chiffrement — Il doit s'agir d'une clé symétrique, telle qu'une clé AES ([Advanced Encryption Standard](#)).
- Une clé de signature : il peut s'agir d'une clé symétrique ou d'une paire de clés asymétrique.

Entrée (depuis le chiffreur d'élément)

- Contexte de [chiffrement DynamoDB \(non utilisé\)](#)

Sortie (vers le chiffreur d'élément)

- Clé de chiffrement transmise comme entrée.
- Clé de signature transmise comme entrée.

Langages de programmation disponibles pour le client de chiffrement Amazon DynamoDB

Note

Notre bibliothèque de chiffrement côté client a été [renommée AWS Database Encryption SDK](#). La rubrique suivante fournit des informations sur les versions 1. x —2. x du client de chiffrement DynamoDB pour Java et versions 1. x —3. x du client de chiffrement DynamoDB pour Python. Pour plus d'informations, consultez la section [SDK AWS de chiffrement de base de données pour la prise en charge des versions DynamoDB](#).

Le client de chiffrement Amazon DynamoDB est disponible pour les langages de programmation suivants. Les bibliothèques spécifiques au langage varient, mais les implémentations qui en résultent sont interopérables. Par exemple, vous pouvez chiffrer (et signer) un élément avec le client Java et le déchiffrer avec le client Python.

Pour plus d'informations, consultez la rubrique correspondante.

Rubriques

- [Client de chiffrement Amazon DynamoDB pour Java](#)
- [Client de chiffrement DynamoDB pour Python](#)

Client de chiffrement Amazon DynamoDB pour Java

Note

Notre bibliothèque de chiffrement côté client a été [renommée AWS Database Encryption SDK](#). La rubrique suivante fournit des informations sur les versions 1. x —2. x du client de chiffrement DynamoDB pour Java et versions 1. x —3. x du client de chiffrement DynamoDB pour Python. Pour plus d'informations, consultez la section [SDK AWS de chiffrement de base de données pour la prise en charge des versions DynamoDB](#).

Cette rubrique explique comment installer et utiliser le client de chiffrement Amazon DynamoDB pour Java. [Pour plus de détails sur la programmation avec le client de chiffrement DynamoDB, consultez](#)

[les exemples Java, les exemples du référentiel GitHub et le Javadoc pour aws-dynamodb-encryption-java le client de chiffrement DynamoDB.](#)

Note

Versions 1. x. x du client de chiffrement DynamoDB pour Java sont [end-of-support en phase à compter de juillet 2022](#). Passez à une version plus récente dès que possible.

Rubriques

- [Conditions préalables](#)
- [Installation](#)
- [Utilisation du client de chiffrement DynamoDB pour Java](#)
- [Exemple de code pour le client de chiffrement DynamoDB pour Java](#)

Conditions préalables

Avant d'installer le client de chiffrement Amazon DynamoDB pour Java, assurez-vous de remplir les conditions préalables suivantes.

Environnement de développement Java

Vous aurez besoin de Java 8 ou version ultérieure. Sur le site web d'Oracle, consultez la page [Téléchargements Java SE](#), puis téléchargez et installez le kit Java SE Development (JDK).

Si vous utilisez le kit JDK Oracle, vous devez également télécharger et installer les [fichiers Java Cryptography Extension \(JCE\) Unlimited Strength Jurisdiction Policy](#).

AWS SDK pour Java

Le client de chiffrement DynamoDB nécessite le module DynamoDB même si votre application n'interagit pas avec DynamoDB. AWS SDK pour Java Vous pouvez installer la totalité du kit SDK ou le seul module. Si vous utilisez Maven, ajoutez `aws-java-sdk-dynamodb` à votre fichier `pom.xml`.

Pour plus d'informations sur l'installation et la configuration du AWS SDK pour Java, consultez [AWS SDK pour Java](#).

Installation

Vous pouvez installer le client de chiffrement Amazon DynamoDB pour Java de la manière suivante.

Manuellement

Pour installer le client de chiffrement Amazon DynamoDB pour Java, clonez ou téléchargez le référentiel. [aws-dynamodb-encryption-java](#) GitHub

Utilisation d'Apache Maven

Le client de chiffrement Amazon DynamoDB pour Java est disponible [via Apache](#) Maven avec la définition de dépendance suivante.

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-dynamodb-encryption-java</artifactId>
  <version>version-number</version>
</dependency>
```

Après avoir installé le SDK, commencez par consulter l'exemple de code présenté dans ce guide et le Javadoc du client de [chiffrement DynamoDB activé](#). GitHub

Utilisation du client de chiffrement DynamoDB pour Java

Note

Notre bibliothèque de chiffrement côté client a été [renommée AWS Database Encryption SDK](#). La rubrique suivante fournit des informations sur les versions 1. x —2. x du client de chiffrement DynamoDB pour Java et versions 1. x —3. x du client de chiffrement DynamoDB pour Python. Pour plus d'informations, consultez la section [SDK AWS de chiffrement de base de données pour la prise en charge des versions DynamoDB](#).

Cette rubrique décrit certaines fonctionnalités du client de chiffrement DynamoDB en Java qui ne sont peut-être pas disponibles dans d'autres implémentations de langage de programmation.

[Pour plus de détails sur la programmation avec le client de chiffrement DynamoDB, consultez les exemples Java, les exemples ci-dessous et le Javadoc pour aws-dynamodb-encryption-java repository](#) GitHub le client de chiffrement DynamoDB.

Rubriques

- [Crypteurs d'objets : AttributeEncryptor et Dynamo DBEncryptor](#)
- [Configuration du comportement d'enregistrement](#)
- [Actions d'attribut en Java](#)
- [Remplacer les noms des tables](#)

Crypteurs d'objets : AttributeEncryptor et Dynamo DBEncryptor

[Le client de chiffrement DynamoDB en Java possède deux chiffreurs d'éléments : le Dynamo de niveau inférieur et le. DBEncryptor AttributeEncryptor](#)

AttributeEncryptor est une classe d'assistance qui vous aide à utiliser le [Dynamo DBMapper](#) DynamoDB Encryptor dans le AWS SDK pour Java client de chiffrement DynamoDB. Lorsque vous utilisez le AttributeEncryptor avec le DynamoDBMapper, il chiffre et signe vos éléments de manière transparente lorsque vous les enregistrez. Il vérifie et déchiffre également vos éléments de manière transparente lorsque vous les chargez.

Configuration du comportement d'enregistrement

Vous pouvez utiliser le AttributeEncryptor et DynamoDBMapper pour ajouter ou remplacer des éléments de tableau par des attributs signés uniquement ou chiffrés et signés. Pour ces tâches, nous vous recommandons de le configurer pour utiliser le comportement d'enregistrement PUT, comme illustré dans l'exemple suivant. Sinon, il est possible que vous ne puissiez pas déchiffrer vos données.

```
DynamoDBMapperConfig mapperConfig =
    DynamoDBMapperConfig.builder().withSaveBehavior(SaveBehavior.PUT).build();
DynamoDBMapper mapper = new DynamoDBMapper(ddb, mapperConfig, new
    AttributeEncryptor(encryptor));
```

Si vous utilisez le comportement d'enregistrement par défaut, qui met à jour uniquement les attributs modélisés dans l'élément de table, les attributs non modélisés ne sont pas inclus dans la signature et ne sont pas modifiés par les écritures de table. Par conséquent, lors de lectures ultérieures de tous les attributs, la signature ne sera pas validée, car elle n'inclut pas d'attributs non modélisés.

Vous pouvez également utiliser le comportement de sauvegarde CLOBBER. Comportement d'enregistrement est identique au comportement d'enregistrement PUT, si ce n'est qu'il désactive le verrouillage optimiste et remplace l'élément dans la table.

Pour éviter les erreurs de signature, le client de chiffrement DynamoDB lance une exception d'exécution si un est utilisé avec `AttributeEncryptor` un qui n'est pas configuré avec `DynamoDBMapper` un comportement d'enregistrement de ou. `CLOBBER PUT`

Pour voir ce code utilisé dans un exemple, consultez [Utilisation de la Dynamo DBMapper](#) l'exemple [AwsKmsEncryptedObject.java](#) dans le `aws-dynamodb-encryption-java` référentiel dans GitHub.

Actions d'attribut en Java

Les [actions d'attribut](#) déterminent les valeurs d'attribut chiffrées et signées, qui sont uniquement signées et qui sont ignorées. La méthode que vous utilisez pour spécifier les actions attributaires varie selon que vous utilisez le `DynamoDBMapper` et `AttributeEncryptor` ou le [DynamoDBEncryptor](#) de niveau inférieur.

Important

Après avoir utilisé vos actions d'attribut pour chiffrer vos éléments de table, l'ajout ou la suppression d'attributs de votre modèle de données peut provoquer une erreur de validation de signature qui vous empêche de déchiffrer vos données. Pour obtenir une explication détaillée, consultez [Modification de votre modèle de données](#).

Actions attributaires pour le Dynamo DBMapper

Quand vous utilisez les `DynamoDBMapper` et `AttributeEncryptor`, vous utilisez les annotations pour spécifier les actions d'attribut. Le client de chiffrement DynamoDB utilise les annotations d'attribut [DynamoDB standard qui définissent le type d'attribut](#) afin de déterminer comment protéger un attribut. Par défaut, tous les attributs sont chiffrés et signés à l'exception des clés primaires, qui sont signées, mais pas chiffrées.

Note

Ne chiffrez pas la valeur des attributs avec l'[annotation @DynamoDBVersion Attribute](#), bien que vous puissiez (et devriez) les signer. Sinon, les conditions qui utilisent sa valeur auront des effets inattendus.

```
// 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")
```

Pour spécifier des exceptions, utilisez les annotations de chiffrement définies dans le client de chiffrement DynamoDB pour Java. Si vous les spécifiez au niveau classe, elles deviennent la valeur par défaut pour la classe.

```
// Sign only
@DoNotEncrypt

// Do nothing; not encrypted or signed
@DoNotTouch
```

Par exemple, ces annotations signent mais ne chiffrent pas l'attribut `PublicationYear`, et ni ne chiffrent ou ne signent la valeur d'attribut `ISBN`.

```
// Sign only (override the default)
@DoNotEncrypt
@DynamoDBAttribute(attributeName="PublicationYear")

// Do nothing (override the default)
@DoNotTouch
@DynamoDBAttribute(attributeName="ISBN")
```


Actions attributaires pour le Dynamo DBEncryptor

Pour spécifier des actions d'attribut lorsque vous utilisez DBEncryptor directement le [Dynamo](#), créez un `HashMap` objet dans lequel les paires nom-valeur représentent les noms d'attributs et les actions spécifiées.

Les valeurs valides des actions d'attribut sont définies dans le type énuméré `EncryptionFlags`. Vous pouvez utiliser `ENCRYPT` et `SIGN` conjointement, utiliser `SIGN` seul, ou omettre les deux. Toutefois, si vous l'utilisez `ENCRYPT` seul, le client de chiffrement DynamoDB génère une erreur. Vous ne pouvez pas chiffrer un attribut que vous ne signez pas.

```
ENCRYPT
```

SIGN

 Warning

Ne chiffrez pas les attributs de la clé primaire. Ils doivent rester en texte brut pour que DynamoDB puisse trouver l'élément sans exécuter une analyse complète du tableau.

Si vous spécifiez une clé primaire dans le contexte de chiffrement, puis que vous la spécifiez ENCRYPT dans l'action d'attribut pour l'un ou l'autre des attributs de clé primaire, le client de chiffrement DynamoDB émet une exception.

Par exemple, le code Java suivant crée un code actions HashMap qui chiffre et signe tous les attributs de l'recordélément. Les exceptions sont les attributs de clé de partition et de clé de tri, qui sont signés mais non chiffrés, et l'attribut test, qui n'est ni signé ni chiffré.

```
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;
    }
}
```

Puis, quand vous appelez la méthode [encryptRecord](#) de `DynamoDBEncryptor`, spécifiez la map comme valeur du paramètre `attributeFlags`. Par exemple, cet appel d'`encryptRecord` utilise la map `actions`.

```
// Encrypt the plaintext record
final Map<String, AttributeValue> encrypted_record = encryptor.encryptRecord(record,
    actions, encryptionContext);
```

Remplacer les noms des tables

Dans le client de chiffrement DynamoDB, le nom de la table DynamoDB est un élément du contexte de chiffrement [DynamoDB transmis aux méthodes de chiffrement et de déchiffrement](#). Lorsque vous chiffrez ou signez des éléments de table, le contexte de chiffrement DynamoDB, y compris le nom de la table, est lié cryptographiquement au texte chiffré. Si le contexte de chiffrement DynamoDB transmis à la méthode de déchiffrement ne correspond pas au contexte de chiffrement DynamoDB transmis à la méthode de chiffrement, l'opération de déchiffrement échoue.

Il arrive parfois que le nom d'une table change, par exemple lorsque vous sauvegardez une table ou que vous effectuez une [point-in-time restauration](#). Lorsque vous déchiffrez ou vérifiez la signature de ces éléments, vous devez transmettre le même contexte de chiffrement DynamoDB que celui utilisé pour chiffrer et signer les éléments, y compris le nom de table d'origine. Le nom de la table actuelle n'est pas nécessaire.

Lorsque vous utilisez le `DynamoDBEncryptor`, vous assemblez le contexte de chiffrement DynamoDB manuellement. Toutefois, si vous utilisez le `DynamoDBMapper`, `AttributeEncryptor` crée le contexte de chiffrement DynamoDB pour vous, y compris le nom de la table actuelle. Pour indiquer à `AttributeEncryptor` de créer un contexte de chiffrement avec un nom de table différent, utilisez le `EncryptionContextOverrideOperator`.

Par exemple, le code suivant crée des instances du fournisseur de matériaux cryptographiques (CMP) et du `DynamoDBEncryptor`. Ensuite, il appelle la méthode `setEncryptionContextOverrideOperator` de `DynamoDBEncryptor`. Il utilise l'opérateur `overrideEncryptionContextTableName`, qui remplace un nom de table. Lorsqu'il est configuré de cette façon, il `AttributeEncryptor` crée un contexte de chiffrement DynamoDB qui `newTableName` inclut à la place de `oldTableName`. Pour un exemple complet, consultez [EncryptionContextOverridesWithDynamoDBMapper.java](#).

```
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, keyArn);
final DynamoDBEncryptor encryptor = DynamoDBEncryptor.getInstance(cmp);

encryptor.setEncryptionContextOverrideOperator(EncryptionContextOperators.overrideEncryptionContextTableName(
    oldTableName, newTableName));
```

Lorsque vous appelez la méthode de chargement de `DynamoDBMapper`, qui déchiffre et vérifie l'élément, vous spécifiez le nom de la table d'origine.

```
mapper.load(itemClass, DynamoDBMapperConfig.builder()  
    .withTableNameOverride(DynamoDBMapperConfig.TableNameOverride.withTableNameReplacement(oldTable, newTable))  
    .build());
```

Vous pouvez également utiliser l'opérateur `overrideEncryptionContextTableNameUsingMap`, qui remplace plusieurs noms de table.

Les opérateurs de remplacement de nom de table sont généralement utilisés lors du déchiffrement des données et de la vérification des signatures. Vous pouvez toutefois les utiliser pour attribuer une valeur différente au nom de la table dans le contexte de chiffrement DynamoDB lors du chiffrement et de la signature.

N'utilisez pas les opérateurs de remplacement de nom de table si vous utilisez le `DynamoDBEncryptor`. Créez plutôt un contexte de chiffrement avec le nom de la table d'origine et soumettez-le à la méthode de déchiffrement.

Exemple de code pour le client de chiffrement DynamoDB pour Java

Note

Notre bibliothèque de chiffrement côté client a été [renommée AWS Database Encryption SDK](#). La rubrique suivante fournit des informations sur les versions 1. x —2. x du client de chiffrement DynamoDB pour Java et versions 1. x —3. x du client de chiffrement DynamoDB pour Python. Pour plus d'informations, consultez la section [SDK AWS de chiffrement de base de données pour la prise en charge des versions DynamoDB](#).

Les exemples suivants montrent comment utiliser le client de chiffrement DynamoDB pour Java afin de protéger les éléments de table DynamoDB dans votre application. Vous pouvez trouver d'autres exemples (et apporter les vôtres) dans le répertoire des [exemples](#) du [aws-dynamodb-encryption-javaréférentiel](#) sur GitHub.

Rubriques

- [Utilisation de la Dynamo DBEncryptor](#)
- [Utilisation de la Dynamo DBMapper](#)

Utilisation de la Dynamo DBEncryptor

Cet exemple montre comment utiliser le [Dynamo](#) de niveau inférieur DBEncryptor avec le fournisseur [Direct KMS](#). Le fournisseur Direct KMS génère et protège son matériel cryptographique sous un [AWS KMS key](#)in AWS Key Management Service (AWS KMS) que vous spécifiez.

Vous pouvez utiliser n'importe quel [fournisseur de matériel cryptographique](#) (CMP) compatible avec leDynamoDBEncryptor, et vous pouvez utiliser le fournisseur Direct KMS avec le DynamoDBMapper et. [AttributeEncryptor](#)

Voir l'exemple de code complet : [AwsKmsEncryptedItem.java](#)

Étape 1 : Créer le fournisseur KMS direct

Créez une instance du AWS KMS client avec la région spécifiée. Utilisez ensuite l'instance client pour créer une instance du fournisseur Direct KMS selon vos préférences AWS KMS key.

Cet exemple utilise l'Amazon Resource Name (ARN) pour identifier le AWS KMS key, mais vous pouvez utiliser [n'importe quel identifiant de clé valide](#).

```
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);
```

Étape 2 : Créer un élément

Cet exemple définit un record HashMap qui représente un exemple d'élément de table.

```
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"));
```

Étape 3 : Création d'une dynamo DBEncryptor

Créez une instance de `DynamoDBEncryptor` avec le fournisseur KMS direct.

```
final DynamoDBEncryptor encryptor = DynamoDBEncryptor.getInstance(cmp);
```

Étape 4 : Création d'un contexte de chiffrement DynamoDB

Le contexte de [chiffrement DynamoDB](#) contient des informations sur la structure de la table et sur la manière dont elle est chiffrée et signée. Si vous utilisez `DynamoDBMapper`, `AttributeEncryptor` crée automatiquement le contexte de chiffrement.

```
final String tableName = "testTable";

final EncryptionContext encryptionContext = new EncryptionContext.Builder()
    .withTableName(tableName)
    .withHashKeyName(partitionKeyName)
    .withRangeKeyName(sortKeyName)
    .build();
```

Étape 5 : Créer l'objet actions d'attribut

Les [actions d'attribut](#) déterminent quels attributs de l'élément sont chiffrés et signés, lesquels sont uniquement signés et lesquels ne sont ni chiffrés ni signés.

En Java, pour spécifier des actions d'attribut, vous créez des paires `HashMap` de nom d'attribut et de `EncryptionFlags` valeur.

Par exemple, le code Java suivant crée un actions `HashMap` qui chiffre et signe tous les attributs de l'élément, à l'exception des attributs de clé de partition et de clé de tri, qui sont signés, mais non chiffrés, et de l'attribut `testattribut`, qui n'est ni signé ni chiffré.

```
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;
}
}
```

Étape 6 : Chiffrer et signer l'élément

Pour chiffrer et signer l'élément de table, appelez la méthode `encryptRecord` sur l'instance de `DynamoDBEncryptor`. Spécifiez l'élément de table (`record`), les actions d'attribut (`actions`) et le contexte de chiffrement (`encryptionContext`).

```
final Map<String, AttributeValue> encrypted_record = encryptor.encryptRecord(record,
    actions, encryptionContext);
```

Étape 7 : Placer l'élément dans la table DynamoDB

Enfin, placez l'élément chiffré et signé dans la table DynamoDB.

```
final AmazonDynamoDB ddb = AmazonDynamoDBClientBuilder.defaultClient();
ddb.putItem(tableName, encrypted_record);
```

Utilisation de la Dynamo DBMapper

[L'exemple suivant montre comment utiliser la classe d'assistance du mappeur DynamoDB avec le fournisseur Direct KMS](#). Le fournisseur Direct KMS génère et protège son matériel cryptographique sous un [AWS KMS key](#) in AWS Key Management Service (AWS KMS) que vous spécifiez.

Vous pouvez utiliser tout [fournisseur CMP](#) compatible avec `DynamoDBMapper`, et vous pouvez utiliser le fournisseur KMS direct avec le `DynamoDBEncryptor` de bas niveau.

Voir l'exemple de code complet : [AwsKmsEncryptedObject.java](#)

Étape 1 : Créer le fournisseur KMS direct

Créez une instance du AWS KMS client avec la région spécifiée. Utilisez ensuite l'instance client pour créer une instance du fournisseur Direct KMS selon vos préférences AWS KMS key.

Cet exemple utilise l'Amazon Resource Name (ARN) pour identifier le AWS KMS key, mais vous pouvez utiliser [n'importe quel identifiant de clé valide](#).

```
final String keyArn = "arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";  
final String region = "us-west-2";  
  
final AWKMS kms = AWKMSClientBuilder.standard().withRegion(region).build();  
final DirectKmsMaterialProvider cmp = new DirectKmsMaterialProvider(kms, keyArn);
```

Étape 2 : Création du chiffreur DynamoDB et de Dynamo DBMapper

Utilisez le fournisseur Direct KMS que vous avez créé à l'étape précédente pour créer une instance de [DynamoDB Encryptor](#). Vous devez instancier le chiffreur DynamoDB de niveau inférieur pour utiliser le mappeur DynamoDB.

Créez ensuite une instance de votre base de données DynamoDB et une configuration de mappeur, puis utilisez-les pour créer une instance du mappeur DynamoDB.

Important

Lorsque vous utilisez DynamoDBMapper pour ajouter ou modifier des éléments signés (ou chiffrés et signés), configurez-le pour qu'il [utilise un comportement d'enregistrement](#), par exemple PUT, qui inclut tous les attributs, comme illustré dans l'exemple suivant. Sinon, il est possible que vous ne puissiez pas déchiffrer vos données.

```
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));
```

Étape 3 : définition de votre table DynamoDB

Définissez ensuite votre table DynamoDB. Utilisez des annotations pour spécifier les [actions d'attribut](#). Cet exemple crée une table DynamoDB et une classe qui `ExampleTable` représente les éléments DataPoJo de la table.

Dans cet exemple de table, les attributs de la clé primaire seront signés, mais pas chiffrés. Cela s'applique à l'attribut `partition_attribute`, qui est annoté avec `@DynamoDBHashKey`, et l'attribut `sort_attribute`, qui est annoté avec `@DynamoDBRangeKey`.

Les attributs qui sont annotés avec `@DynamoDBAttribute`, par exemple `some numbers`, seront chiffrés et signés. Les exceptions sont les attributs qui utilisent les annotations de chiffrement `@DoNotEncrypt` (signer uniquement) ou `@DoNotTouch` (ne pas chiffrer ni signer) définies par le client de chiffrement DynamoDB. Par exemple, étant donné que l'attribut `leave me` comporte une annotation `@DoNotTouch`, il ne sera ni chiffré ni signé.

```
@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 +
        "];";
}
```

```
}  
}
```

Étape 4 : Chiffrer et enregistrer un élément de table

Désormais, lorsque vous créez un élément de table et que vous utilisez le mappeur DynamoDB pour l'enregistrer, l'élément est automatiquement chiffré et signé avant d'être ajouté à la table.

Cet exemple définit un élément de table appelé `record`. Avant qu'il soit enregistré dans la table, ses attributs sont chiffrés et signés selon les annotations de la classe `DataPoJo`. Dans le cas présent, tous les attributs à l'exception de `PartitionAttribute`, `SortAttribute` et `LeaveMe` sont chiffrés et signés. `PartitionAttribute` et `SortAttributes` sont seulement signés. L'attribut `LeaveMe` n'est ni chiffré ni signé.

Pour chiffrer et signer l'élément `record`, puis l'ajouter à `ExampleTable`, appelez la méthode `save` de la classe `DynamoDBMapper`. Votre mappeur DynamoDB étant configuré pour utiliser PUT le comportement d'enregistrement, l'élément remplace tout élément possédant les mêmes clés primaires, au lieu de le mettre à jour. Cela garantit que les signatures correspondent et que vous pouvez déchiffrer l'élément lorsque vous le récupérez de la table.

```
DataPoJo record = new DataPoJo();  
record.setPartitionAttribute("is this");  
record.setSortAttribute(55);  
record.setExample("data");  
record.setSomeNumbers(99);  
record.setSomeBinary(new byte[]{0x00, 0x01, 0x02});  
record.setLeaveMe("alone");  
  
mapper.save(record);
```

Client de chiffrement DynamoDB pour Python

Note

Notre bibliothèque de chiffrement côté client a été [renommée AWS Database Encryption SDK](#). La rubrique suivante fournit des informations sur les versions 1. x —2. x du client de chiffrement DynamoDB pour Java et versions 1. x —3. x du client de chiffrement DynamoDB pour Python. Pour plus d'informations, consultez la section [SDK AWS de chiffrement de base de données pour la prise en charge des versions DynamoDB](#).

Cette rubrique explique comment installer et utiliser le client de chiffrement DynamoDB pour Python. Vous pouvez trouver le code dans le [aws-dynamodb-encryption-python](#) référentiel sur GitHub, y compris un [exemple de code](#) complet et testé pour vous aider à démarrer.

Note

Versions 1. x. x et 2. x. x du client de chiffrement DynamoDB pour Python sont [end-of-support en phase à](#) compter de juillet 2022. Passez à une version plus récente dès que possible.

Rubriques

- [Conditions préalables](#)
- [Installation](#)
- [Utilisation du client de chiffrement DynamoDB pour Python](#)
- [Exemple de code pour le client de chiffrement DynamoDB pour Python](#)

Conditions préalables

Avant d'installer le client de chiffrement Amazon DynamoDB pour Python, assurez-vous de remplir les conditions préalables suivantes.

Version prise en charge de Python

Python 3.8 ou version ultérieure est requis par le client de chiffrement Amazon DynamoDB pour les versions 3.3.0 et ultérieures de Python. Pour télécharger Python, consultez [Téléchargements Python](#).

Les versions antérieures du client de chiffrement Amazon DynamoDB pour Python prennent en charge Python 2.7, Python 3.4 et versions ultérieures, mais nous vous recommandons d'utiliser la dernière version du client de chiffrement DynamoDB.

Outil d'installation pip pour Python

Python 3.6 et versions ultérieures incluent pip, bien que vous souhaitiez peut-être le mettre à niveau. Pour plus d'informations sur la mise à niveau ou l'installation de pip, consultez [Installation](#) dans la documentation pip.

Installation

Utilisez pip pour installer le client de chiffrement Amazon DynamoDB pour Python, comme indiqué dans les exemples suivants.

Pour installer la dernière version

```
pip install dynamodb-encryption-sdk
```

Pour plus d'informations sur l'utilisation de pip pour installer et mettre à niveau les packages, consultez [Installing Packages](#).

Le client de chiffrement DynamoDB nécessite la bibliothèque de chiffrement [sur toutes les plateformes](#). Toutes les versions de pip installent et créent la bibliothèque de chiffrement sous Windows. pip 8.1 et les versions ultérieures installent et créent la bibliothèque de chiffrement sous Linux. Si vous utilisez une version antérieure de pip et que votre environnement Linux ne possède pas les outils nécessaires pour générer la bibliothèque de chiffrement, vous devez les installer. Pour plus d'informations, consultez [Création du chiffrement sous Linux](#).

Vous pouvez obtenir la dernière version de développement du client de chiffrement DynamoDB à partir du [aws-dynamodb-encryption-python](#) référentiel. GitHub

Après avoir installé le client de chiffrement DynamoDB, commencez par consulter l'exemple de code Python présenté dans ce guide.

Utilisation du client de chiffrement DynamoDB pour Python

Note

Notre bibliothèque de chiffrement côté client a été [renommée AWS Database Encryption SDK](#). La rubrique suivante fournit des informations sur les versions 1. x —2. x du client de chiffrement DynamoDB pour Java et versions 1. x —3. x du client de chiffrement DynamoDB pour Python. Pour plus d'informations, consultez la section [SDK AWS de chiffrement de base de données pour la prise en charge des versions DynamoDB](#).

Cette rubrique décrit certaines fonctionnalités du client de chiffrement DynamoDB pour Python qui ne sont peut-être pas disponibles dans d'autres implémentations de langage de programmation.

Ces fonctionnalités sont conçues pour faciliter l'utilisation du client de chiffrement DynamoDB de la manière la plus sécurisée qui soit. À moins que vous n'ayez un scénario inhabituel, nous vous recommandons de les utiliser.

Pour plus de détails sur la programmation avec le client de chiffrement DynamoDB, consultez les exemples [Python](#) de ce guide, [les exemples du référentiel](#) et [aws-dynamodb-encryption-python](#) la documentation GitHub [Python](#) du client de chiffrement DynamoDB.

Rubriques

- [Classes d'annotations clientes](#)
- [TableInfo classe](#)
- [Actions d'attribut en Python](#)

Classes d'annotations clientes

Le client de chiffrement DynamoDB pour Python inclut plusieurs classes d'assistance client qui reflètent les classes Boto 3 pour DynamoDB. Ces classes d'assistance sont conçues pour faciliter l'ajout de chiffrement et de signature à votre application DynamoDB existante et pour éviter les problèmes les plus courants, comme suit :

- Empêchez-vous de chiffrer la clé primaire de votre élément, soit en ajoutant une action de remplacement pour la clé primaire à l'[AttributeActions](#) objet, soit en lançant une exception si votre [AttributeActions](#) objet demande explicitement au client de chiffrer la clé primaire. Si l'action par défaut de votre objet [AttributeActions](#) est `DO_NOTHING`, les classes d'annotations clientes utilisent cette action pour la clé primaire. Sinon, elles utilisent `SIGN_ONLY`.
- Créez un [TableInfo objet](#) et renseignez le contexte de [chiffrement DynamoDB](#) en fonction d'un appel à DynamoDB. Cela permet de garantir que votre contexte de chiffrement DynamoDB est précis et que le client peut identifier la clé primaire.
- Support des méthodes, telles que `put_item` et `get_item`, qui chiffrent et déchiffrent de manière transparente les éléments de votre table lorsque vous écrivez ou lisez dans une table DynamoDB. Seule la méthode `update_item` n'est pas prise en charge.

Vous pouvez utiliser les classes d'annotations clientes au lieu d'interagir directement avec le [chiffreur d'élément](#) de bas niveau. Utilisez ces classes à moins que vous n'ayez besoin de définir des options avancées dans le chiffreur d'élément.

Les classes d'annotations clientes incluent les éléments suivants :

- [EncryptedTable](#) pour les applications qui utilisent la ressource [Table](#) dans DynamoDB pour traiter une table à la fois.
- [EncryptedResource](#) pour les applications qui utilisent la classe [Service Resource](#) dans DynamoDB pour le traitement par lots.
- [EncryptedClient](#) pour les applications qui utilisent le [client de niveau inférieur dans DynamoDB](#).

Pour utiliser les classes d'assistance client, l'appelant doit être autorisé à appeler l'opération DynamoDB sur la table cible [DescribeTable](#).

TableInfo classe

La [TableInfo](#) classe est une classe auxiliaire qui représente une table DynamoDB, avec des champs pour sa clé primaire et ses index secondaires. Elle vous permet d'obtenir des informations précises et en temps réel sur la table.

Si vous utilisez une [classe d'annotations clientes](#), elle crée et utilise un objet `TableInfo` pour vous. Sinon, vous pouvez en créer un explicitement. Pour obtenir un exemple, consultez [Utilisation du chiffreur d'élément](#).

Lorsque vous appelez la `refresh_indexed_attributes` méthode sur un `TableInfo` objet, elle renseigne les valeurs des propriétés de l'objet en appelant l'opération DynamoDB [DescribeTable](#). L'interrogation de la table est beaucoup plus fiable que le codage en dur des noms d'index. La `TableInfo` classe inclut également une `encryption_context_values` propriété qui fournit les valeurs requises pour le contexte de [chiffrement DynamoDB](#).

Pour utiliser la `refresh_indexed_attributes` méthode, l'appelant doit être autorisé à appeler l'opération [DescribeTable](#) DynamoDB sur la table cible.

Actions d'attribut en Python

Les [actions d'attribut](#) informent le chiffreur d'élément des actions à exécuter sur chaque attribut de l'élément. Pour spécifier les actions d'attribut en Python, créez un objet `AttributeActions` avec une action par défaut et les exceptions éventuelles pour des attributs particuliers. Les valeurs valides sont définies dans le type énuméré `CryptoAction`.

Important

Après avoir utilisé vos actions d'attribut pour chiffrer vos éléments de table, l'ajout ou la suppression d'attributs de votre modèle de données peut provoquer une erreur de validation

de signature qui vous empêche de déchiffrer vos données. Pour obtenir une explication détaillée, consultez [Modification de votre modèle de données](#).

```
DO_NOTHING = 0
SIGN_ONLY = 1
ENCRYPT_AND_SIGN = 2
```

Par exemple, l'objet `AttributeActions` établit `ENCRYPT_AND_SIGN` comme valeur par défaut pour tous les attributs, et spécifie les exceptions pour les attributs `ISBN` et `PublicationYear`.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={
        'ISBN': CryptoAction.DO_NOTHING,
        'PublicationYear': CryptoAction.SIGN_ONLY
    }
)
```

Si vous utilisez une [classe d'annotations clientes](#), vous n'avez pas besoin de spécifier une action d'attribut pour les attributs de clé primaire. Les classes d'annotations clients vous empêchent de chiffrer votre clé primaire.

Si vous n'utilisez pas une classe d'annotations clientes et que l'action par défaut est `ENCRYPT_AND_SIGN`, vous devez spécifier une action pour la clé primaire. L'action recommandée pour les clés primaires est `SIGN_ONLY`. À des fins de simplification, utilisez la méthode `set_index_keys`, qui utilise `SIGN_ONLY` pour les clés primaires ou `DO_NOTHING` quand il s'agit de l'action par défaut.

Warning

Ne chiffrez pas les attributs de la clé primaire. Ils doivent rester en texte brut pour que DynamoDB puisse trouver l'élément sans exécuter une analyse complète du tableau.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
)
```

```
actions.set_index_keys(*table_info.protected_index_keys())
```

Exemple de code pour le client de chiffrement DynamoDB pour Python

Note

Notre bibliothèque de chiffrement côté client a été [renommée AWS Database Encryption SDK](#). La rubrique suivante fournit des informations sur les versions 1. x —2. x du client de chiffrement DynamoDB pour Java et versions 1. x —3. x du client de chiffrement DynamoDB pour Python. Pour plus d'informations, consultez la section [SDK AWS de chiffrement de base de données pour la prise en charge des versions DynamoDB](#).

Les exemples suivants montrent comment utiliser le client de chiffrement DynamoDB pour Python afin de protéger les données DynamoDB dans votre application. Vous pouvez trouver d'autres exemples (et apporter les vôtres) dans le répertoire des [exemples](#) du [aws-dynamodb-encryption-python](#) référentiel sur GitHub.

Rubriques

- [Utiliser la classe d'assistance EncryptedTable client](#)
- [Utilisation du chiffreur d'élément](#)

Utiliser la classe d'assistance EncryptedTable client

L'exemple suivant montre comment utiliser le [fournisseur KMS direct](#) avec la `EncryptedTable` [classe d'annotations cliente](#). Cet exemple utilise le même [fournisseur CMP](#) que l'exemple [Utilisation du chiffreur d'élément](#) qui suit. Cependant, il utilise la classe `EncryptedTable` au lieu d'interagir directement avec le [chiffreur d'élément](#) de bas niveau.

En comparant ces exemples, vous pouvez voir le travail que la classe d'annotations cliente effectue à votre place. Cela implique de créer le contexte de [chiffrement DynamoDB](#) et de s'assurer que les attributs de clé primaire sont toujours signés, mais jamais chiffrés. Pour créer le contexte de chiffrement et découvrir la clé primaire, les classes d'assistance client appellent l'opération DynamoDB [DescribeTable](#). Pour exécuter ce code, vous devez avoir l'autorisation d'appeler cette opération.

Consultez l'exemple de code complet : [aws_kms_encrypted_table.py](#)

Étape 1 : Créer la table

Commencez par créer une instance d'une table DynamoDB standard avec le nom de la table.

```
table_name='test-table'  
table = boto3.resource('dynamodb').Table(table_name)
```

Étape 2 : Créer un fournisseur CMP

Créez une instance du [fournisseur CMP](#) que vous avez sélectionné.

Cet exemple utilise le [fournisseur KMS direct](#), mais vous pouvez utiliser n'importe quel fournisseur CMP compatible. Pour créer un fournisseur KMS direct, spécifiez un [AWS KMS key](#). Cet exemple utilise le Amazon Resource Name (ARN) du AWS KMS key, mais vous pouvez utiliser n'importe quel identifiant de clé valide.

```
kms_key_id='arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'  
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key_id)
```

Étape 3 : Créer l'objet actions d'attribut

Les [actions d'attribut](#) informent le chiffreur d'élément des actions à exécuter sur chaque attribut de l'élément. L'objet `AttributeActions` de l'exemple chiffre et signe tous les éléments à l'exception de l'attribut `test`, qui est ignoré.

Ne spécifiez pas d'actions d'attribut pour les attributs de clé primaire quand vous utilisez une classe d'annotations cliente. La classe `EncryptedTable` signe, mais ne chiffre jamais, les attributs de clé primaire.

```
actions = AttributeActions(  
    default_action=CryptoAction.ENCRYPT_AND_SIGN,  
    attribute_actions={'test': CryptoAction.DO_NOTHING}  
)
```

Étape 4 : Créer la table chiffrée

Créez la table chiffrée à l'aide de la table standard, du fournisseur KMS direct et des actions d'attribut. Cette étape termine la configuration.

```
encrypted_table = EncryptedTable(  

```

```
    table=table,  
    materials_provider=kms_cmp,  
    attribute_actions=actions  
)
```

Étape 5 : Placer l'élément en texte brut dans la table

Lorsque vous appelez la `put_item` méthode sur `leencrypted_table`, les éléments de votre table sont chiffrés, signés et ajoutés de manière transparente à votre table DynamoDB.

Tout d'abord, définissez l'élément de table.

```
plaintext_item = {  
    'partition_attribute': 'value1',  
    'sort_attribute': 55  
    'example': 'data',  
    'numbers': 99,  
    'binary': Binary(b'\x00\x01\x02'),  
    'test': 'test-value'  
}
```

Puis, insérez-le dans la table.

```
encrypted_table.put_item(Item=plaintext_item)
```

Pour obtenir l'élément de la table DynamoDB sous sa forme cryptée, appelez `get_item` la méthode sur l'objet `table`. Pour obtenir l'élément chiffré, appelez la méthode `get_item` sur l'objet `encrypted_table`.

Utilisation du chiffreur d'élément

Cet exemple montre comment interagir directement avec le chiffreur d'[éléments dans le client de chiffrement DynamoDB lors du chiffrement d'éléments de table, au lieu d'utiliser les classes d'assistance du client qui interagissent avec le crypteur](#) d'éléments à votre place.

Lorsque vous utilisez cette technique, vous créez le contexte de chiffrement DynamoDB et l'objet `CryptoConfig` de configuration () manuellement. Vous cryptez également les éléments lors d'un appel et vous les placez dans votre table DynamoDB lors d'un appel distinct. Cela vous permet de personnaliser vos `put_item` appels et d'utiliser le client de chiffrement DynamoDB pour chiffrer et signer les données structurées qui ne sont jamais envoyées à DynamoDB.

Cet exemple utilise le [fournisseur KMS direct](#), mais vous pouvez utiliser n'importe quel fournisseur CMP compatible.

Voir l'exemple de code complet : [aws_kms_encrypted_item.py](#)

Étape 1 : Créer la table

Commencez par créer une instance d'une ressource de table DynamoDB standard avec le nom de la table.

```
table_name='test-table'  
table = boto3.resource('dynamodb').Table(table_name)
```

Étape 2 : Créer un fournisseur CMP

Créez une instance du [fournisseur CMP](#) que vous avez sélectionné.

Cet exemple utilise le [fournisseur KMS direct](#), mais vous pouvez utiliser n'importe quel fournisseur CMP compatible. Pour créer un fournisseur KMS direct, spécifiez un [AWS KMS key](#). Cet exemple utilise le Amazon Resource Name (ARN) du AWS KMS key, mais vous pouvez utiliser n'importe quel identifiant de clé valide.

```
kms_key_id='arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'  
kms_cmp = AwsKmsCryptographicMaterialsProvider(key_id=kms_key_id)
```

Étape 3 : Utiliser la classe TableInfo d'assistance

Pour obtenir des informations sur la table à partir de DynamoDB, créez une instance de [TableInfo](#) la classe d'assistance. Lorsque vous travaillez directement avec le chiffreur d'élément, vous devez créer une instance TableInfo et appeler ses méthodes. Les [classes d'annotation clientes](#) le font pour vous.

La `refresh_indexed_attributes` méthode TableInfo utilise l'opération [DescribeTable](#) DynamoDB pour obtenir des informations précises et en temps réel sur la table. Elles incluent la clé primaire et ses index secondaires locaux et globaux. L'appelant doit avoir l'autorisation d'appeler DescribeTable.

```
table_info = TableInfo(name=table_name)  
table_info.refresh_indexed_attributes(table.meta.client)
```

Étape 4 : Création du contexte de chiffrement DynamoDB

Le contexte de [chiffrement DynamoDB](#) contient des informations sur la structure de la table et sur la manière dont elle est chiffrée et signée. Cet exemple crée un contexte de chiffrement DynamoDB de manière explicite, car il interagit avec le chiffreur d'éléments. Les [classes d'assistance client](#) créent le contexte de chiffrement DynamoDB pour vous.

Pour obtenir la clé de partition et la clé de tri, vous pouvez utiliser les propriétés de la classe d'[TableInfo](#) assistance.

```
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)
)
```

Étape 5 : Créer l'objet actions d'attribut

Les [actions d'attribut](#) informent le chiffreur d'élément des actions à exécuter sur chaque attribut de l'élément. L'objet `AttributeActions` de l'exemple chiffre et signe tous les éléments à l'exception des attributs de clé primaire, qui sont signés, mais pas chiffrés, et de l'attribut `test`, qui est ignoré.

Lorsque vous interagissez directement avec le chiffreur d'élément et que votre action par défaut est `ENCRYPT_AND_SIGN`, vous devez spécifier une autre action pour la clé primaire. Vous pouvez utiliser la méthode `set_index_keys`, qui utilise `SIGN_ONLY` pour la clé primaire, ou `DO_NOTHING` s'il s'agit de l'action par défaut.

Pour spécifier la clé primaire, cet exemple utilise les clés d'index de l'[TableInfo](#) objet, qui sont renseignées par un appel à DynamoDB. Cette technique est plus sûre que le codage en dur des noms de clé primaire.

```
actions = AttributeActions(
    default_action=CryptoAction.ENCRYPT_AND_SIGN,
    attribute_actions={'test': CryptoAction.DO_NOTHING}
)
```

```
actions.set_index_keys(*table_info.protected_index_keys())
```

Étape 6 : Créer la configuration pour l'élément

Pour configurer le client de chiffrement DynamoDB, utilisez les objets que vous venez de créer dans [CryptoConfig](#) une configuration pour l'élément de table. Les classes d'assistance client créent le CryptoConfig pour vous.

```
crypto_config = CryptoConfig(
    materials_provider=kms_cmp,
    encryption_context=encryption_context,
    attribute_actions=actions
)
```

Étape 7 : Chiffrer l'élément

Cette étape chiffre et signe l'élément, mais ne le place pas dans la table DynamoDB.

Lorsque vous utilisez une classe d'assistance client, vos éléments sont chiffrés et signés de manière transparente, puis ajoutés à votre table DynamoDB lorsque vous appelez la méthode de la `put_item` classe d'assistance. Lorsque vous utilisez le chiffreur d'élément directement, les actions de chiffrage et de placement sont indépendantes.

Tout d'abord, créez un élément en texte brut.

```
plaintext_item = {
    'partition_attribute': 'value1',
    'sort_key': 55,
    'example': 'data',
    'numbers': 99,
    'binary': Binary(b'\x00\x01\x02'),
    'test': 'test-value'
}
```

Puis, chiffrez-le et signez-le. La méthode `encrypt_python_item` requiert l'objet de configuration `CryptoConfig`.

```
encrypted_item = encrypt_python_item(plaintext_item, crypto_config)
```

Étape 8 : Placer l'élément dans la table

Cette étape place l'élément chiffré et signé dans la table DynamoDB.

```
table.put_item(Item=encrypted_item)
```

Pour afficher l'élément chiffré, appelez la méthode `get_item` sur l'objet original `table`, et non sur l'objet `encrypted_table`. Elle obtient l'élément de la table DynamoDB sans le vérifier et le déchiffrer.

```
encrypted_item = table.get_item(Key=partition_key)['Item']
```

L'illustration suivante présente une partie d'un exemple d'élément de table chiffré et signé.

Les valeurs des attributs chiffrés sont des données binaires. Les noms et valeurs des attributs de clé primaire (`partition_attribute` et `sort_attribute`) et l'attribut `test` demeurent en texte brut. La sortie affiche aussi l'attribut qui contient la signature (`*amzn-ddb-map-sig*`) et l'[attribut de description des matériaux](#) (`*amzn-ddb-map-desc*`).


```
{
  '*amzn-ddb-map-desc*': Binary(b'\x00\x00\x00\x00\x00\x00\x00\x10amzn-ddb-env-alg\x00\x00\x00\xe0AQEBAAHhA84wnXjEJdBbBBYlRUFcZZK2j7xwh6UyLoL28nQ+0FAAAAH4wfAYJKoZIhvcNAQcGoG8wbQIBADBoBgkqhkiG9w0BBwEwHgYJYIZIAWUDBAEuMBEEDPeFBydmoJDizYl0R0C4M7wAK6E1/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\xfb\x1d6a\xc5\xd5\x1aZ\xed\xd6\xce\xe9X\xfbT\xcb'numbers': Binary(b'\xd5\xa0\d\xcc\x85\xf5\x1e\xb9-f!\xb9\xb8\x8a\x1aT\xbaq\xfb'partition_attribute': 'value1',
  'sort_attribute': 55,
  'test': 'test-value'
}
```

Modification de votre modèle de données

Note

Notre bibliothèque de chiffrement côté client a été [renommée AWS Database Encryption SDK](#). La rubrique suivante fournit des informations sur les versions 1. x —2. x du client de chiffrement DynamoDB pour Java et versions 1. x —3. x du client de chiffrement DynamoDB pour Python. Pour plus d'informations, consultez la section [SDK AWS de chiffrement de base de données pour la prise en charge des versions DynamoDB](#).

Chaque fois que vous chiffrez ou déchiffrez un élément, vous devez fournir des [actions attributaires](#) qui indiquent au client de chiffrement DynamoDB les attributs à chiffrer et à signer, les attributs à signer (mais pas à chiffrer) et ceux à ignorer. Les actions d'attribut ne sont pas enregistrées dans l'élément chiffré et le client de chiffrement DynamoDB ne met pas automatiquement à jour vos actions d'attribut.

 Important

Le client de chiffrement DynamoDB ne prend pas en charge le chiffrement des données de table DynamoDB existantes non chiffrées.

Chaque fois que vous modifiez votre modèle de données, c'est-à-dire lorsque vous ajoutez ou supprimez des attributs de vos éléments de table, vous risquez une erreur. Si les actions d'attribut que vous spécifiez ne rendent pas compte de tous les attributs de l'élément, l'élément peut ne pas être chiffré et signé comme vous l'escomptiez. Surtout, si les actions d'attribut que vous fournissez lors du déchiffrement d'un élément diffèrent des actions d'attribut que vous avez fournies lors du chiffrement de l'élément, la vérification de la signature peut échouer.

Par exemple, si les actions d'attribut utilisées pour chiffrer l'élément lui disent de signer l'attribut `test`, la signature de l'élément comportera l'attribut `test`. Cependant, si les actions d'attribut utilisées pour déchiffrer l'élément ne tiennent pas compte de l'attribut `test`, la vérification échouera parce que le client essaiera de vérifier une signature qui n'inclut pas l'attribut `test`.

Cela pose un problème particulier lorsque plusieurs applications lisent et écrivent les mêmes éléments DynamoDB, car le client de chiffrement DynamoDB doit calculer la même signature pour les éléments de toutes les applications. C'est également un problème pour toute application distribuée car les modifications dans les actions d'attribut doivent se propager à tous les hôtes. Même si un hôte accède à vos tables DynamoDB dans le cadre d'un seul processus, la mise en place d'un processus basé sur les meilleures pratiques permettra d'éviter les erreurs si le projet devient plus complexe.

Pour éviter les erreurs de validation de signature qui vous empêchent de lire les éléments de votre tableau, suivez les instructions suivantes.

- [Ajout d'un attribut](#) : si le nouvel attribut modifie vos actions d'attribut, déployez entièrement la modification d'action d'attribut avant d'inclure le nouvel attribut dans un élément.
- [Suppression d'un attribut](#) : si vous arrêtez d'utiliser un attribut dans vos articles, ne modifiez pas vos actions d'attribut.

- Modification de l'action : une fois que vous avez utilisé une configuration d'actions attributaires pour chiffrer les éléments de votre tableau, vous ne pouvez pas modifier en toute sécurité l'action par défaut ou l'action d'un attribut existant sans rechiffrer chaque élément de votre tableau.

Les erreurs de validation de signature peuvent être extrêmement difficiles à résoudre, de sorte que la meilleure approche consiste à les prévenir.

Rubriques

- [Ajout d'un attribut](#)
- [Suppression d'un attribut](#)

Ajout d'un attribut

Lorsque vous ajouterez un nouvel attribut à des éléments de table, vous devrez peut-être modifier vos actions attributaires. Pour éviter les erreurs de validation de signature, nous vous recommandons d'implémenter cette modification en deux étapes. Vérifiez que la première étape est terminée avant de commencer la deuxième étape.

1. Modifiez les actions d'attribut dans toutes les applications qui lisent ou écrivent dans la table. Déployez ces modifications et confirmez que la mise à jour a été propagée à tous les hôtes de destination.
2. Écrire des valeurs dans le nouvel attribut dans vos éléments de table.

Cette approche en deux étapes garantit que toutes les applications et les hôtes ont les mêmes actions d'attribut et calculera la même signature, avant toute rencontre avec le nouvel attribut. Ceci est important même lorsque l'action de l'attribut est Ne rien faire (ne pas chiffrer ou signer), car la valeur par défaut de certains chiffreurs est de chiffrer et de signer.

Les exemples suivants montrent le code de la première étape de ce processus. Ils ajoutent un nouvel attribut d'élément `link`, qui stocke un lien vers un autre élément de table. Étant donné que ce lien doit rester en texte brut, l'exemple lui attribue l'action `sign-only`. Après avoir entièrement déployé cette modification, puis vérifié que toutes les applications et tous les hôtes possèdent les nouvelles actions attributaires, vous pouvez commencer à utiliser l'attribut `link` dans vos éléments de table.

Java DynamoDB Mapper

Lors de l'utilisation de DynamoDB Mapper et AttributeEncryptor, par défaut, tous les attributs sont chiffrés et signés à l'exception des clés primaires, qui sont signées mais pas chiffrées. Pour spécifier une action de signature uniquement, utilisez l'annotation `@DoNotEncrypt`.

Cet exemple utilise l'annotation `@DoNotEncrypt` du nouvel attribut `link`.

```
@DynamoDBTable(tableName = "ExampleTable")
public static final class DataPoJo {
    private String partitionAttribute;
    private int sortAttribute;
    private String link;

    @DynamoDBHashKey(attributeName = "partition_attribute")
    public String getPartitionAttribute() {
        return partitionAttribute;
    }

    public void setPartitionAttribute(String partitionAttribute) {
        this.partitionAttribute = partitionAttribute;
    }

    @DynamoDBRangeKey(attributeName = "sort_attribute")
    public int getSortAttribute() {
        return sortAttribute;
    }

    public void setSortAttribute(int sortAttribute) {
        this.sortAttribute = sortAttribute;
    }

    @DynamoDBAttribute(attributeName = "link")
    @DoNotEncrypt
    public String getLink() {
        return link;
    }

    public void setLink(String link) {
        this.link = link;
    }
}
```

```
@Override
public String toString() {
    return "DataPoJo [partitionAttribute=" + partitionAttribute + ",
        sortAttribute=" + sortAttribute + ",
        link=" + link + "];"
}
}
```

Java DynamoDB encryptor

Dans le chiffreur DynamoDB de niveau inférieur, vous devez définir des actions pour chaque attribut. Cet exemple utilise une instruction `switch` où la valeur par défaut est `encryptAndSign` et des exceptions sont spécifiées pour la clé de partition, la clé de tri et le nouvel attribut `link`. Dans cet exemple, si le code d'attribut de lien n'était pas entièrement déployé avant son utilisation, l'attribut de lien serait chiffré et signé par certaines applications, mais seulement signé par d'autres.

```
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

Dans le client de chiffrement DynamoDB pour Python, vous pouvez définir une action par défaut pour tous les attributs, puis spécifier des exceptions.

Si vous utilisez une [classe d'annotations clientes](#), vous n'avez pas besoin de spécifier une action d'attribut pour les attributs de clé primaire. Les classes d'annotations clients vous empêchent de chiffrer votre clé primaire. Toutefois, si vous n'utilisez pas de classes d'annotations clientes, vous devez définir l'action `SIGN_ONLY` sur votre clé de partition et la clé de tri. Si vous chiffrez accidentellement votre partition ou votre clé de tri, vous ne pourrez pas récupérer vos données sans une analyse complète de la table.

Cet exemple spécifie une exception pour le nouvel attribut `link`, qui obtient l'action `SIGN_ONLY`.

```
actions = AttributeActions(  
    default_action=CryptoAction.ENCRYPT_AND_SIGN,  
    attribute_actions={  
        'example': CryptoAction.DO_NOTHING,  
        'link': CryptoAction.SIGN_ONLY  
    }  
)
```

Suppression d'un attribut

Si vous n'avez plus besoin d'un attribut dans les éléments chiffrés avec le client de chiffrement DynamoDB, vous pouvez arrêter de l'utiliser. Toutefois, ne supprimez pas ou ne modifiez pas l'action de cet attribut. Si vous le faites, puis que vous rencontrez un élément avec cet attribut, la signature calculée pour l'élément ne correspondra pas à la signature d'origine et la validation de la signature échouera.

Bien que vous soyez tenté de supprimer toutes les traces de l'attribut de votre code, ajoutez un commentaire indiquant que l'élément n'est plus utilisé au lieu de le supprimer. Même si vous effectuez une analyse complète de table pour supprimer toutes les instances de l'attribut, un élément chiffré avec cet attribut peut être mis en cache ou en cours de traitement quelque part dans votre configuration.

Résolution des problèmes liés à votre application cliente de chiffrement DynamoDB

Note

Notre bibliothèque de chiffrement côté client a été [renommée AWS Database Encryption SDK](#). La rubrique suivante fournit des informations sur les versions 1. x —2. x du client de

chiffrement DynamoDB pour Java et versions 1. x —3. x du client de chiffrement DynamoDB pour Python. Pour plus d'informations, consultez la section [SDK AWS de chiffrement de base de données pour la prise en charge des versions DynamoDB](#).

Cette section décrit les problèmes que vous pouvez rencontrer lors de l'utilisation du client de chiffrement DynamoDB et propose des suggestions pour les résoudre.

Pour fournir des commentaires sur le client de chiffrement DynamoDB, signalez un problème dans [aws-dynamodb-encryption-java](#) le référentiel or. [aws-dynamodb-encryption-python](#) GitHub

Pour émettre des commentaires sur cette documentation, utilisez le lien des commentaires sur n'importe quelle page.

Rubriques

- [Accès refusé](#)
- [Échec de la vérification de la signature](#)
- [Problèmes liés aux anciennes versions des tables globales](#)
- [Mauvaise performance du fournisseur le plus récent](#)

Accès refusé

Problème : votre application se voit refuser l'accès à une ressource dont elle a besoin.

Suggestion : en savoir plus sur les autorisations requises et les ajouter au contexte de sécurité dans lequel votre application s'exécute.

Détails

Pour exécuter une application qui utilise la bibliothèque d'un client de chiffrement DynamoDB, l'appelant doit être autorisé à utiliser ses composants. Sinon, l'accès aux éléments requis lui est refusé.

- Le client de chiffrement DynamoDB ne nécessite pas de compte Amazon Web Services AWS() et ne dépend d'aucun service. AWS Toutefois, si votre application l'utilise AWS, vous avez besoin [d'un Compte AWS](#) et d'[utilisateurs autorisés](#) à utiliser le compte.
- Le client de chiffrement DynamoDB n'a pas besoin d'Amazon DynamoDB. Toutefois, si l'application qui utilise le client crée des tables DynamoDB, place des éléments dans une table ou extrait des

éléments d'une table, l'appelant doit être autorisé à utiliser les opérations DynamoDB requises dans votre. Compte AWS Pour plus de détails, consultez les [rubriques relatives au contrôle d'accès](#) dans le guide du développeur Amazon DynamoDB.

- Si votre application utilise une [classe d'assistance client](#) dans le client de chiffrement DynamoDB pour Python, l'appelant doit être autorisé à appeler l'opération DynamoDB. [DescribeTable](#)
- Le client de chiffrement DynamoDB n'a pas AWS Key Management Service besoin de ().AWS KMS Toutefois, si votre application utilise un [fournisseur de matériel KMS direct](#), ou si elle utilise un [fournisseur le plus récent](#) avec un magasin de fournisseurs qui l'utilise AWS KMS, l'appelant doit être autorisé à utiliser les opérations AWS KMS [GenerateDataKey](#) et à [déchiffrer](#).

Échec de la vérification de la signature

Problème : un élément ne peut pas être déchiffré en raison de l'échec de la vérification de la signature. L'élément peut aussi ne pas avoir été chiffré et signé comme vous l'escomptez.

Suggestion : vérifiez que les actions d'attribut que vous fournissez représentent tous les attributs de l'élément. Lors du déchiffrement d'un élément, veillez à fournir les actions d'attribut qui correspondent aux actions utilisées pour chiffrer l'élément.

Détails

Les [actions d'attribut](#) que vous fournissez indiquent au client de chiffrement DynamoDB les attributs à chiffrer et à signer, les attributs à signer (mais pas à chiffrer) et ceux à ignorer.

Si les actions d'attribut que vous spécifiez ne rendent pas compte de tous les attributs de l'élément, l'élément peut ne pas être chiffré et signé comme vous l'escomptiez. Si les actions d'attribut que vous fournissez lors du déchiffrement d'un élément diffèrent des actions d'attribut que vous avez fournies lors du chiffrement de l'élément, la vérification de la signature peut échouer. Il s'agit d'un problème particulier pour les applications distribuées dans lesquelles les nouvelles actions d'attribut peuvent ne pas avoir été propagées sur tous les hôtes.

Les erreurs de validation de signature sont difficiles à résoudre. Pour vous aider à les prévenir, prenez des précautions supplémentaires lorsque vous modifiez votre modèle de données. Pour en savoir plus, consultez [Modification de votre modèle de données](#).

Problèmes liés aux anciennes versions des tables globales

Problème : les éléments d'une ancienne version de la table globale Amazon DynamoDB ne peuvent pas être déchiffrés car la vérification des signatures échoue.

Suggestion : définissez des actions d'attribut afin que les champs de réplication réservés ne soient ni chiffrés ni signés.

Détails

Vous pouvez utiliser le client de chiffrement DynamoDB avec les tables globales [DynamoDB](#). Nous vous recommandons d'utiliser des tables globales avec une [clé KMS multirégionale et de répliquer la clé KMS](#) dans tous les Régions AWS endroits où la table globale est répliquée.

À partir de la [version 2019.11.21](#) des tables globales, vous pouvez utiliser des tables globales avec le client de chiffrement DynamoDB sans configuration particulière. Toutefois, si vous utilisez des tables globales [version 2017.11.29](#), vous devez vous assurer que les champs de réplication réservés ne sont ni chiffrés ni signés.

[Si vous utilisez la version 2017.11.29 des tables globales, vous devez définir les actions d'attribut pour les attributs suivants DO_NOTHING en Java ou en @DoNotTouch Python.](#)

- `aws:rep:deleting`
- `aws:rep:updatetime`
- `aws:rep:updateregion`

Si vous utilisez une autre version des tables globales, aucune action n'est requise.

Mauvaise performance du fournisseur le plus récent

Problème : votre application est moins réactive, en particulier après la mise à jour vers une version plus récente du client de chiffrement DynamoDB.

Suggestion : ajustez la time-to-live valeur et la taille du cache.

Détails

Le fournisseur le plus récent est conçu pour améliorer les performances des applications qui utilisent le client de chiffrement DynamoDB en autorisant une réutilisation limitée du matériel cryptographique. Lorsque vous configurez le fournisseur le plus récent pour votre application, vous devez trouver un équilibre entre l'amélioration des performances et les problèmes de sécurité liés à la mise en cache et à la réutilisation.

Dans les nouvelles versions du client de chiffrement DynamoDB, la valeur (TTL) détermine time-to-live la durée pendant laquelle les fournisseurs de matériel cryptographique mis en cache () peuvent

être utilisés. CMPs Le TTL détermine également la fréquence à laquelle le fournisseur le plus récent vérifie la présence d'une nouvelle version du CMP.

Si votre TTL est trop long, votre application peut enfreindre vos règles commerciales ou vos normes de sécurité. Si votre TTL est trop court, des appels fréquents au magasin du fournisseur peuvent entraîner le ralentissement des demandes provenant de votre application et d'autres applications partageant votre compte de service. Pour résoudre ce problème, ajustez le TTL et la taille du cache à une valeur correspondant à vos objectifs de latence et de disponibilité et conforme à vos normes de sécurité. Pour en savoir plus, consultez [Définition d'une time-to-live valeur](#).

Changement du nom du client de chiffrement Amazon DynamoDB

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Le 9 juin 2023, notre bibliothèque de chiffrement côté client a été renommée AWS Database Encryption SDK. Le SDK AWS de chiffrement de base de données est compatible avec Amazon DynamoDB. Il peut déchiffrer et lire les éléments chiffrés par l'ancien client de chiffrement DynamoDB. Pour plus d'informations sur les anciennes versions du client de chiffrement DynamoDB, consultez [AWS SDK de chiffrement de base de données pour la prise en charge des versions DynamoDB](#)

Le SDK AWS de chiffrement de base de données fournit la version 3. x de la bibliothèque de chiffrement côté client Java pour DynamoDB, qui constitue une réécriture majeure du client de chiffrement DynamoDB pour Java. Il inclut de nombreuses mises à jour, telles qu'un nouveau format de données structuré, une prise en charge améliorée de la mutualisation, des modifications de schéma fluides et la prise en charge du chiffrement consultable.

Pour en savoir plus sur les nouvelles fonctionnalités introduites avec le SDK AWS de chiffrement de base de données, consultez les rubriques suivantes.

[Chiffrement consultable](#)

Vous pouvez concevoir des bases de données capables de rechercher des enregistrements chiffrés sans déchiffrer l'intégralité de la base de données. En fonction de votre modèle de menace et de vos exigences en matière de requêtes, vous pouvez utiliser le chiffrement consultable pour effectuer des recherches de correspondance exacte ou des requêtes complexes plus personnalisées sur vos enregistrements chiffrés.

[Porte-clés](#)

Le SDK AWS Database Encryption utilise des trousseaux de clés pour chiffrer les [enveloppes](#). Les porte-clés génèrent, chiffrent et déchiffrant les clés de données qui protègent vos dossiers. Le SDK AWS Database Encryption prend en charge les AWS KMS jeux de clés utilisant le

chiffrement symétrique ou RSA asymétrique [AWS KMS keys](#) pour protéger vos clés de données, ainsi que les porte-clés AWS KMS hiérarchiques qui vous permettent de protéger vos documents cryptographiques sous une clé KMS de chiffrement symétrique sans avoir à appeler AWS KMS chaque fois que vous chiffrez ou déchiffrez un enregistrement. Vous pouvez également spécifier votre propre matériau clé avec les porte-clés Raw AES et RSA Raw.

[Changements de schéma simplifiés](#)

Lorsque vous configurez le SDK de chiffrement de AWS base de données, vous fournissez des [actions cryptographiques](#) qui indiquent au client les champs à chiffrer et à signer, les champs à signer (mais pas à chiffrer) et ceux à ignorer. Après avoir utilisé le SDK AWS de chiffrement de base de données pour protéger vos enregistrements, vous pouvez toujours apporter des modifications à votre modèle de données. Vous pouvez mettre à jour vos actions cryptographiques, telles que l'ajout ou la suppression de champs chiffrés, en un seul déploiement.

[Configuration des tables DynamoDB existantes pour le chiffrement côté client](#)

Les anciennes versions du client de chiffrement DynamoDB ont été conçues pour être implémentées dans de nouvelles tables non remplies. Avec le SDK AWS Database Encryption pour DynamoDB, vous pouvez migrer vos tables Amazon DynamoDB existantes vers la version 3.x de la bibliothèque de chiffrement côté client Java pour DynamoDB.

Référence

Notre bibliothèque de chiffrement côté client a été renommée SDK de chiffrement de AWS base de données. Ce guide du développeur fournit toujours des informations sur le client de [chiffrement DynamoDB](#).

Les rubriques suivantes fournissent des informations techniques sur le SDK de chiffrement AWS de base de données.

Format de description du matériau

La [description du matériau](#) sert d'en-tête à un enregistrement crypté. Lorsque vous chiffrez et signez des champs avec le SDK de chiffrement de AWS base de données, le crypteur enregistre la description du matériel au fur et à mesure qu'il assemble le matériel cryptographique et stocke la description du matériel dans un nouveau champ (`aws_dbe_head`) qu'il ajoute à votre enregistrement. La description matérielle est une structure de données formatée portable qui contient la clé de données cryptée et des informations sur la manière dont l'enregistrement a été crypté et signé. Le tableau suivant décrit les valeurs qui constituent la description du matériau. Les octets sont ajoutés dans l'ordre indiqué.

Value	Longueur en octets
Version	1
Signatures Enabled	1
Record ID	32
Encrypt Legend	Variable
Encryption Context Length	2
???	Variable
Encrypted Data Key Count	1
Encrypted Data Keys	Variable

Value	Longueur en octets
Record Commitment	1

Version

Version du format de ce `aws_dbe_head` champ.

Signatures activées

Indique si les signatures numériques ECDSA sont activées pour cet enregistrement.

Valeur d'octet	Signification
0x01	Signatures numériques ECDSA activées (par défaut)
0x00	Signatures numériques ECDSA désactivées

Identifiant de l'enregistrement

Une valeur de 256 bits générée aléatoirement qui identifie l'enregistrement. L'identifiant de l'enregistrement :

- Identifie de manière unique l'enregistrement crypté.
- Lie la description du matériau à l'enregistrement crypté.

Chiffrer la légende

Description sérialisée des champs authentifiés qui ont été chiffrés. La légende de chiffrement est utilisée pour déterminer les champs que la méthode de déchiffrement doit tenter de déchiffrer.

Valeur d'octet	Signification
0x65	ENCRYPT_AND_SIGN
0x73	SIGN_ONLY

Le Encrypt Legend est sérialisé comme suit :

1. Lexicographiquement par la séquence d'octets qui représente leur chemin canonique.
2. Pour chaque champ, dans l'ordre, ajoutez l'une des valeurs d'octets spécifiées ci-dessus pour indiquer si ce champ doit être crypté.

Longueur du contexte de chiffrement

La longueur du contexte de chiffrement. Il s'agit d'une valeur de 2 octets interprétée comme un entier 16 bits non signé. La longueur maximale est de 65 535 octets.

Contexte de chiffrement

Ensemble de paires nom-valeur contenant des données authentifiées supplémentaires arbitraires et non secrètes.

Lorsque les [signatures numériques ECDSA](#) sont activées, le contexte de chiffrement contient la paire clé-valeur. {"aws-crypto-footer-ecdsa-key": Qtxt} Qtxt représente le point de la courbe elliptique Q compressé conformément à [SEC 1 version 2.0 puis codé](#) en base64.

Nombre de clés de données cryptées

Nombre de clés de données chiffrées. Il s'agit d'une valeur de 1 octet interprétée comme un entier non signé de 8 bits qui indique le nombre de clés de données chiffrées. Le nombre maximum de clés de données chiffrées dans chaque enregistrement est de 255.

Clés de données cryptées

Séquence de clés de données chiffrées. La longueur de la séquence est déterminée par le nombre de clés de données chiffrées et la longueur de chacune. La séquence contient au moins une clé de données chiffrée.

Le tableau suivant décrit les champs qui composent chaque clé de données chiffrée. Les octets sont ajoutés dans l'ordre indiqué.

Structure de la clé de données chiffrée

Champ	Longueur en octets
Key Provider ID Length	2
Key Provider ID	Variable. Est égal à la valeur spécifiée dans les 2 octets précédents (Longueur de l'ID du fournisseur de clés).

Champ	Longueur en octets
Key Provider Information Length	2
Key Provider Information	Variable. Est égal à la valeur spécifiée dans les 2 octets précédents (Longueur de l'information du fournisseur de clés).
Encrypted Data Key Length	2
Encrypted Data Key	Variable. Est égal à la valeur spécifiée dans les 2 octets précédents (Longueur de la clé de données chiffrée).

Longueur de l'identifiant du fournisseur clé

Longueur de l'identifiant du fournisseur de clés. Il s'agit d'une valeur de 2 octets interprétée comme un entier 16 bits non signé qui spécifie le nombre d'octets qui contiennent l'ID du fournisseur des clés.

Identifiant du fournisseur clé

Identifiant du fournisseur de clés. Il est utilisé pour indiquer le fournisseur de la clé de données chiffrée et doit être extensible.

Longueur des informations sur les fournisseurs clés

Longueur de l'information du fournisseur de clés. Il s'agit d'une valeur de 2 octets interprétée comme un entier 16 bits non signé qui spécifie le nombre d'octets qui contiennent l'information du fournisseur des clés.

Informations clés sur les fournisseurs

Information sur le fournisseur de clés. Il est déterminé par le fournisseur de clés.

Lorsque vous utilisez un AWS KMS trousseau de clés, cette valeur contient le nom de ressource Amazon (ARN) du AWS KMS key.

Longueur de la clé de données cryptée

Longueur de la clé de données chiffrée. Il s'agit d'une valeur de 2 octets interprétée comme un entier 16 bits non signé qui spécifie le nombre d'octets qui contiennent la clé de données chiffrée.

Clé de données cryptée

Clé de données chiffrée. Il s'agit de la clé de données cryptée par le fournisseur de clés.

Engagement record

Un hachage distinct de 256 bits basé sur le hachage (HMAC) calculé sur tous les octets de description matérielle précédents à l'aide de la clé de validation.

AWS KMS Détails techniques du porte-clés hiérarchique

Le trousseau de [clés AWS KMS hiérarchique](#) utilise une clé de données unique pour chiffrer chaque champ et chiffre chaque clé de données avec une clé d'encapsulation unique dérivée d'une clé de branche active. Il utilise une [dérivation de clé](#) en mode compteur avec une fonction pseudo-aléatoire avec HMAC SHA-256 pour dériver la clé d'encapsulation de 32 octets avec les entrées suivantes.

- Un sel aléatoire de 16 octets
- La clé de branche active
- La valeur [codée en UTF-8](#) pour l'identifiant du fournisseur de clés « » aws-kms-hierarchy

Le trousseau de clés hiérarchique utilise la clé d'encapsulation dérivée pour chiffrer une copie de la clé de données en texte brut à l'aide du protocole AES-GCM-256 avec une balise d'authentification de 16 octets et les entrées suivantes.

- La clé d'encapsulation dérivée est utilisée comme clé de chiffrement AES-GCM
- La clé de données est utilisée comme message AES-GCM
- Un vecteur d'initialisation aléatoire (IV) de 12 octets est utilisé comme AES-GCM IV
- Données authentifiées supplémentaires (AAD) contenant les valeurs sérialisées suivantes.

Value	Longueur en octets	Interprété comme
"aws-kms-hierarchy"	17	Être encodé UTF-8
L'identifiant de la clé de branche	Variable	Être encodé UTF-8

Value	Longueur en octets	Interprété comme
La version de la clé de branche	16	Être encodé UTF-8
Contexte de chiffrement	Variable	Paires de valeurs clés codées en UTF-8

Historique du document relatif au guide du développeur du SDK AWS de chiffrement des bases de données

Le tableau suivant décrit les modifications importantes apportées à cette documentation. En plus de ces principales modifications, nous mettons fréquemment à jour la documentation pour améliorer les descriptions et les exemples, et pour répondre aux commentaires que vous nous envoyez. Pour recevoir une notification concernant des modifications importantes, abonnez-vous au flux RSS.

Modification	Description	Date
Nouvelle fonction	Ajout de documentation pour le porte-clés AWS KMS ECDH et le porte-clés ECDH brut .	17 juin 2024
Version de disponibilité générale (GA)	Présentation de la prise en charge de la bibliothèque de chiffrement côté client .NET pour DynamoDB.	17 janvier 2024
Version de disponibilité générale (GA)	Documentation mise à jour pour la version GA de la version 3. x de la bibliothèque de chiffrement côté client Java pour DynamoDB.	24 juillet 2023

Warning

Les clés de branche créées lors de la version préliminaire pour les développeurs ne sont plus prises en charge.

Changement de nom du client de chiffrement DynamoDB	La bibliothèque de chiffrement côté client est renommée AWS Database Encryption SDK.	9 juin 2023
Version préliminaire	Documentation ajoutée et mise à jour pour la version 3. x de la bibliothèque de chiffrement côté client Java pour DynamoDB, qui inclut un nouveau format de données structurées, une prise en charge améliorée de la mutualisation, des modifications de schéma fluides et une prise en charge du chiffrement consultable.	9 juin 2023
Modification de la documentation	Remplacez le AWS Key Management Service terme clé principale du client (CMK) par AWS KMS key clé KMS.	30 août 2021
Nouvelle fonction	Ajout du support pour AWS Key Management Service (AWS KMS) les clés multirégionales. Les clés multirégionales sont des AWS KMS clés différentes Régions AWS qui peuvent être utilisées de manière interchangeable car elles ont le même identifiant de clé et le même matériau de clé.	8 juin 2021
Nouvel exemple	Ajout d'un exemple d'utilisation de Dynamo DBMapper en Java.	6 septembre 2018

[Support de Python](#)

Ajout de la prise en charge de Python, en plus de Java. 2 mai 2018

[Première version](#)

Première version de la présente documentation. 2 mai 2018

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.