Guide du développeur

AWS Encryption SDK

AWS Encryption SDK: Guide du développeur

Table of Contents

Qu'est-ce que AWS Encryption SDK ?

Le kit AWS Encryption SDK est une bibliothèque de chiffrement client conçue pour faciliter pour tous le chiffrement et le déchiffrement de données à l'aide des normes du secteur et des bonnes pratiques. Il vous permet de vous concentrer sur les fonctionnalités intrinsèques de votre application, plutôt que sur la meilleure façon de chiffrer et de déchiffrer vos données. Le kit AWS Encryption SDK est fourni gratuitement dans le cadre de la licence Apache 2.0.

Le kit AWS Encryption SDK répond aux questions suivantes :

- Quel algorithme de chiffrement dois-je utiliser ?
- Comment, ou dans lequel mode, devrais-je utiliser cet algorithme ?
- Comment puis-je générer la clé de chiffrement ?
- Comment puis-je protéger la clé de chiffrement, et où dois-je la stocker ?
- Comment puis-je rendre mes données chiffrées portables ?
- Comment puis-je m'assurer que le destinataire prévu peut lire mes données chiffrées ?
- Comment puis-je m'assurer que mes données chiffrées ne sont pas modifiées entre leur écriture et leur lecture ?
- Comment utiliser les clés de données AWS KMS renvoyées ?

Avec leAWS Encryption SDK, vous définissez un fournisseur de clés principales (Java et Python) ou un jeu de clés (C, C#/.NET et JavaScript) qui détermine les clés d'encapsulation que vous utilisez pour protéger vos données. Ensuite, vous chiffrez et déchiffrez vos données à l'aide de méthodes simples fournies par le kit AWS Encryption SDK. Le kit AWS Encryption SDK se charge du reste.

Sans le kit AWS Encryption SDK, il est possible que vous consacriez plus d'efforts au développement d'une solution de chiffrement qu'aux fonctionnalités intrinsèques de votre application. Le kit AWS Encryption SDK répond à ces questions en fournissant les éléments suivants.

Une implémentation par défaut qui respecte les bonnes pratiques de chiffrement

> Par défaut, le kit AWS Encryption SDK génère une clé de données unique pour chaque objet de données qu'il chiffre. Cette opération respecte la bonne pratique de chiffrement qui consiste à utiliser des clés de données uniques pour chaque opération de chiffrement.

Le kit AWS Encryption SDK chiffre vos données à l'aide d'un algorithme de clé symétrique sécurisé et authentifié. Pour en savoir plus, consultez the section called “Suites d'algorithmes prises en charge”.

Un cadre pour protéger les clés de données à l'aide de clés d'encapsulation

AWS Encryption SDKprotège les clés de données qui chiffrent vos données en les chiffrant sous une ou plusieurs clés d'encapsulation. En fournissant un cadre permettant de chiffrer les clés de données à l'AWS Encryption SDKaide de plusieurs clés d'encapsulation, vous contribuez à rendre vos données chiffrées portables.

Par exemple, chiffrez les données sous une AWS KMS key clé d'entrée AWS KMS et une clé à partir de votre HSM local. Vous pouvez utiliser l'une ou l'autre des clés d'encapsulation pour déchiffrer les données, au cas où l'une d'entre elles ne serait pas disponible ou si l'appelant n'est pas autorisé à utiliser les deux clés.

Un message formaté qui stocke les clés de données chiffrées avec les données chiffrées

Le kit AWS Encryption SDK stocke les données chiffrées et la clé de données chiffrée dans un message chiffré qui utilise un format de données défini. Cela signifie que vous n'avez pas besoin d'effectuer le suivi ou de protéger les clés de données utilisées pour chiffrer vos données, car le kit AWS Encryption SDK s'en charge.

Certaines implémentations linguistiques du AWS Encryption SDK nécessitent un AWS SDK, mais AWS Encryption SDK cela n'en nécessite pas Compte AWS et cela ne dépend d'aucun AWS service. Vous Compte AWS n'en avez besoin que si vous choisissez de l'utiliser AWS KMS keyspour protéger vos données.

Développé dans des référentiels open source

AWS Encryption SDKIl 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 l'implémentation de votre langage.

- Kit SDK de chiffrement AWS pour C — aws-encryption-sdk-c
- AWS Encryption SDKpour .NET : aws-encryption-sdk-netrépertoire du `aws-encryption-sdk-dafny` dépôt.
- AWSCLI de chiffrement — aws-encryption-sdk-cli
- Kit SDK de chiffrement AWS pour Java — aws-encryption-sdk-java

- Kit SDK de chiffrement AWS pour JavaScript — aws-encryption-sdk-javascript
- Kit SDK de chiffrement AWS pour Python — aws-encryption-sdk-python

Compatibilité avec les bibliothèques et services de chiffrement

AWS Encryption SDKIl est pris en charge dans plusieurs langages de programmation. Toutes les implémentations de langage 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. En outre, lors du chiffrement et du déchiffrement, vous devez utiliser des porte-clés compatibles, ou des clés principales et des fournisseurs de clés principales. Pour plus de détails, consultez the section called “Compatibilité du porte-clés”.

Cependant, le kit AWS Encryption SDK ne peut pas interopérer avec d'autres bibliothèques. Comme chaque bibliothèque renvoie des données chiffrées dans un format différent, vous ne pouvez pas chiffrer avec une bibliothèque et déchiffrer avec une autre.

Client de chiffrement DynamoDB et chiffrement côté client Amazon S3

: AWS Encryption SDKImpossible de déchiffrer les données chiffrées par le client de chiffrement DynamoDB ou le chiffrement côté client Amazon S3. Ces bibliothèques ne peuvent pas déchiffrer le message crypté qu'elles AWS Encryption SDK renvoient.

AWS Key Management Service (AWS KMS)

: Les clés de données AWS Encryption SDK peuvent être utilisées AWS KMS keyspour protéger vos données, y compris les clés KMS multirégionales. Par exemple, vous pouvez configurer le AWS Encryption SDK pour crypter vos données sous un ou plusieurs AWS KMS keys de vosCompte AWS. Toutefois, vous devez utiliser le AWS Encryption SDK pour déchiffrer ces données.

  AWS Encryption SDKImpossible de déchiffrer le texte chiffré renvoyé par les opérations de AWS KMS chiffrement ou de chiffrement. ReEncrypt De même, l'opération AWS KMS Decrypt ne peut pas déchiffrer le message chiffré renvoyé par AWS Encryption SDK.

  Le ne AWS Encryption SDK prend en charge que les clés KMS de chiffrement symétriques. Vous ne pouvez pas utiliser de clé KMS asymétrique pour le chiffrement ou la connexion auAWS Encryption SDK. Le AWS Encryption SDK génère ses propres clés de signature ECDSA pour les suites d'algorithmes qui signent des messages.

Pour vous aider à choisir la bibliothèque ou le service à utiliser, voir Comment choisir un outil ou un service de AWS chiffrement dans Services et outils de chiffrement.

Support et maintenance

Il AWS Encryption SDK utilise la même politique de maintenance que le AWS SDK et les outils, y compris ses phases de version et de cycle de vie. La meilleure pratique consiste à utiliser la dernière version disponible du AWS Encryption SDK pour votre langage de programmation et à effectuer une mise à niveau au fur et à mesure que de nouvelles versions sont publiées. Lorsqu'une version nécessite des modifications importantes, telles que la mise à niveau depuis des AWS Encryption SDK versions antérieures à 1.7. x vers les versions 2.0. x et versions ultérieures, nous fournissons des instructions détaillées pour vous aider.

Chaque implémentation du langage de programmation AWS Encryption SDK est développée dans un GitHub référentiel open source distinct. Le cycle de vie et la phase de support de chaque version sont susceptibles de varier selon les référentiels. Par exemple, une version donnée de AWS Encryption SDK peut être en phase de disponibilité générale (support complet) dans un langage de programmation, mais en end-of-support phase dans un autre langage de programmation. Nous vous recommandons d'utiliser une version entièrement prise en charge dans la mesure du possible et d'éviter les versions qui ne sont plus prises en charge.

Pour connaître la phase du cycle de vie des AWS Encryption SDK versions de votre langage de programmation, consultez le `SUPPORT_POLICY.rst` fichier de chaque AWS Encryption SDK référentiel.

- Kit SDK de chiffrement AWS pour C— Support_Policy.rst
- AWS Encryption SDKpour .NET — Support_Policy.rst
- AWSCLI de chiffrement — Support_Policy.rst
- Kit SDK de chiffrement AWS pour Java— Support_Policy.rst
- Kit SDK de chiffrement AWS pour JavaScript— Support_Policy.rst
- Kit SDK de chiffrement AWS pour Python— Support_Policy.rst

Pour plus d'informations, consultez la section « Politique de maintenance AWS des SDK Versions du AWS Encryption SDK et des outils » dans le Guide de référence AWS des SDK et des outils.

En savoir plus

Si vous souhaitez en savoir plus sur le kit AWS Encryption SDK et le chiffrement côté client, veuillez consulter ces sources.

- Pour obtenir de l'aide concernant les termes et les concepts utilisés dans ce kit SDK, consultez Concepts du kit AWS Encryption SDK.
- Pour les directives relatives aux meilleures pratiques, voirBonnes pratiques pour AWS Encryption SDK.
- Pour plus d'informations sur le fonctionnement de ce kit SDK, consultez Fonctionnement du kit SDK.
- Pour des exemples illustrant comment configurer les options dans leAWS Encryption SDK, voirConfiguration du AWS Encryption SDK.
- Pour obtenir des informations techniques détaillées, consultez le document Référence.
- Pour les spécifications techniques duAWS Encryption SDK, voir les AWS Encryption SDKspécifications dans GitHub.
- Pour obtenir des réponses à vos questions sur l'utilisation duAWS Encryption SDK, lisez et publiez sur le forum de discussion AWS Crypto Tools.

Pour de plus amples informations sur l'implémentation du kit AWS Encryption SDK dans différents langages de programmation.

- C : Voir Kit SDK de chiffrement AWS pour C la documentation AWS Encryption SDK C et le aws-encryption-sdk-cdépôt sur GitHub.
- C#/.NET : Voir AWS Encryption SDKpour .NET et le aws-encryption-sdk-netrépertoire du `aws-encryption-sdk-dafny` dépôt sur. GitHub
- Interface de ligne de commande : consultez Interface de ligne de commande AWS Encryption SDK la documentation relative à la CLI de AWS chiffrement et au aws-encryption-sdk-cliréférentiel sur GitHub.
- Java : voirKit SDK de chiffrement AWS pour Java, le AWS Encryption SDK Javadoc et le aws-encryption-sdk-javadépôt activés. GitHub

  JavaScript: Voir the section called “JavaScript” et le aws-encryption-sdk-javascriptréférentiel activé GitHub.

- Python : voir Kit SDK de chiffrement AWS pour Python la documentation AWS Encryption SDK Python et le aws-encryption-sdk-pythondépôt sur GitHub.

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 vulnérabilité de sécurité potentielle dans le kit AWS Encryption SDK, veuillez avertir la sécurité AWS. Ne créez pas de GitHub problème public.
- Pour fournir des commentaires sur leAWS Encryption SDK, signalez un problème dans le GitHub référentiel du langage de programmation que vous utilisez.
- Pour nous faire part de vos commentaires sur cette documentation, utilisez les liens de commentaires sur cette page. Vous pouvez également déposer un problème ou contribuer au aws-encryption-sdk-docsréférentiel open source pour cette documentation sur GitHub.

Concepts du kit AWS Encryption SDK

Cette section présente les concepts utilisés dans le kit AWS Encryption SDK, et fournit un glossaire et une référence. Il est conçu pour vous aider à comprendre son AWS Encryption SDK fonctionnement et les termes que nous utilisons pour le décrire.

Vous avez besoin d'aide ?

- Découvrez comment le chiffrement AWS Encryption SDK des enveloppes est utilisé pour protéger vos données.
- Découvrez les éléments du chiffrement des enveloppes : les clés de données qui protègent vos données et les clés d'encapsulation qui protègent vos clés de données.
- Découvrez les porte-clés et les fournisseurs de clés principales qui déterminent les clés d'encapsulage que vous utilisez.
- Découvrez le contexte de chiffrement qui renforce l'intégrité de votre processus de chiffrement. C'est facultatif, mais c'est une bonne pratique que nous recommandons.
- Découvrez le message chiffré renvoyé par les méthodes de chiffrement.
- Vous êtes alors prêt à utiliser le AWS Encryption SDK dans votre langage de programmation préféré.

Rubriques

- Chiffrement d'enveloppe
- Clé de données
- Clé d'emballage
- Fournisseurs de porte-clés et de clés principales
- Contexte de chiffrement
- Message chiffré
- Suite d'algorithmes
- Gestionnaire de matériaux de chiffrement
- Chiffrement symétrique et asymétrique
- Engagement clé
- Politique d'engagement
- Signatures numériques

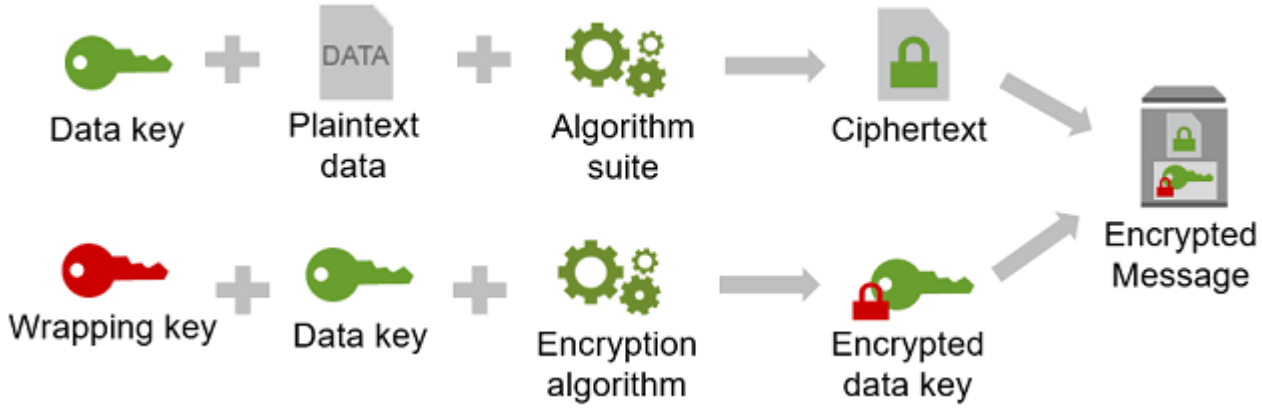
Chiffrement d'enveloppe

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

AWS Encryption SDKChiffre chaque message avec une clé de données unique. Il chiffre ensuite la clé de données sous la clé d'encapsulation que vous spécifiez. Il stocke la clé de données cryptée avec les données cryptées dans le message crypté qu'il renvoie.

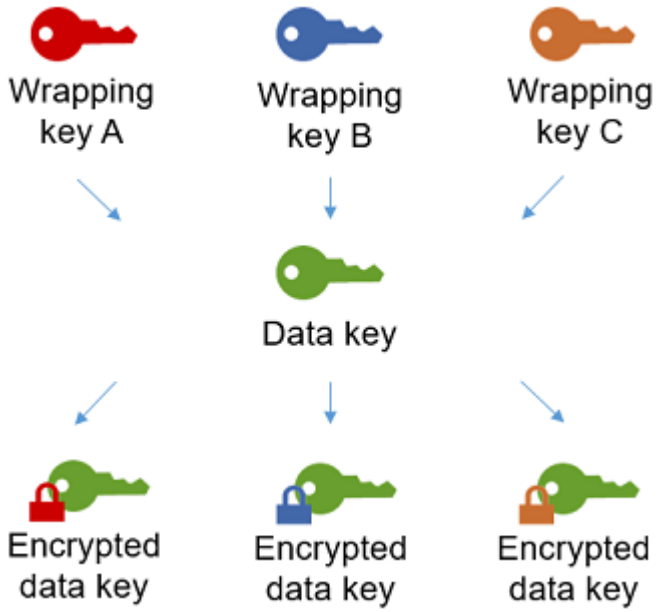
Pour spécifier votre clé d'encapsulation, vous utilisez un porte-clés ou un fournisseur de clés principales.



Chiffrer les mêmes données sous plusieurs clés d'encapsulation

Vous pouvez chiffrer la clé de données sous plusieurs clés d'encapsulation. Vous souhaiterez 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. Il AWS Encryption SDK stocke toutes les clés de données cryptées avec les données cryptées dans le message crypté.

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



Combinaison des points forts de plusieurs algorithmes

Pour chiffrer vos données, il AWS Encryption SDK utilise par défaut une suite d'algorithmes sophistiquée avec chiffrement symétrique AES-GCM, fonction de dérivation de clés (HKDF) et signature. 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. Cependant, les algorithmes de clé publique fournissent une séparation inhérente des rôles et facilitent la gestion des clés. Pour combiner les points forts de chacun, vous pouvez chiffrer vos données avec un chiffrement par clé symétrique, puis chiffrer la clé de données avec un chiffrement par clé publique.

Clé de données

Une clé de donnée est une clé de chiffrement utilisée par le kit AWS Encryption SDK pour chiffrer vos données. Chaque clé de données est un tableau d'octets qui respecte les exigences concernant les clés cryptographiques. À moins que vous n'utilisiez la mise en cache des clés de données, le kit AWS Encryption SDK utilise une clé de données unique pour chiffrer chaque message.

Vous n'avez pas besoin de spécifier, de générer, d'implémenter, d'étendre, de protéger ou d'utiliser des clés de données. Le kit AWS Encryption SDK se charge de ces tâches à votre place lorsque vous appelez les opérations de chiffrement et de déchiffrement.

Pour protéger vos clés de données, elles sont AWS Encryption SDK chiffrées à l'aide d'une ou de plusieurs clés de chiffrement appelées clés d'encapsulation ou clés principales. Après utilisation de vos clés de données en texte brut par le kit AWS Encryption SDK pour chiffrer vos données, il les supprime de la mémoire dès que possible. Ensuite, il stocke les clés de données chiffrées ainsi que les données chiffrées dans le message chiffré renvoyé par les opérations de chiffrement. Pour plus de détails, consultez the section called “Fonctionnement du kit SDK”.

> **Tip**
>
> Dans le kit AWS Encryption SDK, nous différencions les clés de données des clés de chiffrement des données. Plusieurs des suites d'algorithmes prises en charge, y compris la suite par défaut, utilisent une fonction de dérivation de clés qui empêche que la clé de données atteigne ses limites de chiffrement. La fonction de dérivation de clés utilise la

clé de données en tant qu'entrée et renvoie une clé de chiffrement des données, qui est utilisée pour chiffrer les données. 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 d'identifier plus facilement les clés AWS Encryption SDK d'encapsulation valides lors du déchiffrement.

Clé d'emballage

Une clé d'encapsulation est une clé de chiffrement AWS Encryption SDK utilisée pour chiffrer la clé de données qui chiffre vos données. Chaque clé de données en texte brut 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 ou un fournisseur de clés principales.

> **Note**
>
> La clé encapsulée fait référence aux clés d'un trousseau de clés ou d'un fournisseur de clés principales. La clé principale est généralement associée à la `MasterKey` classe que vous instanciez lorsque vous utilisez un fournisseur de clé principale.

Il AWS Encryption SDK prend en charge plusieurs clés d'encapsulation couramment utilisées, telles que AWS Key Management Service (AWS KMS) symétriques AWS KMS keys(y compris les clés KMS multirégionales), les clés brutes AES-GCM (Advanced Encryption Standard/Galois Counter Mode) et les clés RSA brutes. Vous pouvez également étendre ou implémenter vos propres clés d'encapsulation.

Lorsque vous utilisez le chiffrement des enveloppes, vous devez protéger vos clés d'emballage contre tout accès non autorisé. Vous pouvez le faire de l'une des manières suivantes :

- Utilisez un service web conçu à cet effet, comme 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 ne disposez pas de système de gestion de clés, nous vous recommandons d'utiliser AWS KMS. Il s'AWS Encryption SDKintègre AWS KMS pour vous aider à protéger et à utiliser vos clés d'emballage. Cependant, AWS Encryption SDK il ne nécessite AWS aucun AWS service.

Fournisseurs de porte-clés et de clés principales

Pour spécifier les clés d'encapsulation que vous utilisez pour le chiffrement et le déchiffrement, vous utilisez un jeu de clés (C, C#/.NET et JavaScript) ou un fournisseur de clés principales (Java, Python, CLI). Vous pouvez utiliser les trousseaux de clés et les fournisseurs de clés principales AWS Encryption SDK fournis ou concevoir vos propres implémentations. AWS Encryption SDKIl fournit des porte-clés et des fournisseurs de clés principales compatibles entre eux sous réserve de contraintes linguistiques. Pour plus de détails, consultez Compatibilité du porte-clés.

Un porte-clés génère, chiffre et déchiffre des clés de données. 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. Vous pouvez également définir un porte-clés sans clés enveloppantes ou un porte-clés plus complexe avec des options de configuration supplémentaires. Pour obtenir de l'aide sur le choix et l'utilisation des porte-clés qu'ils AWS Encryption SDK définissent, reportez-vous Utilisation des porte-clés à la section. Les porte-clés sont pris en charge en C, C#/.NET et dans la version 3. JavaScript x duKit SDK de chiffrement AWS pour Java.

Un fournisseur de clés principales est une alternative au trousseau de clés. Le fournisseur de clés principales renvoie les clés d'encapsulage (ou clés principales) que vous spécifiez. Chaque clé principale est associée à un fournisseur de clés principales, mais un fournisseur de clés principales fournit généralement plusieurs clés principales. Les fournisseurs de clés principales sont pris en charge en Java, Python et dans la CLI de AWS chiffrement.

Vous devez spécifier un trousseau de clés (ou un fournisseur de clés principales) pour le chiffrement. Vous pouvez spécifier le même trousseau de clés (ou fournisseur de clés principales), ou un autre, pour le déchiffrement. Lors du chiffrement, il AWS Encryption SDK utilise toutes les clés d'encapsulation que vous spécifiez pour chiffrer la clé de données. Lors du déchiffrement, il AWS Encryption SDK utilise uniquement les clés d'encapsulation que vous spécifiez pour déchiffrer une clé de données cryptée. La spécification des clés d'encapsulage pour le déchiffrement est facultative, mais c'est une AWS Encryption SDK bonne pratique.

Pour plus de détails sur la spécification des clés d'encapsulage, consultezSélection des clés d'emballage.

Contexte de chiffrement

Pour améliorer la sécurité de vos opérations de chiffrement, incluez un contexte de chiffrement dans toutes les demandes de chiffrement de données. L'utilisation d'un contexte de chiffrement est facultative, mais nous vous recommandons un contexte cryptographique dans le cadre des bonnes pratiques.

Un contexte de chiffrement est un ensemble de paires nom-valeur qui contient des données non secrètes arbitraires authentifiées supplémentaires. Le contexte de chiffrement peut contenir toutes les données que vous choisissez, mais il comprend généralement les données qui sont utiles pour la journalisation et le suivi, telles que les données relatives au type de fichier, à l'objectif ou à la propriété. Lorsque vous chiffrez des données, le contexte de chiffrement est lié de façon chiffrée aux données chiffrées. Le même contexte de chiffrement est donc requis pour déchiffrer les données. Le kit AWS Encryption SDK comprend le contexte de chiffrement en texte brut dans l'en-tête du message chiffré qu'il renvoie.

Le contexte de chiffrement AWS Encryption SDK utilisé comprend le contexte de chiffrement que vous spécifiez et une paire de clés publiques ajoutée par le gestionnaire de matériel cryptographique (CMM). Plus précisément, chaque fois que vous utilisez un algorithme de chiffrement avec signature, le CMM ajoute une paire nom-valeur au contexte de chiffrement et qui se compose d'un nom réservé, `aws-crypto-public-key`, et d'une valeur qui représente la clé de vérification publique. Le nom `aws-crypto-public-key` dans le contexte de chiffrement est réservé par le kit AWS Encryption SDK et ne peut pas être utilisé en tant que nom d'une autre paire dans le contexte de chiffrement. Pour plus d'informations, consultez AAD dans la Référence de format de message.

L'exemple de contexte de chiffrement suivant se compose de deux paires de contexte de chiffrement spécifiées dans la requête et de la paire de clés publique que le CMM ajoute.

```
"Purpose"="Test", "Department"="IT", aws-crypto-public-key=<public key>
```

Pour déchiffrer les données, vous transmettez le message chiffré. Dans la mesure où le kit AWS Encryption SDK peut extraire le message chiffré de l'en-tête de ce même message, vous n'êtes pas tenu de fournir le contexte de chiffrement séparément. Cependant, le contexte de chiffrement peut vous aider à confirmer que vous déchiffrez le bon message chiffré.

- Dans l'interface de ligne de commande du kit AWS Encryption SDK, si vous fournissez un contexte de chiffrement dans une commande de déchiffrement, le CLI vérifie que les valeurs sont présentes dans le contexte de chiffrement du message chiffrer avant de renvoyer les données en texte brut.

- Dans d'autres implémentations de langage de programmation, la réponse de déchiffrement inclut le contexte de chiffrement et les données en texte clair. La fonction de déchiffrement de votre application doit toujours vérifier que le contexte de chiffrement de la réponse de déchiffrement comprend le contexte de chiffrement dans la requête de chiffrement (ou un sous-ensemble) avant de renvoyer les données en texte brut.

> **Note**
>
> Avec la version 4. x du AWS Encryption SDK pour .NET et la version 3. x du Kit SDK de chiffrement AWS pour Java, vous pouvez exiger un contexte de chiffrement dans toutes les demandes de chiffrement avec le contexte de chiffrement requis CMM.

Lorsque vous choisissez un contexte de chiffrement, n'oubliez pas qu'il ne s'agit pas d'un secret. Le contexte de chiffrement est affiché en texte clair dans l'en-tête du message chiffré renvoyéAWS Encryption SDK. Si vous utilisez AWS Key Management Service, le contexte de chiffrement peut également s'afficher en texte brut dans les enregistrements et les journaux d'audit, par exemple AWS CloudTrail.

Pour des exemples de soumission et de vérification d'un contexte de chiffrement dans votre code, consultez les exemples de votre langage de programmation préféré.

Message chiffré

Lorsque vous chiffrez des données avec le kit AWS Encryption SDK, il renvoie un message chiffré.

Un message chiffré est une structure de données formatée portable qui inclut les données chiffrées ainsi que des copies chiffrées des clés de données, l'identifiant de l'algorithme et, éventuellement, un contexte de chiffrement et une signature numérique. Les opérations de chiffrement du kit AWS Encryption SDK renvoient un message chiffré et les opérations de déchiffrement utilisent un message chiffré en tant qu'entrée.

Le fait d'associer les données chiffrées et leurs clés de données chiffrées simplifie l'opération de déchiffrement et vous évite de devoir stocker et gérer des clés de données chiffrées indépendamment des données qui sont chiffrées.

Pour obtenir des informations techniques sur les messages chiffrés, consultez Format de message chiffré.

Suite d'algorithmes

Il AWS Encryption SDK utilise une suite d'algorithmes pour chiffrer et signer les données contenues dans le message crypté renvoyé par les opérations de chiffrement et de déchiffrement. Le kit AWS Encryption SDK prend en charge plusieurs suites d'algorithmes. Toutes les suites prises en charge utilisent l'algorithme AES (Advanced Encryption Standard) en tant qu'algorithme principal et l'associent à d'autres algorithmes et valeurs.

Le kit AWS Encryption SDK établit une suite d'algorithmes recommandée en tant que suite par défaut pour toutes les opérations de chiffrement. La valeur par défaut peut évoluer à mesure que les normes et les bonnes pratiques s'améliorent. Vous pouvez spécifier une suite d'algorithmes alternative dans les demandes de chiffrement de données ou lors de la création d'un gestionnaire de matériel cryptographique (CMM), mais à moins qu'une alternative ne soit requise dans votre situation, il est préférable d'utiliser la suite par défaut. La valeur par défaut actuelle est AES-GCM avec une fonction de dérivation de extract-and-expand clé basée sur HMAC (HKDF), un engagement de clé, une signature ECDSA (Elliptic Curve Digital Signature Algorithm) et une clé de chiffrement 256 bits.

Si votre application nécessite des performances élevées et que les utilisateurs qui chiffrent les données et ceux qui les déchiffrent jouissent de la même confiance, vous pouvez envisager de spécifier une suite d'algorithmes sans signature numérique. Cependant, nous recommandons vivement une suite d'algorithmes qui inclut un engagement clé et une fonction de dérivation clé. Les suites d'algorithmes dépourvues de ces fonctionnalités ne sont prises en charge que pour des raisons de rétrocompatibilité.

Gestionnaire de matériaux de chiffrement

Le gestionnaire de matériel cryptographique (CMM) assemble le matériel cryptographique utilisé pour chiffrer et déchiffrer les données. Les matériaux de chiffrement incluent des clés de données en texte brut et chiffrées, ainsi que, éventuellement, une clé de signature de message. Vous n'interagissez jamais directement avec le CMM. Les méthodes de chiffrement et de déchiffrement s'en occupent pour vous.

Vous pouvez utiliser le CMM par défaut ou le CMM de mise en cache qu'il AWS Encryption SDK fournit, ou écrire un CMM personnalisé. Et vous pouvez spécifier un CMM, mais ce n'est pas obligatoire. Lorsque vous spécifiez un trousseau de clés ou un fournisseur de clés principales, un CMM par défaut est AWS Encryption SDK créé pour vous. Le CMM par défaut obtient le matériel de chiffrement ou de déchiffrement auprès du trousseau de clés ou du fournisseur de clés principales que vous spécifiez. Pour cela, il peut être nécessaire d'appeler un service de chiffrement, comme AWS Key Management Service (AWS KMS).

Étant donné que le CMM fait office de liaison entre le AWS Encryption SDK et un trousseau de clés (ou fournisseur de clés principales), il constitue un point idéal pour la personnalisation et l'extension, notamment pour le soutien à l'application des politiques et à la mise en cache. Le kit AWS Encryption SDK fournit un CMM de mise en cache pour prendre en charge la mise en cache des clés de données.

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. Pour plus de détails, voir Algorithmes cryptographiques dans le AWSGuide des services et outils cryptographiques.

AWS Encryption SDKUtilise le cryptage des enveloppes. 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 renvoie un message crypté qui inclut les données cryptées et au moins une copie cryptée de la clé de données.

Chiffrement de vos données (chiffrement symétrique)

> Pour chiffrer vos données, il AWS Encryption SDK utilise une clé de données symétrique et une suite d'algorithmes qui inclut un algorithme de chiffrement symétrique. Pour déchiffrer les données, il AWS Encryption SDK 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 ou le fournisseur de clé principale que vous fournissez pour une opération de chiffrement et de déchiffrement détermine la manière dont la clé de données symétrique est cryptée et déchiffrée. Vous pouvez choisir un fournisseur de trousseau ou de clé principale qui utilise un chiffrement symétrique, tel qu'un trousseau de AWS KMS clés, ou un fournisseur qui utilise un chiffrement asymétrique, tel qu'un trousseau de clés RSA brut ou. `JceMasterKey`

Engagement clé

Le AWS Encryption SDK support key commitment (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 des clés garantit que seule la clé de données qui a chiffré votre message sera

utilisée pour le déchiffrer. Le chiffrement et le déchiffrement avec un engagement clé constituent une bonne pratique. AWS Encryption SDK

La plupart des chiffrements symétriques modernes (y compris AES) chiffrent un texte en clair avec une clé secrète unique, telle que la clé de données unique AWS Encryption SDK utilisée pour chiffrer chaque message en texte clair. Le déchiffrement de ces données avec la même clé de données renvoie un texte en clair identique à l'original. Le déchiffrement avec une autre clé échouera généralement. Cependant, il est possible de déchiffrer un texte chiffré sous deux clés différentes. Dans de rares cas, il est possible de trouver une clé capable de déchiffrer quelques octets de texte chiffré en un texte clair différent, mais toujours intelligible.

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

Pour éviter ce scénario, le AWS Encryption SDK support prend en charge l'engagement clé lors du chiffrement et du déchiffrement. Lorsque le AWS Encryption SDK chiffre un message avec engagement de clé, il lie cryptographiquement la clé de données unique qui a produit le texte chiffré à la chaîne d'engagement de clé, un identifiant de clé de données non secret. Ensuite, il stocke la chaîne d'engagement clé dans les métadonnées du message crypté. Lorsqu'il déchiffre un message avec une clé d'engagement, il AWS Encryption SDK vérifie que la clé de données est la seule et unique clé pour ce message crypté. Si la vérification de la clé de données échoue, l'opération de déchiffrement échoue.

Support pour les engagements clés est introduit dans la version 1.7. x, qui peut déchiffrer les messages avec un engagement clé, mais ne le chiffrera pas avec un engagement clé. Vous pouvez utiliser cette version pour déployer pleinement la capacité de déchiffrer le texte chiffré avec un engagement clé. La version 2.0. x inclut une prise en charge complète des principaux engagements. Par défaut, il chiffre et déchiffre uniquement avec un engagement clé. Il s'agit d'une configuration idéale pour les applications qui n'ont pas besoin de déchiffrer le texte chiffré par les versions antérieures du. AWS Encryption SDK

Bien que le chiffrement et le déchiffrement avec engagement clé soient une bonne pratique, nous vous laissons décider du moment de leur utilisation et du rythme auquel vous les adoptez. À partir de la version 1.7. x, AWS Encryption SDK prend en charge une politique d'engagement qui définit la

suite d'algorithmes par défaut et limite les suites d'algorithmes pouvant être utilisées. Cette politique détermine si vos données sont cryptées et déchiffrées avec un engagement clé.

L'engagement clé se traduit par un message crypté légèrement plus volumineux (+ 30 octets) et son traitement prend plus de temps. Si votre application est très sensible à la taille ou aux performances, vous pouvez choisir de ne pas participer à l'engagement clé. Mais ne le faites que si vous le devez.

Pour plus d'informations sur la migration vers les versions 1.7. x et 2.0. x, y compris leurs principales caractéristiques d'engagement, voirMigrer votreAWS Encryption SDK. Pour obtenir des informations techniques sur les principaux engagements, voir the section called “Référence relative aux algorithmes du kit” etthe section called “Référence Format des messages du kit”.

Politique d'engagement

Une politique d'engagement est un paramètre de configuration qui détermine si votre application chiffre et déchiffre avec un engagement clé. Le chiffrement et le déchiffrement avec un engagement clé constituent une bonne pratique. AWS Encryption SDK

La politique d'engagement repose sur trois valeurs.

> **Note**
>
> Il se peut que vous deviez faire défiler le tableau horizontalement ou verticalement pour voir le tableau dans son intégralité.

Valeurs de la politique d'engagement

Valeur	Chiffre avec un engagement clé	Chiffre sans engagement de clé	Déchiffre avec un engagement clé	Déchiffre sans engagement clé
ForbidEncryptAllowDecrypt	✗	✓	✓	✓
RequireEncryptAllowDecrypt	✓	✗	✓	✓

Valeur	Chiffre avec un engagement clé	Chiffre sans engagement de clé	Déchiffre avec un engagement clé	Déchiffre sans engagement clé
RequireEncryptRequireDecrypt				

Le paramétrage de la politique d'engagement est introduit dans AWS Encryption SDK la version 1.7. x. Il est valide dans tous les langages de programmation pris en charge.

- `ForbidEncryptAllowDecrypt`déchiffre avec ou sans engagement clé, mais il ne chiffrera pas avec engagement clé. Il s'agit de la seule valeur valide pour la politique d'engagement dans la version 1.7. x et il est utilisé pour toutes les opérations de chiffrement et de déchiffrement. Il est conçu pour préparer tous les hôtes exécutant votre application à déchiffrer avec clé d'engagement avant qu'ils ne rencontrent un texte chiffré avec clé d'engagement.
- `RequireEncryptAllowDecrypt`chiffre toujours avec clé d'engagement. Il peut être déchiffré avec ou sans engagement clé. Cette valeur, introduite dans la version 2.0. x, vous permet de commencer à chiffrer avec un engagement clé, tout en déchiffrant les anciens textes chiffrés sans engagement clé.
- `RequireEncryptRequireDecrypt`chiffre et déchiffre uniquement avec un engagement clé. Cette valeur est la valeur par défaut pour la version 2.0. x. Utilisez cette valeur lorsque vous êtes certain que tous vos textes chiffrés sont chiffrés avec une clé d'engagement.

Le paramètre de politique d'engagement détermine les suites d'algorithmes que vous pouvez utiliser. À partir de la version 1.7. x, AWS Encryption SDK prend en charge les suites d'algorithmes pour l'engagement des clés, avec ou sans signature. Si vous spécifiez une suite d'algorithmes qui entre en conflit avec votre politique d'engagement, une erreur est AWS Encryption SDK renvoyée.

Pour obtenir de l'aide pour définir votre politique d'engagement, consultezDéfinition de votre politique d'engagement.

Signatures numériques

Pour garantir l'intégrité d'un message numérique lorsqu'il passe d'un système à l'autre, vous pouvez appliquer une signature numérique au message. Les signatures numériques sont toujours

asymétriques. Vous utilisez votre clé privée pour créer la signature et l'ajouter au message d'origine. Votre destinataire utilise une clé publique pour vérifier que le message n'a pas été modifié depuis que vous l'avez signé.

AWS Encryption SDKChiffre vos données à l'aide d'un algorithme de cryptage authentifié, AES-GCM, et le processus de déchiffrement vérifie l'intégrité et l'authenticité d'un message crypté 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 une AWS KMS clé comme clé d'encapsulation, cela signifie qu'il est possible pour un utilisateur disposant des autorisations KMS Decrypt de créer des textes chiffrés sans appeler KMS Encrypt.

Pour éviter ce problème, il est possible d'AWS Encryption SDKajouter une signature ECDSA (Elliptic Curve Digital Signature Algorithm) à la fin des messages chiffrés. Lorsqu'une suite d'algorithmes de signature est utilisée, une clé privée temporaire et une paire de clés publiques sont générées pour chaque message chiffré. AWS Encryption SDK Il AWS Encryption SDK stocke la clé publique dans le contexte de chiffrement de la clé de données et supprime la clé privée, et personne ne peut créer une autre signature vérifiant avec la clé publique. Étant donné que l'algorithme lie la clé publique à la clé de données cryptée en tant que données authentifiées supplémentaires dans l'en-tête du message, un utilisateur qui ne peut que déchiffrer des messages ne peut pas modifier la clé publique.

La vérification des signatures entraîne un coût de performance significatif lors du déchiffrement. Si les utilisateurs qui chiffrent les données et ceux qui les déchiffrent jouissent de la même confiance, envisagez d'utiliser une suite d'algorithmes qui n'inclut pas la signature.

Fonctionnement du kit AWS Encryption SDK

Les flux de travail présentés dans cette section expliquent comment AWS Encryption SDK crypte les données et déchiffre les messages chiffrés. Ces flux de travail décrivent le processus de base utilisant les fonctionnalités par défaut. Pour plus de détails sur la définition et l'utilisation de composants personnalisés, consultez le GitHub référentiel de chaque implémentation de langage prise en charge.

Il AWS Encryption SDK utilise le cryptage des enveloppes pour protéger vos données. Chaque message est crypté sous une clé de données unique. La clé de données est ensuite chiffrée par les clés d'encapsulation que vous spécifiez. Pour déchiffrer le message chiffré, il AWS Encryption SDK utilise les clés d'encapsulation que vous spécifiez pour déchiffrer au moins une clé de données cryptée. Il peut ensuite déchiffrer le texte chiffré et renvoyer un message en texte clair.

Vous avez besoin d'aide avec la terminologie que nous utilisons dans le AWS Encryption SDK ? Consultez the section called “Concepts”.

Comment AWS Encryption SDK crypte les données

AWS Encryption SDKFournit des méthodes qui chiffrent les chaînes, les tableaux d'octets et les flux d'octets. Pour des exemples de code, consultez la rubrique Exemples de chaque Langages de programmation section.

1. Créez un trousseau de clés (ou un fournisseur de clés principales) qui spécifie les clés d'encapsulation qui protègent vos données.
2. Transmettez le trousseau de clés et les données en texte brut à une méthode de cryptage. Nous vous recommandons de transmettre un contexte de chiffrement facultatif et non secret.
3. La méthode de cryptage demande au trousseau de clés le matériel de cryptage. Le trousseau de clés renvoie des clés de chiffrement de données uniques pour le message : une clé de données en texte brut et une copie de cette clé de données chiffrée par chacune des clés d'encapsulation spécifiées.
4. La méthode de chiffrement utilise la clé de données en texte brut pour chiffrer les données, puis la supprime. Si vous fournissez un contexte de chiffrement (une AWS Encryption SDK bonne pratique), la méthode de chiffrement lie cryptographiquement le contexte de chiffrement aux données chiffrées.
5. La méthode de chiffrement renvoie un message chiffré contenant les données chiffrées, les clés de données chiffrées et d'autres métadonnées, y compris le contexte de chiffrement, si vous en avez utilisé un.

Comment AWS Encryption SDK décrypte un message crypté

AWS Encryption SDKfournit des méthodes qui déchiffrent le message chiffré et renvoient du texte en clair. Pour des exemples de code, consultez la rubrique Exemples de chaque Langages de programmation section.

Le trousseau de clés (ou fournisseur de clé principale) qui déchiffre le message chiffré doit être compatible avec celui utilisé pour chiffrer le message. L'une de ses clés d'encapsulation doit être capable de déchiffrer une clé de données cryptée dans le message crypté. Pour plus d'informations sur la compatibilité avec les trousseaux de clés et les fournisseurs de clés principales, consultezthe section called “Compatibilité du porte-clés”.

1. Créez un trousseau de clés ou un fournisseur de clés principales avec des clés d'encapsulation capables de déchiffrer vos données. Vous pouvez utiliser le même trousseau de clés que celui que vous avez fourni pour la méthode de chiffrement ou un autre.
2. Passez le message crypté et le trousseau de clés à une méthode de déchiffrement.
3. La méthode de déchiffrement demande au trousseau de clés ou au fournisseur de clés principales de déchiffrer l'une des clés de données chiffrées du message chiffré. Il transmet des informations à partir du message chiffré, y compris les clés de données chiffrées.
4. Le porte-clés utilise ses clés d'encapsulage pour déchiffrer l'une des clés de données chiffrées. En cas de succès, la réponse inclut la clé de données en texte brut. Si aucune des clés d'encapsulation spécifiées par le trousseau de clés ou le fournisseur de clés principales ne peut déchiffrer une clé de données chiffrée, l'appel de déchiffrement échoue.
5. La méthode de déchiffrement utilise la clé de données en texte brut pour déchiffrer les données, supprime la clé de données en texte clair et renvoie les données en texte clair.

Suites d'algorithmes prises en charge dans le kit AWS Encryption SDK

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

La suite d'algorithmes du kit AWS Encryption SDK utilise l'algorithme AES (Advanced Encryption Standard) en mode GCM (Galois Counter Mode), connu sous le nom de AES-GCM, pour chiffrer les données brutes. LeAWS Encryption SDKprend en charge uniquement les clés de chiffrement 256, 192 et 128 bits. La longueur du vecteur d'initialisation (IV) est toujours de 12 octets. La longueur de la balise d'authentification est toujours de 16 octets.

Par défaut, leAWS Encryption SDKutilise une suite d'algorithmes avec AES-GCM avec un système HMACextract-and-expandfonction de dérivation des clés (HKDF), la signature et une clé de chiffrement 256 bits. Si l'icônepolitique d'engagementa besoinEngagement clé, leAWS Encryption SDKsélectionne une suite d'algorithmes qui prend également en charge l'engagement clé ; sinon, elle sélectionne une suite d'algorithmes avec dérivation et signature de clés, mais pas un engagement clé.

Recommandation : AES-GCM avec dérivation des clés, signature et engagement clé

LeAWS Encryption SDKrecommande une suite d'algorithmes qui dérive une clé de chiffrement AES-GCM en fournissant une clé de chiffrement de données 256 bits à la base HMACextract-and-expandfonction de dérivation des clés (HKDF). LeAWS Encryption SDKajoute une signature ECDSA (Elliptic Curve Digital Signature Algorithm). Pour prendre en chargeEngagement clé, cette suite d'algorithmes dérive également unchaîne d'engagement clé— un identifiant de clé de données non secret — qui est stocké dans les métadonnées du message chiffré. Cette chaîne d'engagement de clé est également dérivée via HKDF à l'aide d'une procédure similaire à la dérivation de la clé de chiffrement des données.

AWS Encryption SDKSuite d'algorithmes

Algorithme de chiffrement	Longueur de la clé de chiffrement des données (en bits)	Algorithme de dérivation de clé	Algorithme de signature	Engagement clé
AES-GCM	256	HKDF avec SHA-384	ECDSA avec P-384 et SHA-384	HKDF avec SHA-512

La clé HKDF vous permet d'éviter la réutilisation accidentelle d'une clé de chiffrement des données et réduit le risque de surutilisation d'une clé de données.

Pour la signature, cette suite d'algorithmes utilise ECDSA avec un algorithme de fonction de hachage chiffré (SHA-384). ECDSA est utilisé par défaut, même lorsqu'il n'est pas spécifié par la stratégie pour la clé principale sous-jacente. Signature des messagesvérifie que l'expéditeur du message a été autorisé à chiffrer les messages et qu'il n'a pas de répudiation. Cela s'avère particulièrement utile lorsque la stratégie d'autorisation pour une clé principale autorise un ensemble d'utilisateurs à chiffrer des données et un autre ensemble d'utilisateurs à déchiffrer des données.

Les suites d'algorithmes avec engagement clé garantissent que chaque texte chiffré ne décrypte qu'un seul texte brut. Pour ce faire, ils valident l'identité de la clé de données utilisée comme entrée dans l'algorithme de chiffrement. Lors du chiffrement, ces suites d'algorithmes dérivent une chaîne d'engagement clé. Avant de déchiffrer, ils vérifient que la clé de données correspond à la chaîne d'engagement de clé. Si ce n'est pas le cas, l'appel de déchiffrement échoue.

Autres suites d'algorithmes prises en charge

Le kit AWS Encryption SDK prend en charge les suites d'algorithme alternatif suivantes pour assurer la compatibilité descendante. En général, nous vous déconseillons ces suites d'algorithme. Cependant, nous reconnaissons que la signature peut entraver considérablement les performances, c'est pourquoi nous proposons une suite d'engagement clé avec dérivation des clés pour ces cas. Pour les applications qui doivent comporter des compromis plus importants sur les performances, nous continuons d'offrir des suites qui manquent de signature, d'engagement clé et de dérivées clés.

AES-GCM sans engagement clé

Les suites d'algorithmes sans engagement de clé ne valident pas la clé de données avant le déchiffrement. Par conséquent, ces suites d'algorithmes peuvent déchiffrer un seul texte chiffré en différents messages en texte brut. Cependant, parce que les suites d'algorithmes avec un engagement clé produisent unmessage chiffré légèrement plus grand (+30 octets)et ils prennent plus de temps à traiter, ils ne sont peut-être pas le meilleur choix pour chaque application.

LeAWS Encryption SDKprend en charge une suite d'algorithmes avec dérivation de clé, engagement clé, signature et une suite avec dérivation de clé et engagement clé, mais pas de signature. Nous ne recommandons pas d'utiliser une suite d'algorithmes sans engagement clé. Si nécessaire, nous recommandons une suite d'algorithmes avec dérivation de clés et engagement de clé, mais pas de signature. Toutefois, si le profil de performances de votre application prend en charge l'utilisation d'une suite d'algorithmes, l'utilisation d'une suite d'algorithmes avec engagement clé, dérivation de clés et signature est une bonne pratique.

AES-GCM sans signature

Les suites d'algorithmes sans signature ne disposent pas de la signature ECDSA qui fournit l'authenticité et la non-répudiation. Utilisez ces suites uniquement lorsque les utilisateurs qui chiffrent les données et ceux qui déchiffrent font l'objet d'une même approbation.

Lorsque vous utilisez une suite d'algorithmes sans signature, nous vous recommandons d'en choisir une avec la dérivation de clés et l'engagement de clé.

AES-GCM sans dérivation de clé

Les suites d'algorithmes sans dérivation de clé utilisent la clé de chiffrement des données en tant que clé de chiffrement AES-GCM, au lieu d'utiliser une fonction de dérivation de clé pour dériver une clé unique. Nous déconseillons d'utiliser cette suite pour générer un texte chiffré, mais leAWS Encryption SDKle prend en charge pour des raisons de compatibilité.

Pour plus d'informations sur la façon dont ces suites sont représentées et utilisées dans la bibliothèque, consultez the section called “Référence relative aux algorithmes du kit”.

Utilisation de l'AWS Encryption SDK avec AWS KMS

Pour utiliser leAWS Encryption SDK, vous devez configurer des trousses de clés ou des fournisseurs de clés principales avec des clés d'encapsulation. Si vous n'avez pas d'infrastructure à clé, nous vous recommandons d'utiliser AWS Key Management Service (AWS KMS). La plupart des exemples de code duAWS Encryption SDK nécessitent un AWS KMS key.

Pour interagir avecAWS KMS,AWS Encryption SDK il faut disposer duAWS SDK correspondant à votre langage de programmation préféré. La bibliothèqueAWS Encryption SDK cliente fonctionne avec lesAWS SDK pour prendre en charge les clés principales stockées dansAWS KMS.

Pour vous préparer à utiliser le kit AWS Encryption SDK avec AWS KMS

1. Créez un Compte AWS. Pour savoir comment créer et activer un nouveau compte Amazon Web Services ? dans leAWS Knowledge Center.
2. Créez un chiffrement symétriqueAWS KMS key. Pour obtenir de l'aide, consultez la section Création de clés dans le Guide duAWS Key Management Service développeur.

   > **Tip**
   >
   > Pour utiliser la clé AWS KMS key par programmation, vous avez besoin de l'ID ou de l'Amazon Resource Name (ARN) de la clé AWS KMS key. Pour obtenir de l'aide pour trouver l'ID ou l'ARN d'unAWS KMS key, consultez la section Recherche de l'ID de clé et de l'ARN dans le Guide duAWS Key Management Service développeur.

3. Générez un ID de 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 utiliser leAWS 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. Comme bonne pratique en matière 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 d'utilisateur IAMAWS (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 vosAWS informations d'identification en suivant les instructions du AWS SDK for JavaAWS SDK for JavaScript, AWS SDK for Python (Boto)ou AWS SDK for C++(pour C), ainsi que 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 devez également spécifier le jeton de session.

   Cette procédure permet aux kits SDK AWS de signer les demandes envoyées à AWS pour vous. Les exemples de code du kit AWS Encryption SDK qui interagit avec AWS KMS supposent que vous avez terminé cette étape.

5. Téléchargez et installez le kit AWS Encryption SDK. Pour savoir comment procéder, consultez les instructions d'installation pour le langage de programmation que vous souhaitez utiliser.

Bonnes pratiques pour AWS Encryption SDK

LeAWS Encryption SDKest conçu pour vous faciliter la protection de vos données à l'aide des normes du secteur et des bonnes pratiques. Alors que de nombreuses bonnes pratiques sont sélectionnées pour vous dans les valeurs par défaut, certaines pratiques sont facultatives mais recommandées chaque fois que cela est possible.

Utilisation de la version la plus récente

Dès le début de l'utilisation duAWS Encryption SDK, utilisez la version la plus récente proposée dans votreLangages de programmation. Si vous avez utilisé leAWS Encryption SDK, effectuez la mise à niveau vers chaque version la plus récente dès que possible. Cela garantit que vous utilisez la configuration recommandée et que vous profitez des nouvelles propriétés de sécurité pour protéger vos données. Pour plus de détails sur les versions prises en charge, y compris des conseils pour la migration et le déploiement,Support et maintenanceetVersions du AWS Encryption SDK.

Si une nouvelle version rend obsolète des éléments de votre code, remplacez-les dès que possible. Les avertissements d'obsolescence et les commentaires de code recommandent généralement une bonne alternative.

Pour faciliter les mises à niveau importantes et réduire les risques d'erreur, nous proposons parfois une version temporaire ou transitoire. Utilisez ces versions et la documentation qui l'accompagne pour vous assurer de pouvoir mettre à niveau votre application sans perturber votre flux de production.

Utilisation des valeurs par défaut

LeAWS Encryption SDKintègre les meilleures pratiques dans ses valeurs par défaut. Dans la mesure du possible, utilisez-les. Dans les cas où la valeur par défaut n'est pas pratique, nous proposons des alternatives, telles que des suites d'algorithmes sans signature. Nous offrons également des possibilités de personnalisation aux utilisateurs avancés, tels que des porte-clés personnalisés, des fournisseurs de clés principales et des gestionnaires de matériel cryptographique (GMAO). Utilisez ces alternatives avancées avec prudence et faites vérifier vos choix par un ingénieur en sécurité dans la mesure du possible.

Utilisation d'un contexte de chiffrement

Pour améliorer la sécurité de vos opérations de chiffrement, incluez uncontexte de chiffrementavec une valeur significative dans toutes les demandes de chiffrement de données.

L'utilisation d'un contexte de chiffrement est facultative, mais nous vous recommandons un contexte cryptographique dans le cadre des bonnes pratiques. Un contexte de chiffrement fournit des données authentifiées supplémentaires (données AAD) pour le chiffrement authentifié dans leAWS Encryption SDK. Bien qu'il ne s'agit pas d'un secret, le contexte de chiffrement peut vous aider àprotéger l'intégrité et l'authenticitéde vos données cryptées.

DansAWS Encryption SDK, vous ne spécifiez un contexte de chiffrement que lors du chiffrement. Lors du déchiffrement, leAWS Encryption SDKutilise le contexte de chiffrement dans l'en-tête du message chiffré que leAWS Encryption SDKrenvoie. Avant que votre application renvoie des données en texte brut, vérifiez que le contexte de chiffrement que vous avez utilisé pour chiffrer le message est inclus dans le contexte de chiffrement qui a été utilisé pour déchiffrer le message. Pour plus d'informations, consultez les exemples de votre langage de programmation.

Lorsque vous utilisez l'interface de ligne de commande, leAWS Encryption SDKvérifie le contexte de chiffrement pour vous.

Protégez vos clés d'emballage

LeAWS Encryption SDKgénère une clé de données unique pour chiffrer chaque message en texte brut. Ensuite, il crypte la clé de données avec les clés d'encapsulation que vous fournissez. Si vos clés d'encapsulation sont perdues ou supprimées, vos données chiffrées sont irrécupérables. Si vos clés ne sont pas sécurisées, vos données peuvent être vulnérables.

Utilisez des clés d'encapsulation protégées par une infrastructure de clés sécurisée, telle queAWS Key Management Service(AWS KMS). Lorsque vous utilisez des clés AES ou RSA brutes, utilisez une source aléatoire et un stockage durable qui répond à vos exigences de sécurité. Génération et stockage de clés d'encapsulation dans un module de sécurité matérielle (HSM) ou un service qui fournit des HSM, commeAWS CloudHSM, est une bonne pratique.

Utilisez les mécanismes d'autorisation de votre infrastructure de clés pour limiter l'accès à vos clés d'encapsulation aux seuls utilisateurs qui en ont besoin. Mettre en œuvre les principes des meilleures pratiques, tels que le moindre privilège. Lorsque vous utilisezAWS KMS keys, utilisez des stratégies de clés et des stratégies IAM qui mettent en œuvrePrincipes de bonnes pratiques.

Spécifiez vos clés d'emballage

C'est toujours une bonne pratique despécifiez vos clés d'emballageexplicitement lors du déchiffrement, ainsi que du chiffrement. Lorsque vous le faites, leAWS Encryption SDKutilise uniquement les clés que vous spécifiez. Cette pratique garantit que vous n'utilisez que les clés de chiffrement que vous souhaitez utiliser. PourAWS KMSenrubannage des clés, il améliore également les performances en vous empêchant d'utiliser par inadvertance des clés dans un

autreCompte AWSou Region, ou en tentant de déchiffrer avec des clés que vous n'êtes pas autorisé à utiliser.

Lors du chiffrement, les trousseaux de clés et les fournisseurs de clés principales queAWS Encryption SDKles fournitures nécessitent que vous spécifiiez des clés d'encapsulation. Ils utilisent toutes et uniquement les clés d'encapsulation que vous spécifiez. Vous devez également spécifier des clés d'encapsulation lors du chiffrement et du déchiffrement avec des trousseaux de clés AES bruts, des porte-clés RSA bruts et JCEMasterKeys.

Toutefois, lors du déchiffrement avecAWS KMStrousseaux de clés et fournisseurs de clés principales, vous n'êtes pas obligé de spécifier des clés d'encapsulation. LeAWS Encryption SDKpeut obtenir l'identifiant de clé à partir des métadonnées de la clé de données chiffrée. Toutefois, la spécification des clés d'encapsulation est une bonne pratique que nous recommandons.

Pour prendre en charge cette meilleure pratique lorsque vous travaillez avecAWS KMSEnveloppement de clés, nous recommandons les points suivants :

- UtiliserAWS KMSporte-clés qui spécifient les clés d'habillage. Lors du chiffrement et du déchiffrement, ces trousseaux de clés utilisent uniquement les clés d'encapsulation que vous avez spécifiées.
- Lorsque vous utilisezAWS KMSles clés principales et les fournisseurs de clés principales, utilisent les constructeurs en mode strict introduits dansVersion 1.7.h/24, j/7duAWS Encryption SDK. Ils créent des fournisseurs qui chiffrent et déchiffrent uniquement avec les clés d'encapsulation que vous spécifiez. Les constructeurs des fournisseurs de clés principales qui déchiffrent toujours avec n'importe quelle clé d'encapsulation sont déconseillés dans la version 1.7.h/24, j/7et supprimé dans la version 2.0.h/24, j/7.

Lors de la spécificationAWS KMSl'encapsulation des clés pour le déchiffrement n'est pas pratique, vous pouvez utiliser des fournisseurs de découverte. LeAWS Encryption SDKen C et JavaScript soutenirAWS KMSporte-clés découverte. Les fournisseurs de clés principales avec un mode de découverte sont disponibles pour Java et Python dans les versions 1.7.h/24, j/7et version ultérieure. Ces fournisseurs de découverte, qui ne sont utilisés que pour le déchiffrement avecAWS KMSclés d'encapsulation, dirigez explicitementAWS Encryption SDKpour utiliser n'importe quelle clé d'encapsulation qui crypte une clé de données.

Si vous devez utiliser un fournisseur de découverte, utilisez sonFiltre de découvertefonctionnalités pour limiter les clés d'encapsulation qu'ils utilisent. Par exemple, les recettesAWS KMSporte-clés découverte régionalutilise uniquement les clés d'habillage d'unRégion AWS. Vous pouvez

également configurerAWS KMSporte-clés etAWS KMS Fournisseurs de clés principalespour utiliser uniquement leclés d'emballageen particulierComptes AWS. Également, comme toujours, utilisez des stratégies de clés et des stratégies IAM pour contrôler l'accès à votreAWS KMSclés d'emballage.

Utiliser des signatures numériques

Il est recommandé d'utiliser une suite d'algorithmes avec signature. Signatures numériquesvérifier que l'expéditeur du message était autorisé à envoyer le message et protéger l'intégrité du message. Toutes les versions duAWS Encryption SDKutiliser des suites d'algorithmes avec signature par défaut.

Si vos exigences de sécurité n'incluent pas les signatures numériques, vous pouvez sélectionner une suite d'algorithmes sans signatures numériques. Nous recommandons toutefois d'utiliser des signatures numériques, en particulier lorsqu'un groupe d'utilisateurs chiffre des données et qu'un autre groupe d'utilisateurs déchiffre ces données.

Utilisation des engagements de clé

Il est recommandé d'utiliser la fonctionnalité de sécurité d'engagement clé. En vérifiant l'identité duclé de donnéesqui a crypté vos données,Engagement de clévous empêche de déchiffrer tout texte chiffré pouvant donner lieu à plusieurs messages en texte brut.

LeAWS Encryption SDKfournit une prise en charge complète du chiffrement et du déchiffrement avec un engagement clé à partir deversion 2.0.h/24, j/7. Par défaut, tous vos messages sont chiffrés et déchiffrés avec engagement clé. Version 1.7.h/24, j/7duAWS Encryption SDKpeut décrypter des textes chiffrés avec engagement clé. Il est conçu pour aider les utilisateurs des versions antérieures à déployer la version 2.0.h/24, j/7Réussite.

La Support des engagements clés comprendnouvelles suites d'algorithmeset unnouveau format des messagesqui produit un texte chiffré de seulement 30 octets plus grand qu'un texte chiffré sans engagement de clé. La conception minimise son impact sur les performances afin que la plupart des utilisateurs puissent profiter des avantages d'un engagement clé. Si votre application est très sensible à la taille et aux performances, vous pouvez décider d'utiliser lePolitique d'engagementparamètre pour désactiver l'engagement clé ou autoriserAWS Encryption SDKpour décrypter des messages sans engagement, mais ne le faites que si vous le devez.

Limiter le nombre de clés de données chiffrées

C'est une bonne pratique deLimiter le nombre de clés de données chiffréesdans les messages que vous déchiffrez, en particulier les messages provenant de sources non fiables. Le

déchiffrement d'un message contenant de nombreuses clés de données chiffrées que vous ne pouvez pas déchiffrer peut entraîner des retards prolongés, augmenter les dépenses, limiter votre application et d'autres personnes partageant votre compte, et potentiellement épuiser votre infrastructure de clés. Sans limite, un message chiffré peut contenir jusqu'à 65 535 clés de données chiffrées (2^16 - 1). Pour plus d'informations, consultez Limiter les clés de données chiffrées.

Pour plus d'informations sur leAWS Encryption SDKfonctions de sécurité qui sous-tendent ces bonnes pratiques, consultez.Client Encryption amélioré : EXPLICITE KeyIds Engagementdans leAWSBlog de sécurité.

Configuration du AWS Encryption SDK

AWS Encryption SDKIl est conçu pour être facile à utiliser. Bien qu'il AWS Encryption SDK dispose de plusieurs options de configuration, les valeurs par défaut sont soigneusement choisies pour être pratiques et sûres 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.

Lorsque vous configurez votre implémentation, passez en revue les AWS Encryption SDK meilleures pratiques et mettez-en en œuvre autant que possible.

Rubriques

Sélection d'un langage de programmation

AWS Encryption SDKIl 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. Toutefois, vous pouvez sélectionner un langage de programmation pour une implémentation particulière. Par exemple, si vous préférez travailler avec des porte-clés, vous pouvez choisir le Kit SDK de chiffrement AWS pour C ou leKit SDK de chiffrement AWS pour JavaScript.

Sélection des clés d'emballage

AWS Encryption SDKGénère une clé de données symétrique unique pour chiffrer chaque message. À moins que vous n'utilisiez la mise en cache des clés de données, vous n'avez pas besoin de configurer, de gérer ou d'utiliser les clés de données. Ils AWS Encryption SDK le font pour toi.

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

Pour spécifier vos clés d'encapsulation pour le chiffrement et le déchiffrement, vous utilisez un trousseau de clés (C et JavaScript) ou un fournisseur de clés principales (Java, Python, Encryption AWS CLI). 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 chiffrées (une par clé d'encapsulage) sont stockées avec les données cryptées dans le message crypté AWS Encryption SDK renvoyé. Pour déchiffrer les données, vous AWS Encryption SDK devez d'abord utiliser l'une de vos clés d'encapsulation pour déchiffrer une clé de données cryptée.

Pour spécifier un fournisseur de clés ou de clés principales AWS KMS key dans un trousseau de clés, utilisez un identifiant de AWS KMS clé compatible. 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 Servicedéveloppeur.

- Lors du chiffrement à l'Kit SDK de chiffrement AWS pour Javaaide de la CLI de AWS chiffrement, vous pouvez utiliser n'importe quel identifiant de clé valide (ID de clé, ARN de clé, nom d'alias ou ARN d'alias) pour une clé KMS. Kit SDK de chiffrement AWS pour JavaScript Kit SDK de chiffrement AWS pour Python Lorsque vous chiffrez avec leKit SDK de chiffrement AWS pour C, vous ne pouvez utiliser qu'un identifiant de clé ou un ARN de clé.

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

- Lors du déchiffrement en mode strict (où vous spécifiez des clés d'encapsulation particulières), vous devez utiliser un ARN de clé pour l'identifier. AWS KMS keys Cette exigence s'applique à toutes les implémentations de langage du kit AWS Encryption SDK.

  Lorsque vous chiffrez avec un AWS KMS trousseau de clés, l'ARN de la clé est AWS Encryption SDK stocké AWS KMS key dans les métadonnées de la clé de données cryptée. Lors du déchiffrement en mode strict, il AWS Encryption SDK vérifie que le même ARN de clé apparaît dans le trousseau de clés (ou dans le fournisseur de clés principales) avant de tenter d'utiliser la clé d'encapsulation pour déchiffrer la clé de données chiffrée. Si vous utilisez un identifiant de clé différent, il ne le AWS Encryption SDK reconnaîtra ni ne l'utiliseraAWS KMS key, même si les identifiants font référence à la même clé.

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. Dans un fournisseur de clé principale, `Provider ID` il s'agit de l'équivalent de l'espace de noms et de l'équivalent du nom. `Key ID` 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, la clé d'encapsulage ne AWS Encryption SDK sera pas reconnue ni utilisée, même si le contenu de la clé est le même.

Utilisation de plusieurs régions AWS KMS keys

Vous pouvez utiliser AWS Key Management Service (AWS KMS) des clés multirégionales comme clés d'encapsulation dans leAWS Encryption SDK. Si vous chiffrez avec une clé multirégionale dans une cléRégion AWS, vous pouvez déchiffrer en utilisant une clé multirégionale associée dans une autre. Région AWS Support pour les clés multirégionales a été introduit dans la version 2.3. x du AWS Encryption SDK et version 3.0. x de la CLI AWS de chiffrement.

AWS KMSLes clés 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 Servicedéveloppeur.

Pour prendre en charge les clés multirégionales, des porte-clés et des fournisseurs de clés principales AWS KMS compatibles avec plusieurs régions sont AWS Encryption SDK inclus. Le

nouveau symbole prenant en compte plusieurs régions dans chaque langage de programmation prend en charge les clés à région unique et à régions multiples.

- Pour les clés à région unique, le symbole compatible avec plusieurs régions se comporte comme le porte-clés à région unique AWS KMS et le fournisseur de clés principales. Il tente de déchiffrer le texte chiffré uniquement à l'aide de la clé à région unique qui a chiffré les données.
- Pour les clés multirégionales, le symbole compatible avec plusieurs régions 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 trousseaux de clés compatibles avec plusieurs régions et les fournisseurs de clés principales qui acceptent plusieurs clés KMS, vous pouvez spécifier plusieurs clés à région unique et 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.

Vous pouvez également utiliser une clé multirégionale avec les porte-clés standard à région unique et les fournisseurs de AWS KMS clés principales. Toutefois, vous devez utiliser la même clé multirégionale dans la même région pour chiffrer et déchiffrer. Les trousseaux de clés à région unique et les fournisseurs de clés principales tentent de déchiffrer le texte chiffré uniquement avec les clés qui ont chiffré les données.

Les exemples suivants montrent comment chiffrer et déchiffrer des données à l'aide de clés multirégionales et des nouveaux porte-clés et fournisseurs de clés principales compatibles avec plusieurs régions. Ces exemples chiffrent les données de la `us-east-1` région et déchiffrent les données de la `us-west-2` région à l'aide des clés multirégionales associées dans chaque région. Avant d'exécuter ces exemples, remplacez l'exemple d'ARN clé multirégional par une valeur valide provenant de votreCompte AWS.

C

Pour chiffrer avec une clé multirégionale, utilisez la `Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder()` méthode d'instanciation du trousseau de clés. Spécifiez une clé multirégionale.

Cet exemple simple n'inclut pas de contexte de chiffrement. Pour un exemple d'utilisation d'un contexte de chiffrement en C, consultezChiffrement et déchiffrement de chaînes.

Pour un exemple complet, consultez le fichier kms_multi_region_keys.cpp dans le Kit SDK de chiffrement AWS pour C référentiel sur GitHub.

```
/* Encrypt with a multi-Region KMS key in us-east-1 */

/* Load error strings for debugging */
aws_cryptosdk_load_error_strings();

/* Initialize a multi-Region keyring */
const char *mrk_us_east_1 = "arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab";

struct aws_cryptosdk_keyring *mrk_keyring =
    Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder().Build(mrk_us_east_1);

/* Create a session; release the keyring */
struct aws_cryptosdk_session *session =
    aws_cryptosdk_session_new_from_keyring_2(aws_default_allocator(),
 AWS_CRYPTOSDK_ENCRYPT, mrk_keyring);

aws_cryptosdk_keyring_release(mrk_keyring);

/* Encrypt the data
 *   aws_cryptosdk_session_process_full is designed for non-streaming data
 */
aws_cryptosdk_session_process_full(
    session, ciphertext, ciphertext_buf_sz, &ciphertext_len, plaintext,
 plaintext_len));

/* Clean up the session */
aws_cryptosdk_session_destroy(session);
```

C# / .NET

Pour chiffrer avec une clé multirégionale dans la région USA Est (Virginie du Nord) (us-east-1), instanciez un `CreateAwsKmsMrkKeyringInput` objet avec un identifiant de clé pour la clé multirégionale et un client pour la région spécifiée. AWS KMS Utilisez ensuite la `CreateAwsKmsMrkKeyring()` méthode pour créer le porte-clés.

Le `CreateAwsKmsMrkKeyring()` procédé crée un trousseau de clés contenant exactement une clé multirégionale. Pour chiffrer avec plusieurs clés d'encapsulation, y compris une clé multirégionale, utilisez cette méthode. `CreateAwsKmsMrkMultiKeyring()`

Pour un exemple complet, consultez AwsKmsMrkKeyringExample.cs dans le référentiel AWS Encryption SDK for .NET sur GitHub.

```
//Encrypt with a multi-Region KMS key in us-east-1 Region

// Instantiate the AWS Encryption SDK and material providers
var encryptionSdk = AwsEncryptionSdkFactory.CreateDefaultAwsEncryptionSdk();
var materialProviders =

 AwsCryptographicMaterialProvidersFactory.CreateDefaultAwsCryptographicMaterialProviders();


// Multi-Region keys have a distinctive key ID that begins with 'mrk'
// Specify a multi-Region key in us-east-1
string mrkUSEast1 = "arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab";

// Create the keyring
// You can specify the Region or get the Region from the key ARN
var createMrkEncryptKeyringInput = new CreateAwsKmsMrkKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(RegionEndpoint.USEast1),
    KmsKeyId = mrkUSEast1
};
var mrkEncryptKeyring =
 materialProviders.CreateAwsKmsMrkKeyring(createMrkEncryptKeyringInput);

// Define the encryption context
var encryptionContext = new Dictionary<string, string>()
{
    {"purpose", "test"}
};

// Encrypt your plaintext data.
var encryptInput = new EncryptInput
{
    Plaintext = plaintext,
    Keyring = mrkEncryptKeyring,
    EncryptionContext = encryptionContext
};
var encryptOutput = encryptionSdk.Encrypt(encryptInput);
```

AWS Encryption CLI

Cet exemple chiffre le `hello.txt` fichier sous une clé multirégionale dans la région us-east-1. Étant donné que l'exemple spécifie un ARN clé avec un élément Region, il n'utilise pas l'attribut region du `--wrapping-keys` paramètre.

Lorsque l'ID de clé de la clé d'encapsulation ne spécifie pas de région, vous pouvez utiliser l'attribut de région `--wrapping-keys` pour spécifier la région, par exemple`--wrapping-keys key=$keyID region=us-east-1`.

```
# Encrypt with a multi-Region KMS key in us-east-1 Region

# To run this example, replace the fictitious key ARN with a valid value.
$ mrkUSEast1=arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab

$ aws-encryption-cli --encrypt \
                     --input hello.txt \
                     --wrapping-keys key=$mrkUSEast1 \
                     --metadata-output ~/metadata \
                     --encryption-context purpose=test \
                     --output .
```

Java

Pour chiffrer avec une clé multirégionale, instanciez une clé multirégionale `AwsKmsMrkAwareMasterKeyProvider` et spécifiez-la.

Pour un exemple complet, voir BasicMultiRegionKeyEncryptionExample.javale Kit SDK de chiffrement AWS pour Java référentiel sur GitHub.

```
//Encrypt with a multi-Region KMS key in us-east-1 Region

// Instantiate the client
final AwsCrypto crypto = AwsCrypto.builder()
    .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
    .build();

// Multi-Region keys have a distinctive key ID that begins with 'mrk'
// Specify a multi-Region key in us-east-1
final String mrkUSEast1 = "arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab";
```

```
// Instantiate an AWS KMS master key provider in strict mode for multi-Region keys
// Configure it to encrypt with the multi-Region key in us-east-1
final AwsKmsMrkAwareMasterKeyProvider kmsMrkProvider =
 AwsKmsMrkAwareMasterKeyProvider
    .builder()
    .buildStrict(mrkUSEast1);

// Create an encryption context
final Map<String, String> encryptionContext = Collections.singletonMap("Purpose",
 "Test");

// Encrypt your plaintext data
final CryptoResult<byte[], AwsKmsMrkAwareMasterKey> encryptResult =
 crypto.encryptData(
    kmsMrkProvider,
    encryptionContext,
    sourcePlaintext);
byte[] ciphertext = encryptResult.getResult();
```

JavaScript Browser

Pour chiffrer avec une clé multirégionale, utilisez la `buildAwsKmsMrkAwareStrictMultiKeyringBrowser()` méthode pour créer le trousseau de clés et spécifiez une clé multirégionale.

Pour un exemple complet, consultez kms_multi_region_simple.ts dans le référentiel sur. Kit SDK de chiffrement AWS pour JavaScript GitHub

```
/* Encrypt with a multi-Region KMS key in us-east-1 Region */

import {
  buildAwsKmsMrkAwareStrictMultiKeyringBrowser,
  buildClient,
  CommitmentPolicy,
  KMS,
} from '@aws-crypto/client-browser'


/* Instantiate an AWS Encryption SDK client */
const { encrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)
```

```
declare const credentials: {
  accessKeyId: string
  secretAccessKey: string
  sessionToken: string
}

/* Instantiate an AWS KMS client
 * The Kit SDK de chiffrement AWS pour JavaScript gets the Region from the key ARN
 */
const clientProvider = (region: string) => new KMS({ region, credentials })


/* Specify a multi-Region key in us-east-1 */
const multiRegionUsEastKey =
    'arn:aws:kms:us-east-1:111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab'

/* Instantiate the keyring */
const encryptKeyring = buildAwsKmsMrkAwareStrictMultiKeyringBrowser({
    generatorKeyId: multiRegionUsEastKey,
    clientProvider,
  })


/* Set the encryption context */
const context = {
    purpose: 'test',
  }

/* Test data to encrypt */
const cleartext = new Uint8Array([1, 2, 3, 4, 5])

/* Encrypt the data */
const { result } = await encrypt(encryptKeyring, cleartext, {
    encryptionContext: context,
  })
```

JavaScript Node.js

Pour chiffrer avec une clé multirégionale, utilisez la `buildAwsKmsMrkAwareStrictMultiKeyringNode()` méthode pour créer le trousseau de clés et spécifiez une clé multirégionale.

Pour un exemple complet, consultez kms_multi_region_simple.ts dans le référentiel sur. Kit SDK de chiffrement AWS pour JavaScript GitHub

```
//Encrypt with a multi-Region KMS key in us-east-1 Region

import { buildClient } from '@aws-crypto/client-node'

/* Instantiate the AWS Encryption SDK client
const { encrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

/* Test string to encrypt */
const cleartext = 'asdf'

/* Multi-Region keys have a distinctive key ID that begins with 'mrk'
 * Specify a multi-Region key in us-east-1
 */
const multiRegionUsEastKey =
    'arn:aws:kms:us-east-1:111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab'

/* Create an AWS KMS keyring */
const mrkEncryptKeyring = buildAwsKmsMrkAwareStrictMultiKeyringNode({
    generatorKeyId: multiRegionUsEastKey,
  })

/* Specify an encryption context */
const context = {
    purpose: 'test',
  }

/* Create an encryption keyring */
const { result } = await encrypt(mrkEncryptKeyring, cleartext, {
    encryptionContext: context,
  })
```

Python

Pour chiffrer avec une clé AWS KMS multirégionale, utilisez la `MRKAwareStrictAwsKmsMasterKeyProvider()` méthode et spécifiez une clé multirégionale.

Pour un exemple complet, consultez le fichier mrk_aware_kms_provider.py dans le Kit SDK de chiffrement AWS pour Python référentiel sur GitHub.

```
* Encrypt with a multi-Region KMS key in us-east-1 Region

# Instantiate the client
client =
 aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_R

# Specify a multi-Region key in us-east-1
mrk_us_east_1 = "arn:aws:kms:us-east-1:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab"

# Use the multi-Region method to create the master key provider
# in strict mode
strict_mrk_key_provider = MRKAwareStrictAwsKmsMasterKeyProvider(
        key_ids=[mrk_us_east_1]
)

# Set the encryption context
encryption_context = {
        "purpose": "test"
    }

# Encrypt your plaintext data
ciphertext, encrypt_header = client.encrypt(
        source=source_plaintext,
        encryption_context=encryption_context,
        key_provider=strict_mrk_key_provider
)
```

Ensuite, déplacez votre texte chiffré vers la `us-west-2` région. Il n'est pas nécessaire de rechiffrer le texte chiffré.

Pour déchiffrer le texte chiffré en mode strict dans la `us-west-2` région, instanciez le symbole compatible avec plusieurs régions avec le code ARN de la clé multirégionale associée dans la région. `us-west-2` Si vous spécifiez l'ARN d'une clé multirégionale associée dans une région différente (y compris `us-east-1` celle où elle a été chiffrée), le symbole prenant en compte plusieurs régions émettra un appel interrégional pour cela. AWS KMS key

Lors du déchiffrement en mode strict, le symbole prenant en compte plusieurs régions nécessite un ARN clé. Il n'accepte qu'un seul ARN de clé pour chaque ensemble de clés multirégionales associées.

Avant d'exécuter ces exemples, remplacez l'exemple d'ARN clé multirégional par une valeur valide provenant de votreCompte AWS.

C

Pour déchiffrer en mode strict avec une clé multirégionale, utilisez la `Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder()` méthode d'instanciation du trousseau de clés. Spécifiez la clé multirégionale associée dans la région locale (us-west-2).

Pour un exemple complet, consultez le fichier kms_multi_region_keys.cpp dans le Kit SDK de chiffrement AWS pour C référentiel sur GitHub.

```
/* Decrypt with a related multi-Region KMS key in us-west-2 Region */

/* Load error strings for debugging */
aws_cryptosdk_load_error_strings();

/* Initialize a multi-Region keyring */
const char *mrk_us_west_2 = "arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab";

struct aws_cryptosdk_keyring *mrk_keyring =
    Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder().Build(mrk_us_west_2);

/* Create a session; release the keyring */
struct aws_cryptosdk_session *session =
    aws_cryptosdk_session_new_from_keyring_2(aws_default_allocator(),
 AWS_CRYPTOSDK_ENCRYPT, mrk_keyring);

aws_cryptosdk_session_set_commitment_policy(session,
    COMMITMENT_POLICY_REQUIRE_ENCRYPT_REQUIRE_DECRYPT);

aws_cryptosdk_keyring_release(mrk_keyring);

/* Decrypt the ciphertext
 *   aws_cryptosdk_session_process_full is designed for non-streaming data
 */
aws_cryptosdk_session_process_full(
    session, plaintext, plaintext_buf_sz, &plaintext_len, ciphertext,
 ciphertext_len));

/* Clean up the session */
aws_cryptosdk_session_destroy(session);
```

C# / .NET

Pour déchiffrer en mode strict avec une seule clé multirégionale, utilisez les mêmes constructeurs et méthodes que ceux que vous avez utilisés pour assembler l'entrée et créer le trousseau de clés à chiffrer. Instanciez un `CreateAwsKmsMrkKeyringInput` objet avec l'ARN clé d'une clé multirégionale associée et un AWS KMS client pour la région USA Ouest (Oregon) (us-west-2). Utilisez ensuite la `CreateAwsKmsMrkKeyring()` méthode pour créer un trousseau de clés multirégions avec une clé KMS multirégionale.

Pour un exemple complet, consultez AwsKmsMrkKeyringExample.cs dans le référentiel AWS Encryption SDK for .NET sur GitHub.

```
// Decrypt with a related multi-Region KMS key in us-west-2 Region

// Instantiate the AWS Encryption SDK and material providers
var encryptionSdk = AwsEncryptionSdkFactory.CreateDefaultAwsEncryptionSdk();
var materialProviders =

 AwsCryptographicMaterialProvidersFactory.CreateDefaultAwsCryptographicMaterialProviders();

// Specify the key ARN of the multi-Region key in us-west-2
string mrkUSWest2 = "arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab";

// Instantiate the keyring input
// You can specify the Region or get the Region from the key ARN
var createMrkDecryptKeyringInput = new CreateAwsKmsMrkKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(RegionEndpoint.USWest2),
    KmsKeyId = mrkUSWest2
};

// Create the multi-Region keyring
var mrkDecryptKeyring =
 materialProviders.CreateAwsKmsMrkKeyring(createMrkDecryptKeyringInput);

// Decrypt the ciphertext
var decryptInput = new DecryptInput
{
    Ciphertext = ciphertext,
    Keyring = mrkDecryptKeyring
};
```

```
var decryptOutput = encryptionSdk.Decrypt(decryptInput);
```

AWS Encryption CLI

Pour déchiffrer avec la clé multirégionale associée dans la région us-west-2, utilisez l'attribut key du `--wrapping-keys` paramètre pour spécifier son ARN clé.

```
# Decrypt with a related multi-Region KMS key in us-west-2 Region

# To run this example, replace the fictitious key ARN with a valid value.
$ mrkUSWest2=arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab

$ aws-encryption-cli --decrypt \
                     --input hello.txt.encrypted \
                     --wrapping-keys key=$mrkUSWest2 \
                     --commitment-policy require-encrypt-require-decrypt \
                     --encryption-context purpose=test \
                     --metadata-output ~/metadata \
                     --max-encrypted-data-keys 1 \
                     --buffer \
                     --output .
```

Java

Pour déchiffrer en mode strict, instanciez une clé multirégionale `AwsKmsMrkAwareMasterKeyProvider` et spécifiez-la dans la région locale (us-west-2).

Pour un exemple complet, voir BasicMultiRegionKeyEncryptionExample.java dans le Kit SDK de chiffrement AWS pour Java référentiel sur GitHub.

```
// Decrypt with a related multi-Region KMS key in us-west-2 Region

// Instantiate the client
final AwsCrypto crypto = AwsCrypto.builder()
    .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
    .build();

// Related multi-Region keys have the same key ID. Their key ARNs differs only in
 the Region field.
String mrkUSWest2 = "arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab";
```

```
// Use the multi-Region method to create the master key provider
// in strict mode
AwsKmsMrkAwareMasterKeyProvider kmsMrkProvider =
 AwsKmsMrkAwareMasterKeyProvider.builder()
    .buildStrict(mrkUSWest2);

// Decrypt your ciphertext
CryptoResult<byte[], AwsKmsMrkAwareMasterKey> decryptResult = crypto.decryptData(
        kmsMrkProvider,
        ciphertext);
byte[] decrypted = decryptResult.getResult();
```

JavaScript Browser

Pour déchiffrer en mode strict, utilisez la `buildAwsKmsMrkAwareStrictMultiKeyringBrowser()` méthode pour créer le trousseau de clés et spécifiez la clé multirégionale associée dans la région locale (us-west-2).

Pour un exemple complet, consultez kms_multi_region_simple.ts dans le référentiel sur. Kit SDK de chiffrement AWS pour JavaScript GitHub

```
/* Decrypt with a related multi-Region KMS key in us-west-2 Region */

import {
  buildAwsKmsMrkAwareStrictMultiKeyringBrowser,
  buildClient,
  CommitmentPolicy,
  KMS,
} from '@aws-crypto/client-browser'


/* Instantiate an AWS Encryption SDK client */
const { decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

declare const credentials: {
  accessKeyId: string
  secretAccessKey: string
  sessionToken: string
}

/* Instantiate an AWS KMS client
```

```
 * The Kit SDK de chiffrement AWS pour JavaScript gets the Region from the key ARN
 */
const clientProvider = (region: string) => new KMS({ region, credentials })


/* Specify a multi-Region key in us-west-2 */
const multiRegionUsWestKey =
    'arn:aws:kms:us-west-2:111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab'

/* Instantiate the keyring */
const mrkDecryptKeyring = buildAwsKmsMrkAwareStrictMultiKeyringBrowser({
    generatorKeyId: multiRegionUsWestKey,
    clientProvider,
  })


/* Decrypt the data */
const { plaintext, messageHeader } = await decrypt(mrkDecryptKeyring, result)
```

JavaScript Node.js

Pour déchiffrer en mode strict, utilisez la `buildAwsKmsMrkAwareStrictMultiKeyringNode()` méthode pour créer le trousseau de clés et spécifiez la clé multirégionale associée dans la région locale (us-west-2).

Pour un exemple complet, consultez kms_multi_region_simple.ts dans le référentiel sur. Kit SDK de chiffrement AWS pour JavaScript GitHub

```
/* Decrypt with a related multi-Region KMS key in us-west-2 Region */

import { buildClient } from '@aws-crypto/client-node'

/* Instantiate the client
const { decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

/* Multi-Region keys have a distinctive key ID that begins with 'mrk'
 * Specify a multi-Region key in us-east-1
 */
const multiRegionUsWestKey =
    'arn:aws:kms:us-west-2:111122223333:key/mrk-1234abcd12ab34cd56ef1234567890ab'
```

```
/* Create an AWS KMS keyring */
const mrkDecryptKeyring = buildAwsKmsMrkAwareStrictMultiKeyringNode({
    generatorKeyId: multiRegionUsWestKey,
  })

/* Decrypt your ciphertext */
const { plaintext, messageHeader } = await decrypt(decryptKeyring, result)
```

Python

Pour déchiffrer en mode strict, utilisez la `MRKAwareStrictAwsKmsMasterKeyProvider()` méthode de création du fournisseur de clé principale. Spécifiez la clé multirégionale associée dans la région locale (us-west-2).

Pour un exemple complet, consultez le fichier mrk_aware_kms_provider.py dans le Kit SDK de chiffrement AWS pour Python référentiel sur GitHub.

```
# Decrypt with a related multi-Region KMS key in us-west-2 Region

# Instantiate the client
client =
 aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_R

# Related multi-Region keys have the same key ID. Their key ARNs differs only in the
 Region field
mrk_us_west_2 = "arn:aws:kms:us-west-2:111122223333:key/
mrk-1234abcd12ab34cd56ef1234567890ab"

# Use the multi-Region method to create the master key provider
# in strict mode
strict_mrk_key_provider = MRKAwareStrictAwsKmsMasterKeyProvider(
        key_ids=[mrk_us_west_2]
)

# Decrypt your ciphertext
plaintext, _ = client.decrypt(
        source=ciphertext,
        key_provider=strict_mrk_key_provider
)
```

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, consultezUtilisation d'un porte-clés AWS KMS Discovery.)

Si vous avez chiffré avec une clé multirégionale, le symbole compatible avec plusieurs régions en mode découverte essaiera de déchiffrer en utilisant une clé multirégionale associée dans la région locale. S'il n'en existe aucun, l'appel échoue. En mode découverte, aucun appel interrégional ne AWS Encryption SDK sera lancé pour la clé multirégionale utilisée pour le chiffrement.

> **Note**
>
> Si vous utilisez un symbole prenant en compte plusieurs régions en mode découverte pour chiffrer des données, l'opération de chiffrement échoue.

L'exemple suivant montre comment déchiffrer avec le symbole compatible avec plusieurs régions en mode découverte. Comme vous ne spécifiez pas deAWS KMS key, vous AWS Encryption SDK devez obtenir la région à partir d'une autre source. Dans la mesure du possible, spécifiez explicitement la région locale. Sinon, AWS Encryption SDK il obtient la région locale à partir de la région configurée dans le AWS SDK pour votre langage de programmation.

Avant d'exécuter ces exemples, remplacez l'exemple d'ID de compte et l'ARN de la clé multirégionale par des valeurs valides provenant de votreCompte AWS.

C

Pour déchiffrer en mode découverte avec une clé multirégionale, utilisez la `Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder()` méthode de création du trousseau de clés et la `Aws::Cryptosdk::KmsKeyring::DiscoveryFilter::Builder()` méthode de création du filtre de découverte. Pour spécifier la région locale, définissez a `ClientConfiguration` et spécifiez-la dans le AWS KMS client.

Pour un exemple complet, consultez le fichier kms_multi_region_keys.cpp dans le Kit SDK de chiffrement AWS pour C référentiel sur GitHub.

```
/* Decrypt in discovery mode with a multi-Region KMS key */

/* Load error strings for debugging */
aws_cryptosdk_load_error_strings();
```

```
/* Construct a discovery filter for the account and partition. The
 *  filter is optional, but it's a best practice that we recommend.
 */
const char *account_id = "111122223333";
const char *partition = "aws";
const std::shared_ptr<Aws::Cryptosdk::KmsKeyring::DiscoveryFilter> discovery_filter
 =

 Aws::Cryptosdk::KmsKeyring::DiscoveryFilter::Builder(partition).AddAccount(account_id).Buil

/* Create an AWS KMS client in the desired region. */
const char *region = "us-west-2";

Aws::Client::ClientConfiguration client_config;
client_config.region = region;
const std::shared_ptr<Aws::KMS::KMSClient> kms_client =
    Aws::MakeShared<Aws::KMS::KMSClient>("AWS_SAMPLE_CODE", client_config);

struct aws_cryptosdk_keyring *mrk_keyring =
    Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder()
        .WithKmsClient(kms_client)
        .BuildDiscovery(region, discovery_filter);

/* Create a session; release the keyring */
struct aws_cryptosdk_session *session =
    aws_cryptosdk_session_new_from_keyring_2(aws_default_allocator(),
 AWS_CRYPTOSDK_DECRYPT, mrk_keyring);

aws_cryptosdk_keyring_release(mrk_keyring);
commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
/* Decrypt the ciphertext
 *   aws_cryptosdk_session_process_full is designed for non-streaming data
 */
aws_cryptosdk_session_process_full(
    session, plaintext, plaintext_buf_sz, &plaintext_len, ciphertext,
 ciphertext_len));

/* Clean up the session */
aws_cryptosdk_session_destroy(session);
```

C# / .NET

Pour créer un jeu de clés de découverte prenant en compte plusieurs régions dans .NET, instanciez un `CreateAwsKmsMrkDiscoveryKeyringInput` objet qui prend un AWS KMS

client pour une destination spécifique et un filtre de découverte facultatif qui limite les clés KMS à une partition et à un compte particuliersRégion AWS. AWS Encryption SDK AWS Appelez ensuite la `CreateAwsKmsMrkDiscoveryKeyring()` méthode avec l'objet d'entrée. Pour un exemple complet, consultez AwsKmsMrkDiscoveryKeyringExample.cs dans le référentiel AWS Encryption SDK for .NET sur GitHub.

Pour créer un trousseau de clés de découverte compatible avec plusieurs régions pour plusieursRégion AWS, utilisez la `CreateAwsKmsMrkDiscoveryMultiKeyring()` méthode pour créer un trousseau de clés multirégional, ou utilisez-la pour créer plusieurs trousseaux de découverte compatibles avec plusieurs régions, puis utilisez-la `CreateAwsKmsMrkDiscoveryKeyring()` pour les combiner dans un trousseau de clés multirégional. `CreateMultiKeyring()`

Pour un exemple, consultez AwsKmsMrkDiscoveryMultiKeyringExample.cs.

```
// Decrypt in discovery mode with a multi-Region KMS key

// Instantiate the AWS Encryption SDK and material providers
var encryptionSdk = AwsEncryptionSdkFactory.CreateDefaultAwsEncryptionSdk();
var materialProviders =

 AwsCryptographicMaterialProvidersFactory.CreateDefaultAwsCryptographicMaterialProviders();

List<string> account = new List<string> { "111122223333" };

// Instantiate the discovery filter
DiscoveryFilter mrkDiscoveryFilter = new DiscoveryFilter()
{
    AccountIds = account,
    Partition = "aws"
}

// Create the keyring
var createMrkDiscoveryKeyringInput = new CreateAwsKmsMrkDiscoveryKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(RegionEndpoint.USWest2),
    DiscoveryFilter = mrkDiscoveryFilter
};
var mrkDiscoveryKeyring =
 materialProviders.CreateAwsKmsMrkDiscoveryKeyring(createMrkDiscoveryKeyringInput);
```

```
// Decrypt the ciphertext
var decryptInput = new DecryptInput
{
    Ciphertext = ciphertext,
    Keyring = mrkDiscoveryKeyring
};
var decryptOutput = encryptionSdk.Decrypt(decryptInput);
```

AWS Encryption CLI

Pour déchiffrer en mode découverte, utilisez l'attribut de découverte du `--wrapping-keys` paramètre. Les attributs discovery-account et discovery-partition créent un filtre de découverte facultatif mais recommandé.

Pour spécifier la région, cette commande inclut l'attribut de région du `--wrapping-keys` paramètre.

```
# Decrypt in discovery mode with a multi-Region KMS key

$ aws-encryption-cli --decrypt \
                     --input hello.txt.encrypted \
                     --wrapping-keys discovery=true \
                                     discovery-account=111122223333 \
                                     discovery-partition=aws \
                                     region=us-west-2 \
                     --encryption-context purpose=test \
                     --metadata-output ~/metadata \
                     --max-encrypted-data-keys 1 \
                     --buffer \
                     --output .
```

Java

Pour spécifier la région locale, utilisez le `builder().withDiscoveryMrkRegion` paramètre. Sinon, AWS Encryption SDK il obtient la région locale à partir de la région configurée dans le AWS SDK for Java.

Pour un exemple complet, voir DiscoveryMultiRegionDecryptionExample.java dans le Kit SDK de chiffrement AWS pour Java référentiel sur GitHub.

```
// Decrypt in discovery mode with a multi-Region KMS key

// Instantiate the client
```

```
final AwsCrypto crypto = AwsCrypto.builder()
    .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
    .build();

DiscoveryFilter discoveryFilter = new DiscoveryFilter("aws", 111122223333);

AwsKmsMrkAwareMasterKeyProvider mrkDiscoveryProvider =
 AwsKmsMrkAwareMasterKeyProvider
    .builder()
    .withDiscoveryMrkRegion(Region.US_WEST_2)
    .buildDiscovery(discoveryFilter);

// Decrypt your ciphertext
final CryptoResult<byte[], AwsKmsMrkAwareMasterKey> decryptResult = crypto
    .decryptData(mrkDiscoveryProvider, ciphertext);
```

JavaScript Browser

Pour déchiffrer en mode découverte avec une clé multirégionale symétrique, utilisez la méthode. `AwsKmsMrkAwareSymmetricDiscoveryKeyringBrowser()`

Pour un exemple complet, consultez kms_multi_region_discovery.ts dans le référentiel sur. Kit SDK de chiffrement AWS pour JavaScript GitHub

```
/* Decrypt in discovery mode with a multi-Region KMS key */

import {
  AwsKmsMrkAwareSymmetricDiscoveryKeyringBrowser,
  buildClient,
  CommitmentPolicy,
  KMS,
} from '@aws-crypto/client-browser'


/* Instantiate an AWS Encryption SDK client */
const { decrypt } = buildClient()

declare const credentials: {
  accessKeyId: string
  secretAccessKey: string
  sessionToken: string
}
```

```
/* Instantiate the KMS client with an explicit Region */
const client = new KMS({ region: 'us-west-2', credentials })


/* Create a discovery filter */
const discoveryFilter = { partition: 'aws', accountIDs: ['111122223333'] }


/* Create an AWS KMS discovery keyring */
const mrkDiscoveryKeyring = new AwsKmsMrkAwareSymmetricDiscoveryKeyringBrowser({
    client,
    discoveryFilter,
  })


/* Decrypt the data */
const { plaintext, messageHeader } = await decrypt(mrkDiscoveryKeyring, ciphertext)
```

JavaScript Node.js

Pour déchiffrer en mode découverte avec une clé multirégionale symétrique, utilisez la méthode. `AwsKmsMrkAwareSymmetricDiscoveryKeyringNode()`

Pour un exemple complet, consultez kms_multi_region_discovery.ts dans le référentiel sur. Kit SDK de chiffrement AWS pour JavaScript GitHub

```
/* Decrypt in discovery mode with a multi-Region KMS key */

import {
  AwsKmsMrkAwareSymmetricDiscoveryKeyringNode,
  buildClient,
  CommitmentPolicy,
  KMS,
} from '@aws-crypto/client-node'

/* Instantiate the Encryption SDK client
const { decrypt } = buildClient()

/* Instantiate the KMS client with an explicit Region */
const client = new KMS({ region: 'us-west-2' })

/* Create a discovery filter */
const discoveryFilter = { partition: 'aws', accountIDs: ['111122223333'] }
```

```
/* Create an AWS KMS discovery keyring */
const mrkDiscoveryKeyring = new AwsKmsMrkAwareSymmetricDiscoveryKeyringNode({
    client,
    discoveryFilter,
  })

/* Decrypt your ciphertext */
const { plaintext, messageHeader } = await decrypt(mrkDiscoveryKeyring, result)
```

Python

Pour déchiffrer en mode découverte avec une clé multirégionale, utilisez la `MRKAwareDiscoveryAwsKmsMasterKeyProvider()` méthode.

Pour un exemple complet, consultez le fichier mrk_aware_kms_provider.py dans le Kit SDK de chiffrement AWS pour Python référentiel sur GitHub.

```
# Decrypt in discovery mode with a multi-Region KMS key

# Instantiate the client
client = aws_encryption_sdk.EncryptionSDKClient()

# Create the discovery filter and specify the region
decrypt_kwargs = dict(
        discovery_filter=DiscoveryFilter(account_ids="111122223333",
 partition="aws"),
        discovery_region="us-west-2",
    )

# Use the multi-Region method to create the master key provider
# in discovery mode
mrk_discovery_key_provider =
 MRKAwareDiscoveryAwsKmsMasterKeyProvider(**decrypt_kwargs)

# Decrypt your ciphertext
plaintext, _ = client.decrypt(
        source=ciphertext,
        key_provider=mrk_discovery_key_provider
)
```

Choix d'une suite d'algorithmes

Il AWS Encryption SDK prend en charge plusieurs algorithmes de chiffrement symétriques et asymétriques pour chiffrer vos clés de données sous les clés d'encapsulation que vous spécifiez. Toutefois, lorsqu'il utilise ces clés de données pour chiffrer vos données, il utilise AWS Encryption SDK par défaut une suite d'algorithmes recommandée qui utilise l'algorithme AES-GCM avec dérivation de clés, signatures numériques et engagement des 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 AWS Encryption SDK en charge, consultezSuites d'algorithmes prises en charge dans le kit AWS Encryption SDK.

Les exemples suivants vous montrent comment sélectionner une autre suite d'algorithmes lors du chiffrement. Ces exemples sélectionnent une suite d'algorithmes AES-GCM recommandée avec dérivation de clés et engagement de clés, mais sans signatures numériques. Lorsque vous chiffrez avec une suite d'algorithmes qui n'inclut pas de signatures numériques, utilisez le mode de déchiffrement uniquement non signé lors du déchiffrement. Ce mode, qui échoue en cas de détection d'un texte chiffré signé, est particulièrement utile lors du déchiffrement en continu.

C

Pour spécifier une autre suite d'algorithmes dans leKit SDK de chiffrement AWS pour C, vous devez créer un CMM de manière explicite. Utilisez ensuite le `aws_cryptosdk_default_cmm_set_alg_id` avec le CMM et la suite d'algorithmes sélectionnée.

```
/* Specify an algorithm suite without signing */

/* Load error strings for debugging */
aws_cryptosdk_load_error_strings();

/* Construct an AWS KMS keyring */
struct aws_cryptosdk_keyring *kms_keyring =
 Aws::Cryptosdk::KmsKeyring::Builder().Build(key_arn);

/* To set an alternate algorithm suite, create an cryptographic
   materials manager (CMM) explicitly
 */
```

```
struct aws_cryptosdk_cmm *cmm =
 aws_cryptosdk_default_cmm_new(aws_default_allocator(), kms_keyring);
aws_cryptosdk_keyring_release(kms_keyring);

/* Specify the algorithm suite for the CMM */
aws_cryptosdk_default_cmm_set_alg_id(cmm, ALG_AES256_GCM_HKDF_SHA512_COMMIT_KEY);

/* Construct the session with the CMM,
   then release the CMM reference
 */
struct aws_cryptosdk_session *session = aws_cryptosdk_session_new_from_cmm_2(alloc,
 AWS_CRYPTOSDK_ENCRYPT, cmm);
aws_cryptosdk_cmm_release(cmm);

/* Encrypt the data
   Use aws_cryptosdk_session_process_full with non-streaming data
 */
if (AWS_OP_SUCCESS != aws_cryptosdk_session_process_full(
                          session,
                          ciphertext,
                          ciphertext_buf_sz,
                          &ciphertext_len,
                          plaintext,
                          plaintext_len)) {
    aws_cryptosdk_session_destroy(session);
    return AWS_OP_ERR;
}
```

Lorsque vous déchiffrez des données chiffrées sans signature numérique, utilisez. `AWS_CRYPTOSDK_DECRYPT_UNSIGNED` Cela entraîne l'échec du déchiffrement s'il rencontre un texte chiffré signé.

```
/* Decrypt unsigned streaming data */

/* Load error strings for debugging */
aws_cryptosdk_load_error_strings();

/* Construct an AWS KMS keyring */
struct aws_cryptosdk_keyring *kms_keyring =
 Aws::Cryptosdk::KmsKeyring::Builder().Build(key_arn);

/* Create a session for decrypting with the AWS KMS keyring
   Then release the keyring reference
```

```
 */
struct aws_cryptosdk_session *session =

 aws_cryptosdk_session_new_from_keyring_2(alloc, AWS_CRYPTOSDK_DECRYPT_UNSIGNED,
 kms_keyring);
aws_cryptosdk_keyring_release(kms_keyring);

if (!session) {
    return AWS_OP_ERR;
}

/* Limit encrypted data keys */
aws_cryptosdk_session_set_max_encrypted_data_keys(session, 1);

/* Decrypt
   Use aws_cryptosdk_session_process_full with non-streaming data
 */
    if (AWS_OP_SUCCESS != aws_cryptosdk_session_process_full(
                          session,
                          plaintext,
                          plaintext_buf_sz,
                          &plaintext_len,
                          ciphertext,
                          ciphertext_len)) {
    aws_cryptosdk_session_destroy(session);
    return AWS_OP_ERR;
}
```

C# / .NET

Pour spécifier une autre suite d'algorithmes dans AWS Encryption SDK .NET, spécifiez la `AlgorithmSuiteId` propriété d'un EncryptInputobjet. Le AWS Encryption SDK for .NET inclut des constantes que vous pouvez utiliser pour identifier votre suite d'algorithmes préférée.

AWS Encryption SDKfor .NET ne dispose pas de méthode permettant de détecter le texte chiffré signé lors du déchiffrement en continu, car cette bibliothèque ne prend pas en charge le streaming de données.

```
// Specify an algorithm suite without signing

// Instantiate the AWS Encryption SDK and material providers
var encryptionSdk = AwsEncryptionSdkFactory.CreateDefaultAwsEncryptionSdk();
var materialProviders =
```

```
 AwsCryptographicMaterialProvidersFactory.CreateDefaultAwsCryptographicMaterialProviders();

// Create the keyring
var keyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = keyArn
};
var keyring = materialProviders.CreateAwsKmsKeyring(keyringInput);

// Encrypt your plaintext data
var encryptInput = new EncryptInput
{
    Plaintext = plaintext,
    Keyring = keyring,
    AlgorithmSuiteId = AlgorithmSuiteId.ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY
};
var encryptOutput = encryptionSdk.Encrypt(encryptInput);
```

AWS Encryption CLI

Lors du chiffrement du `hello.txt` fichier, cet exemple utilise le `--algorithm` paramètre pour spécifier une suite d'algorithmes sans signatures numériques.

```
# Specify an algorithm suite without signing

# To run this example, replace the fictitious key ARN with a valid value.
$ keyArn=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

$ aws-encryption-cli --encrypt \
                     --input hello.txt \
                     --wrapping-keys key=$keyArn \
                     --algorithm AES_256_GCM_HKDF_SHA512_COMMIT_KEY \
                     --metadata-output ~/metadata \
                     --encryption-context purpose=test \
                     --commitment-policy require-encrypt-require-decrypt \
                     --output hello.txt.encrypted \
                     --decode
```

Lors du déchiffrement, cet exemple utilise le paramètre. `--decrypt-unsigned` Ce paramètre est recommandé pour garantir que vous déchiffrez du texte chiffré non signé, en particulier avec la CLI, qui diffuse toujours les entrées et les sorties.

```
# Decrypt unsigned streaming data

# To run this example, replace the fictitious key ARN with a valid value.
$ keyArn=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

$ aws-encryption-cli --decrypt-unsigned \
                     --input hello.txt.encrypted \
                     --wrapping-keys key=$keyArn \
                     --max-encrypted-data-keys 1 \
                     --commitment-policy require-encrypt-require-decrypt \
                     --encryption-context purpose=test \
                     --metadata-output ~/metadata \
                     --output .
```

Java

Pour spécifier une autre suite d'algorithmes, utilisez la `AwsCrypto.builder().withEncryptionAlgorithm()` méthode. Cet exemple indique une suite d'algorithmes alternative sans signatures numériques.

```
// Specify an algorithm suite without signing

// Instantiate the client
AwsCrypto crypto = AwsCrypto.builder()
    .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
    .withEncryptionAlgorithm(CryptoAlgorithm.ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY)
    .build();

String awsKmsKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

// Create a master key provider in strict mode
KmsMasterKeyProvider masterKeyProvider = KmsMasterKeyProvider.builder()
    .buildStrict(awsKmsKey);

// Create an encryption context to identify this ciphertext
        Map<String, String> encryptionContext = Collections.singletonMap("Example",
 "FileStreaming");

// Encrypt your plaintext data
CryptoResult<byte[], KmsMasterKey> encryptResult = crypto.encryptData(
    masterKeyProvider,
    sourcePlaintext,
```

```
    encryptionContext);
byte[] ciphertext = encryptResult.getResult();
```

Lorsque vous diffusez des données à des fins de déchiffrement, utilisez cette `createUnsignedMessageDecryptingStream()` méthode pour vous assurer que tout le texte chiffré que vous décryptez n'est pas signé.

```
// Decrypt unsigned streaming data

// Instantiate the client
AwsCrypto crypto = AwsCrypto.builder()
  .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
  .withMaxEncryptedDataKeys(1)
  .build();

// Create a master key provider in strict mode
String awsKmsKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
KmsMasterKeyProvider masterKeyProvider = KmsMasterKeyProvider.builder()
  .buildStrict(awsKmsKey);

// Decrypt the encrypted message
FileInputStream in = new FileInputStream(srcFile + ".encrypted");
CryptoInputStream<KmsMasterKey> decryptingStream =
 crypto.createUnsignedMessageDecryptingStream(masterKeyProvider, in);

// Return the plaintext data
// Write the plaintext data to disk
FileOutputStream out = new FileOutputStream(srcFile + ".decrypted");
IOUtils.copy(decryptingStream, out);
decryptingStream.close();
```

JavaScript Browser

Pour spécifier une autre suite d'algorithmes, utilisez le `suiteId` paramètre avec une valeur `AlgorithmSuiteIdentifier` enum.

```
// Specify an algorithm suite without signing

// Instantiate the client
const { encrypt } = buildClient( CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT )
```

```
// Specify a KMS key
const generatorKeyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

// Create a keyring with the KMS key
const keyring = new KmsKeyringBrowser({ generatorKeyId })

// Encrypt your plaintext data
const { result } = await encrypt(keyring, cleartext, { suiteId:
 AlgorithmSuiteIdentifier.ALG_AES256_GCM_IV12_TAG16_HKDF_SHA512_COMMIT_KEY,
 encryptionContext: context, })
```

Lors du déchiffrement, utilisez la méthode standard. `decrypt` Kit SDK de chiffrement AWS pour JavaScriptdans le navigateur n'a pas de `decrypt-unsigned` mode car le navigateur ne prend pas en charge le streaming.

```
// Decrypt unsigned streaming data

// Instantiate the client
const { decrypt } = buildClient( CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT )

// Create a keyring with the same KMS key used to encrypt
const generatorKeyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
const keyring = new KmsKeyringBrowser({ generatorKeyId })

// Decrypt the encrypted message
const { plaintext, messageHeader } = await decrypt(keyring, ciphertextMessage)
```

JavaScript Node.js

Pour spécifier une autre suite d'algorithmes, utilisez le `suiteId` paramètre avec une valeur `AlgorithmSuiteIdentifier` enum.

```
// Specify an algorithm suite without signing

// Instantiate the client
const { encrypt } = buildClient( CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT )

// Specify a KMS key
const generatorKeyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
```

```
// Create a keyring with the KMS key
const keyring = new KmsKeyringNode({ generatorKeyId })

// Encrypt your plaintext data
const { result } = await encrypt(keyring, cleartext, { suiteId:
 AlgorithmSuiteIdentifier.ALG_AES256_GCM_IV12_TAG16_HKDF_SHA512_COMMIT_KEY,
 encryptionContext: context, })
```

Lorsque vous déchiffrez des données chiffrées sans signature numérique, utilisez Stream. decryptUnsignedMessage Cette méthode échoue si elle détecte un texte chiffré signé.

```
// Decrypt unsigned streaming data

// Instantiate the client
const { decryptUnsignedMessageStream } =
 buildClient( CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT )

// Create a keyring with the same KMS key used to encrypt
const generatorKeyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
const keyring = new KmsKeyringNode({ generatorKeyId })

// Decrypt the encrypted message
const outputStream =
 createReadStream(filename) .pipe(decryptUnsignedMessageStream(keyring))
```

Python

Pour spécifier un autre algorithme de chiffrement, utilisez le `algorithm` paramètre avec une valeur `Algorithm` enum.

```
# Specify an algorithm suite without signing

# Instantiate a client
client =
 aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_R
                                        max_encrypted_data_keys=1)

# Create a master key provider in strict mode
aws_kms_key = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
```

```
aws_kms_strict_master_key_provider = StrictAwsKmsMasterKeyProvider(
        key_ids=[aws_kms_key]
)

# Encrypt the plaintext using an alternate algorithm suite
ciphertext, encrypted_message_header = client.encrypt(
    algorithm=Algorithm.AES_256_GCM_HKDF_SHA512_COMMIT_KEY, source=source_plaintext,
 key_provider=kms_key_provider
)
```

Lorsque vous déchiffrez des messages chiffrés sans signature numérique, utilisez le mode `decrypt-unsigned` streaming, en particulier lors du déchiffrement en streaming.

```
# Decrypt unsigned streaming data

# Instantiate the client
client =
 aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_R
                                                    max_encrypted_data_keys=1)

# Create a master key provider in strict mode
aws_kms_key = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
aws_kms_strict_master_key_provider = StrictAwsKmsMasterKeyProvider(
        key_ids=[aws_kms_key]
)

# Decrypt with decrypt-unsigned
with open(ciphertext_filename, "rb") as ciphertext, open(cycled_plaintext_filename,
 "wb") as plaintext:
    with client.stream(mode="decrypt-unsigned",
                       source=ciphertext,
                       key_provider=master_key_provider) as decryptor:
        for chunk in decryptor:
            plaintext.write(chunk)

# Verify that the encryption context
assert all(
   pair in decryptor.header.encryption_context.items() for pair in
 encryptor.header.encryption_context.items()
)
return ciphertext_filename, cycled_plaintext_filename
```

Limiter les clés de données chiffrées

Vous pouvez limiter le nombre de clés de données chiffrées dans un message chiffré. Cette fonctionnalité des meilleures pratiques peut vous aider à détecter un jeu de clés mal configuré lors du chiffrement ou un texte chiffré malveillant lors du déchiffrement. Cela évite également les appels inutiles, coûteux et potentiellement exhaustifs vers votre infrastructure clé. La limitation des clés de données chiffrées est particulièrement utile lorsque vous décryptez des messages provenant d'une source non fiable.

Bien que la plupart des messages chiffrés comportent une clé de données chiffrée pour chaque clé d'encapsulation utilisée dans le chiffrement, un message chiffré peut contenir jusqu'à 65 535 clés de données chiffrées. Un acteur malveillant peut créer un message chiffré avec des milliers de clés de données chiffrées, dont aucune ne peut être déchiffrée. Par conséquent, ils AWS Encryption SDK essaieraient de déchiffrer chaque clé de données chiffrée jusqu'à épuisement des clés de données chiffrées contenues dans le message.

Pour limiter les clés de données chiffrées, utilisez le `MaxEncryptedDataKeys` paramètre. Ce paramètre est disponible pour tous les langages de programmation pris en charge à partir de la version 1.9. x et 2.2. x duAWS Encryption SDK. Il est facultatif et valide lors du chiffrement et du déchiffrement. Les exemples suivants déchiffrent des données chiffrées sous trois clés d'encapsulation différentes. La `MaxEncryptedDataKeys` valeur est définie sur 3.

C

```
/* Load error strings for debugging */
aws_cryptosdk_load_error_strings();

/* Construct an AWS KMS keyring */
struct aws_cryptosdk_keyring *kms_keyring =
      Aws::Cryptosdk::KmsKeyring::Builder().Build(key_arn1, { key_arn2, key_arn3 });

/* Create a session */
struct aws_cryptosdk_session *session =
    aws_cryptosdk_session_new_from_keyring_2(alloc, AWS_CRYPTOSDK_DECRYPT,
 kms_keyring);
aws_cryptosdk_keyring_release(kms_keyring);

/* Limit encrypted data keys */
aws_cryptosdk_session_set_max_encrypted_data_keys(session, 3);
```

```
/* Decrypt */
size_t ciphertext_consumed_output;
aws_cryptosdk_session_process(session,
    plaintext_output,
    plaintext_buf_sz_output,
    &plaintext_len_output,
    ciphertext_input,
    ciphertext_len_input,
    &ciphertext_consumed_output);
assert(aws_cryptosdk_session_is_done(session));
assert(ciphertext_consumed == ciphertext_len);
```

C# / .NET

Pour limiter les clés de données chiffrées dans le AWS Encryption SDK pour .NET, instanciez un client AWS Encryption SDK pour .NET et définissez son `MaxEncryptedDataKeys` paramètre facultatif sur la valeur souhaitée. Appelez ensuite la `Decrypt()` méthode sur l'AWS Encryption SDKinstance configurée.

```
// Decrypt with limited data keys

// Instantiate the material providers
var materialProviders =

 AwsCryptographicMaterialProvidersFactory.CreateDefaultAwsCryptographicMaterialProviders();

// Configure the commitment policy on the AWS Encryption SDK instance
var config = new AwsEncryptionSdkConfig
{
    MaxEncryptedDataKeys = 3
};
var encryptionSdk = AwsEncryptionSdkFactory.CreateAwsEncryptionSdk(config);

// Create the keyring
string keyArn = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
var createKeyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = keyArn
};
var decryptKeyring = materialProviders.CreateAwsKmsKeyring(createKeyringInput);
```

```
// Decrypt the ciphertext
var decryptInput = new DecryptInput
{
    Ciphertext = ciphertext,
    Keyring = decryptKeyring
};
var decryptOutput = encryptionSdk.Decrypt(decryptInput);
```

AWS Encryption CLI

```
# Decrypt with limited encrypted data keys

$ aws-encryption-cli --decrypt \
    --input hello.txt.encrypted \
    --wrapping-keys key=$key_arn1 key=$key_arn2 key=$key_arn3 \
    --buffer \
    --max-encrypted-data-keys 3 \
    --encryption-context purpose=test \
    --metadata-output ~/metadata \
    --output .
```

Java

```
// Construct a client with limited encrypted data keys
final AwsCrypto crypto = AwsCrypto.builder()
  .withMaxEncryptedDataKeys(3)
  .build();

// Create an AWS KMS master key provider
final KmsMasterKeyProvider keyProvider = KmsMasterKeyProvider.builder()
      .buildStrict(keyArn1, keyArn2, keyArn3);

// Decrypt
final CryptoResult<byte[], KmsMasterKey> decryptResult =
 crypto.decryptData(keyProvider, ciphertext)
```

JavaScript Browser

```
// Construct a client with limited encrypted data keys
const { encrypt, decrypt } = buildClient({ maxEncryptedDataKeys: 3 })

declare const credentials: {
```

```
  accessKeyId: string
  secretAccessKey: string
  sessionToken: string
}
const clientProvider = getClient(KMS, {
  credentials: { accessKeyId, secretAccessKey, sessionToken }
})

// Create an AWS KMS keyring
const keyring = new KmsKeyringBrowser({
  clientProvider,
  keyIds: [keyArn1, keyArn2, keyArn3],
})

// Decrypt
const { plaintext, messageHeader } = await decrypt(keyring, ciphertext)
```

JavaScript Node.js

```
// Construct a client with limited encrypted data keys
const { encrypt, decrypt } = buildClient({ maxEncryptedDataKeys: 3 })

// Create an AWS KMS keyring
const keyring = new KmsKeyringBrowser({
  keyIds: [keyArn1, keyArn2, keyArn3],
})

// Decrypt
const { plaintext, messageHeader } = await decrypt(keyring, ciphertext)
```

Python

```
# Instantiate a client with limited encrypted data keys
client = aws_encryption_sdk.EncryptionSDKClient(max_encrypted_data_keys=3)

# Create an AWS KMS master key provider
master_key_provider = aws_encryption_sdk.StrictAwsKmsMasterKeyProvider(
    key_ids=[key_arn1, key_arn2, key_arn3])

# Decrypt
plaintext, header = client.decrypt(source=ciphertext,
 key_provider=master_key_provider)
```

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 de Chine	`aws-cn`
AWS GovCloud (US) Regions	`aws-us-gov`

Les exemples présentés dans cette section montrent comment créer un filtre de découverte. Avant d'utiliser le code, remplacez les valeurs d'exemple par des valeurs valides pour la partition Compte AWS and.

C

Pour des exemples complets, consultez le fichier kms_discovery.cpp dans leKit SDK de chiffrement AWS pour C.

```
/* Create a discovery filter for an AWS account and partition */

const char *account_id = "111122223333";
const char *partition = "aws";
const std::shared_ptr<Aws::Cryptosdk::KmsKeyring::DiscoveryFilter> discovery_filter
 =
```

```
 Aws::Cryptosdk::KmsKeyring::DiscoveryFilter::Builder(partition).AddAccount(account_id).Buil
```

C# / .NET

Pour un exemple complet, voir DiscoveryFilterExample.cs dans le fichier AWS Encryption SDK pour .NET.

```
// Create a discovery filter for an AWS account and partition

List<string> account = new List<string> { "111122223333" };

DiscoveryFilter exampleDiscoveryFilter = new DiscoveryFilter()
{
    AccountIds = account,
    Partition = "aws"
}
```

AWS Encryption CLI

```
# Decrypt in discovery mode with a discovery filter

$ aws-encryption-cli --decrypt \
                     --input hello.txt.encrypted \
                     --wrapping-keys discovery=true \
                                     discovery-account=111122223333 \
                                     discovery-partition=aws \
                     --encryption-context purpose=test \
                     --metadata-output ~/metadata \
                     --max-encrypted-data-keys 1 \
                     --buffer \
                     --output .
```

Java

Pour un exemple complet, voir DiscoveryDecryptionExample.java dans leKit SDK de chiffrement AWS pour Java.

```
// Create a discovery filter for an AWS account and partition

DiscoveryFilter discoveryFilter = new DiscoveryFilter("aws", 111122223333);
```

JavaScript (Node and Browser)

Pour des exemples complets, consultez kms_filtered_discovery.ts (Node.js) et kms_multi_region_discovery.ts (navigateur) dans le. Kit SDK de chiffrement AWS pour JavaScript

```
/* Create a discovery filter for an AWS account and partition */
const discoveryFilter = {
  accountIDs: ['111122223333'],
  partition: 'aws',
}
```

Python

Pour un exemple complet, consultez le fichier discovery_kms_provider.py dans leKit SDK de chiffrement AWS pour Python.

```
# Create the discovery filter and specify the region
decrypt_kwargs = dict(
        discovery_filter=DiscoveryFilter(account_ids="111122223333",
 partition="aws"),
        discovery_region="us-west-2",
    )
```

Définition d'une politique d'engagement

Une politique d'engagement est un paramètre de configuration qui détermine si votre application chiffre et déchiffre avec un engagement clé. Le chiffrement et le déchiffrement avec un engagement clé constituent une bonne pratique. AWS Encryption SDK

La définition et l'ajustement de votre politique d'engagement constituent une étape essentielle de la migration depuis les versions 1.7. x et versions antérieures AWS Encryption SDK à la version 2.0. x et versions ultérieures. Cette progression est expliquée en détail dans la rubrique relative à la migration.

La valeur de la politique d'engagement par défaut dans les dernières versions du AWS Encryption SDK (à partir de la version 2.0). x)`RequireEncryptRequireDecrypt`, est idéal pour la plupart des situations. Toutefois, si vous devez déchiffrer un texte chiffré sans engagement de clé, vous devrez peut-être modifier votre politique d'engagement en. `RequireEncryptAllowDecrypt` Pour des exemples de définition d'une politique d'engagement dans chaque langage de programmation, voirDéfinition de votre politique d'engagement.

Utilisation de données en streaming

Lorsque vous diffusez des données à des fins de déchiffrement, sachez que le texte brut AWS Encryption SDK renvoyé est déchiffré une fois les contrôles d'intégrité terminés, mais avant que la signature numérique ne soit vérifiée. Pour vous assurer de ne pas renvoyer ou utiliser du texte en clair tant que la signature n'est pas vérifiée, nous vous recommandons de mettre en mémoire tampon le texte en clair diffusé jusqu'à ce que le processus de déchiffrement soit complet.

Ce problème se produit uniquement lorsque vous diffusez du texte chiffré à des fins de déchiffrement, et uniquement lorsque vous utilisez une suite d'algorithmes, telle que la suite d'algorithmes par défaut, qui inclut des signatures numériques.

Pour faciliter la mise en mémoire tampon, certaines implémentations de AWS Encryption SDK langage, comme Kit SDK de chiffrement AWS pour JavaScript dans Node.js, incluent une fonctionnalité de mise en mémoire tampon dans le cadre de la méthode de déchiffrement. La CLI de AWS chiffrement, qui diffuse toujours les entrées et les sorties, a introduit un `--buffer` paramètre dans les versions 1.9. x et 2.2. x. Dans d'autres implémentations linguistiques, vous pouvez utiliser les fonctionnalités de mise en mémoire tampon existantes. (Le AWS Encryption SDK pour .NET ne prend pas en charge le streaming.)

Si vous utilisez une suite d'algorithmes sans signature numérique, veillez à utiliser `decrypt-unsigned` cette fonctionnalité dans chaque implémentation de langage. Cette fonctionnalité déchiffre le texte chiffré mais échoue en cas de détection de texte chiffré signé. Pour plus de détails, consultez Choix d'une suite d'algorithmes.

Mise en cache des clés de données

En général, la réutilisation des clés de données est déconseillée, mais AWS Encryption SDK propose une option de mise en cache des clés de données qui permet une réutilisation limitée des clés de données. La mise en cache des clés de données peut améliorer les performances de certaines applications et réduire les appels vers votre infrastructure clé. Avant d'utiliser la mise en cache des clés de données en production, ajustez les seuils de sécurité et testez pour vous assurer que les avantages l'emportent sur les inconvénients de la réutilisation des clés de données.

Utilisation des porte-clés

Les Kit SDK de chiffrement AWS pour C, les Kit SDK de chiffrement AWS pour JavaScript Kit SDK de chiffrement AWS pour Java, et AWS Encryption SDK pour .NET utilisent des porte-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 message, 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 le porte-clés fourni par le kit SDK fournit ou personnaliser vos propres porte-clés compatibles.

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 à l'intérieur d'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 (HSM) ou protégées par d'autres services de clés principales. Pour de plus amples informations, veuillez consulter la rubrique Keyring Interface dans la Spécification AWS Encryption SDK .

Les porte-clés jouent le rôle de clés principales et de fournisseurs de clés principales dans la Kit SDK de chiffrement AWS pour Java CLI de chiffrement et dans la CLI de AWS chiffrement. Kit SDK de chiffrement AWS pour Python Si vous utilisez différentes implémentations linguistiques du AWS Encryption SDK pour chiffrer et déchiffrer vos données, veillez à utiliser des trousseaux de clés et des fournisseurs de clés principales compatibles. Pour plus de détails, consultez Compatibilité du porte-clés.

Cette rubrique explique comment utiliser la fonction porte-clés du porte-clés AWS Encryption SDK et comment choisir un porte-clés. Pour des exemples de création et d'utilisation de trousseaux de clés, consultez le C et les JavaScriptrubriques.

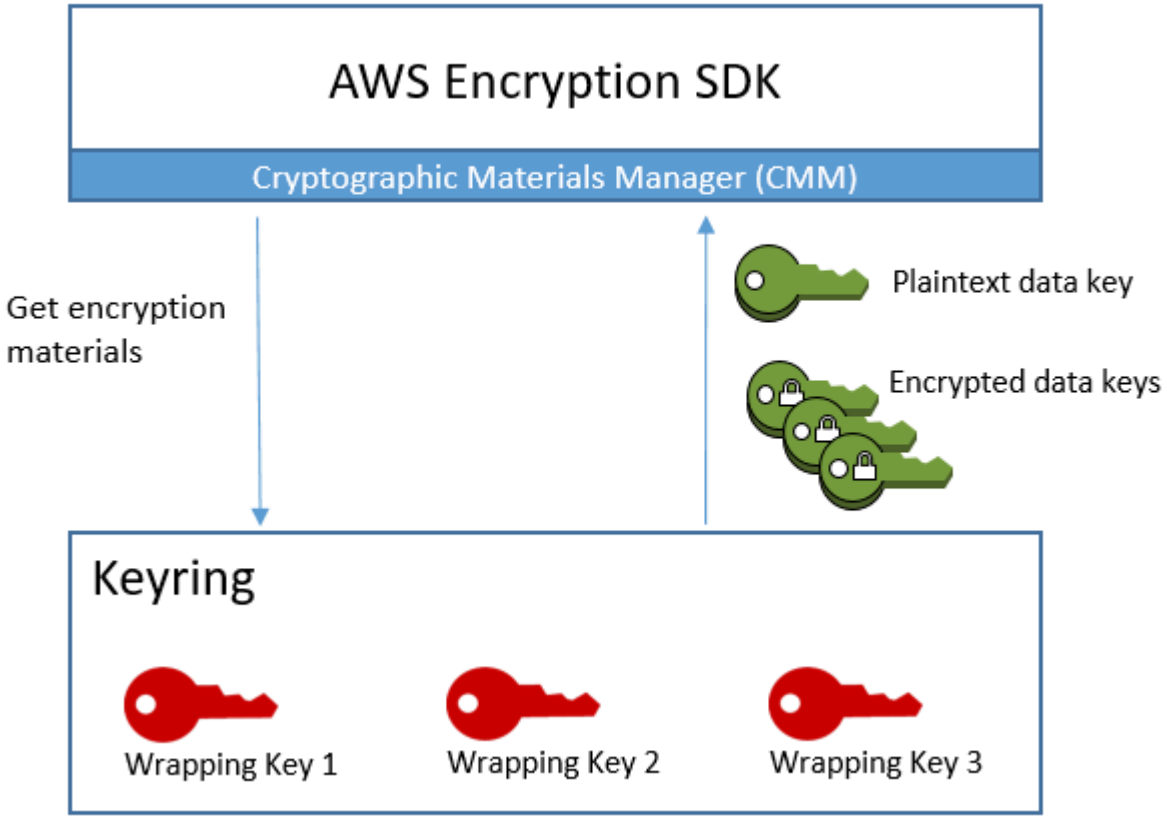
Rubriques

- Fonctionnement des porte-clés
- Compatibilité du porte-clés

- Choisir un porte-clés

Fonctionnement des porte-clés

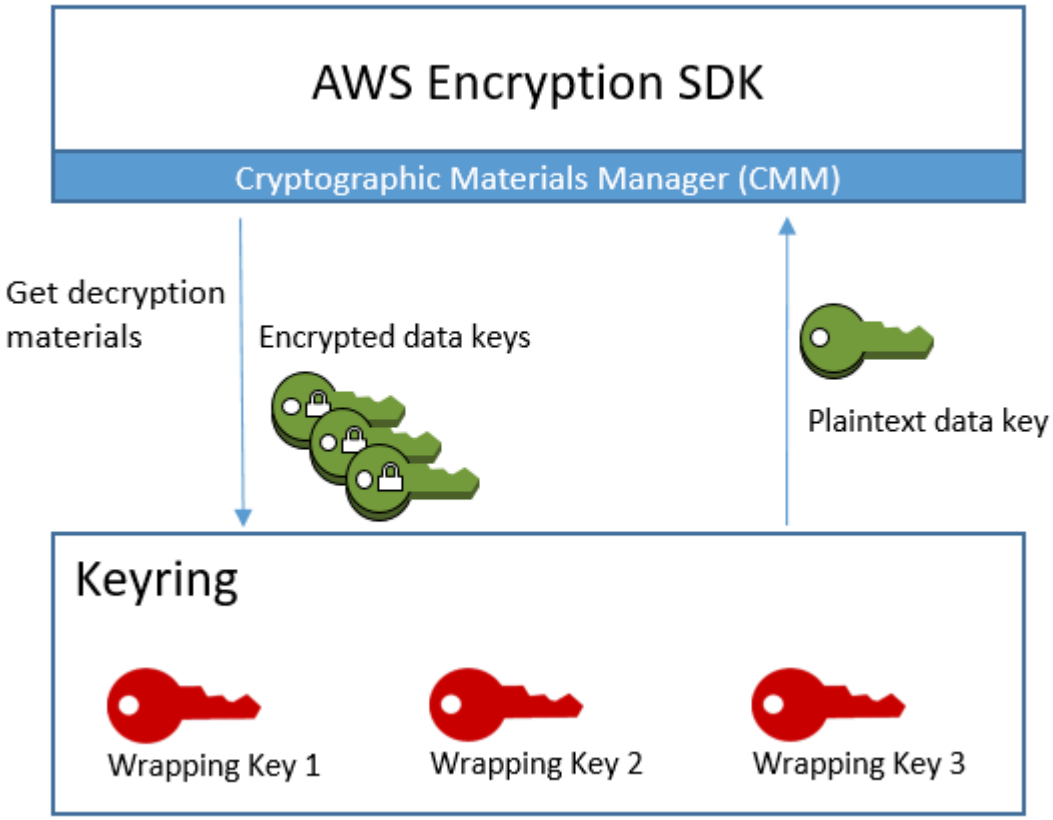
Lorsque vous chiffrez des données, le matériel de cryptage est AWS Encryption SDK demandé au trousseau de clés. Le porte-clés renvoie une clé de données en texte brut et une copie de la clé de données chiffrée par chacune des clés d'encapsulage du trousseau de clés. Il AWS Encryption SDK utilise la clé en texte brut pour chiffrer les données, puis détruit la clé de données en texte brut. Ensuite, il AWS Encryption SDK renvoie un message crypté qui inclut les clés de données cryptées et les données cryptées.



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 inclure (ou avoir accès à) au moins une clé d'encapsulation dans le jeu de clés de chiffrement.

Il AWS Encryption SDK transmet les clés de données cryptées du message crypté au trousseau de clés et demande au trousseau de déchiffrer l'une d'entre elles. Le porte-clés utilise ses clés

d'encapsulage pour déchiffrer l'une des clés de données chiffrées et renvoie une clé de données en texte brut. Le kit AWS Encryption SDK utilise la clé de données en texte brut pour déchiffrer les données. Si aucune des clés d'encapsulage 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 les données, les porte-clés multiple renvoie une copie de la clé de données chiffrée par toutes les clés d'encapsulage dans toutes les porte-clés qui composent le porte-clés multiple. Vous pouvez déchiffrer les données à l'aide d'un trousseau de clés avec n'importe laquelle des clés d'encapsulage du trousseau de clés multiples.

Compatibilité du porte-clés

Bien que les différentes implémentations linguistiques du AWS Encryption SDK présentent certaines différences architecturales, elles sont entièrement compatibles, sous réserve des contraintes linguistiques. Vous pouvez chiffrer vos données à l'aide d'une implémentation linguistique et les déchiffrer avec n'importe quelle autre implémentation linguistique. Toutefois, vous devez utiliser les mêmes clés d'encapsulation ou des clés d'encapsulation correspondantes pour chiffrer et déchiffrer

vos clés de données. Pour plus d'informations sur les contraintes linguistiques, consultez la rubrique relative à chaque implémentation de langage, comme the section called “Compatibilité” dans la Kit SDK de chiffrement AWS pour JavaScript rubrique.

Exigences variables pour les trousseaux de clés de chiffrement

Dans les implémentations de AWS Encryption SDK langage autres que le Kit SDK de chiffrement AWS pour C, toutes les clés d'encapsulation d'un jeu de clés de chiffrement (ou d'un jeu de clés multiples) ou d'un fournisseur de clés principales 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.

L'exception est le Kit SDK de chiffrement AWS pour C cas où l'opération de chiffrement ignore un jeu de clés de découverte standard, mais échoue si vous spécifiez un jeu de clés de découverte multirégional, seul ou dans un jeu de clés multiclés.

Porte-clés et fournisseurs de clés principales compatibles

Le tableau suivant indique les clés principales et les fournisseurs de clés principales compatibles avec les porte-clés fournis AWS Encryption SDK . Toute incompatibilité mineure due à des contraintes de langage est expliquée dans la rubrique relative à l'implémentation du langage.

Porte-clés :	Fournisseur de clés principales :
AWS KMS porte-clés	KMS MasterKey (Java) KMS MasterKeyProvider (Java) KMS MasterKey (Python) KMS MasterKeyProvider (Python) **Note** Le Kit SDK de chiffrement AWS pour Python et Kit SDK de chiffrement AWS pour Java n'incluez pas de clé principale ou

Porte-clés :	Fournisseur de clés principales :
	de fournisseur de clé principale équivalent au trousseau de clés de découverte AWS KMS régional.
AWS KMS Porte-clés hiérarchique	Disponible uniquement avec la version 4. x du AWS Encryption SDK pour .NET et la version 3. x du Kit SDK de chiffrement AWS pour Java.
AWS KMS Porte-clés ECDH	Disponible uniquement avec la version 3. x du Kit SDK de chiffrement AWS pour Java.
Porte-clés AES brut	Lorsqu'ils sont utilisés avec des clés de chiffrement symétriques : JceMasterKey(Java) RawMasterKey(Python)
Porte-clés RSA brut	Lorsqu'ils sont utilisés avec des clés de chiffrement asymétriques : JceMasterKey(Java) RawMasterKey(Python) **Note** Le porte-clés RSA brut ne prend pas en charge les clés KMS asymétriques. Si vous souhaitez utiliser des clés RSA KMS asymétriques, version 4. x of the AWS Encryption SDK for .NET prend en charge les AWS KMS jeux de clés utilisant le chiffrement symétrique (`SYMMETRIC_DEFAULT`) ou le RSA asymétrique. AWS KMS keys
Porte-clés ECDH brut	Disponible uniquement avec la version 3. x du Kit SDK de chiffrement AWS pour Java.

Choisir un porte-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. Dans la mesure du possible, utilisez des clés d'encapsulation protégées par un module de sécurité matériel ou une infrastructure de gestion des clés, telles que les clés KMS in AWS Key Management Service(AWS KMS) ou les clés de chiffrement AWS CloudHSM.

AWS Encryption SDK Il fournit plusieurs porte-clés et configurations de porte-clés dans plusieurs langages de programmation, et vous pouvez créer vos propres porte-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

- 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

AWS KMS porte-clés

Un AWS KMS trousseau de clés utilise un chiffrement symétrique AWS KMS keyspour générer, chiffrer et déchiffrer des 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 utiliser une clé AWS KMS multirégionale dans un AWS KMS trousseau de clés ou un fournisseur de clés principales à partir de la version 2.3. x du AWS Encryption SDK et version 3.0. x de la CLI AWS de chiffrement. Pour plus de détails et des exemples d'utilisation du nouveau symbole prenant en compte plusieurs régions, voir. 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.

> **Note**
>
> La version 4. x du AWS Encryption SDK pour .NET et la version 3. x d'entre eux Kit SDK de chiffrement AWS pour Java sont les seules implémentations de langage de programmation qui prennent en charge les trousseaux de AWS KMS clés utilisant le RSA asymétrique. AWS KMS keys
> Si vous essayez d'inclure une clé KMS asymétrique dans un jeu de clés de chiffrement dans une autre implémentation de langage, l'appel de chiffrement échoue. Si vous l'incluez dans un trousseau de clés de déchiffrement, il est ignoré.
> Toutes les mentions de porte-clés KMS dans le document AWS Encryption SDK font référence aux AWS KMS porte-clés.

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.

Lors du chiffrement, le AWS KMS trousseau de clés que vous utilisez doit comporter une clé génératrice. Lors du déchiffrement, la clé du générateur est facultative et la distinction entre les clés du générateur et les clés supplémentaires est ignorée.

Lorsqu'un jeu de clés de AWS KMS chiffrement 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

- Utilisation d'un porte-clés de découverte AWS KMS régional

Autorisations requises pour les AWS KMS porte-clés

AWS Encryption SDK Cela ne nécessite pas de Compte AWS et ne dépend d'aucun 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'GenerateDataKeyautorisation 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 clé 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'GenerateDataKeyautorisation 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 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.

- Dans un jeu de clés de chiffrement pour le Kit SDK de chiffrement AWS pour C, vous pouvez utiliser un ARN clé ou un alias ARN pour identifier les clés KMS. Dans toutes les autres implémentations de langage, vous pouvez utiliser un identifiant de clé, un ARN de clé, un nom d'alias ou un ARN d'alias pour chiffrer les données.

- Dans un porte-clés de déchiffrement, vous devez utiliser un ARN de clé pour identifier les AWS KMS keys. Cette exigence s'applique à toutes les implémentations de langage du kit AWS Encryption SDK. Pour plus de détails, 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. Cette exigence s'applique à toutes les implémentations de langage du kit AWS Encryption SDK.

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.

Création d'un AWS KMS trousseau de clés pour le chiffrement

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 Les clés de chiffrement AWS KMS keys doivent être symétriques (SYMMETRIC_DEFAULT). Vous pouvez également utiliser une clé KMS multirégionale à chiffrement symétrique. Comme pour tous les porte-clés, vous pouvez utiliser un ou plusieurs AWS KMS porte-clés dans un porte-clés multiple.

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 le message chiffré protégé par ce trousseau de clés, le trousseau 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.)

Dans les implémentations de AWS Encryption SDK langage autres que le Kit SDK de chiffrement AWS pour C, toutes les clés d'encapsulage 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. L'exception est le Kit SDK de chiffrement AWS pour C cas où l'opération de

chiffrement ignore un jeu de clés de découverte standard, mais échoue si vous spécifiez un jeu de clés de découverte multirégional, seul ou dans un jeu de clés multiclés.

Les exemples suivants créent un AWS KMS trousseau de clés avec une clé génératrice et une clé supplémentaire. Ces exemples utilisent des ARN clés pour identifier les clés KMS. Il s'agit d'une bonne pratique pour les AWS KMS trousseaux de clés utilisés pour le chiffrement et d'une exigence pour les AWS KMS trousseaux de clés utilisés pour le déchiffrement. Pour plus de détails, consultez Identification AWS KMS keys dans un AWS KMS porte-clés.

C

Pour identifier un élément AWS KMS key dans un jeu de clés de chiffrement dans le Kit SDK de chiffrement AWS pour C, spécifiez un ARN clé ou un alias ARN. Dans un porte-clés de déchiffrement, vous devez utiliser un ARN de clé. Pour plus de détails, consultez Identification AWS KMS keys dans un AWS KMS porte-clés.

Pour obtenir un exemple complet, veuillez consulter string.cpp.

```
const char * generator_key = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"

const char * additional_key = "arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321"

struct aws_cryptosdk_keyring *kms_encrypt_keyring =
       Aws::Cryptosdk::KmsKeyring::Builder().Build(generator_key,{additional_key});
```

C# / .NET

Pour créer un AWS KMS trousseau de clés avec une ou plusieurs AWS KMS clés dans .NET, créez un trousseau de clés multiples. AWS Encryption SDK Le AWS Encryption SDK for .NET inclut un porte-clés multi-clés uniquement pour les AWS KMS clés.

Lorsque vous spécifiez un AWS KMS key pour un jeu de clés de chiffrement dans le fichier AWS Encryption SDK pour .NET, vous pouvez utiliser n'importe quel identifiant de clé valide : un ID de clé, un ARN de clé, un nom d'alias ou un ARN d'alias. Pour obtenir de l'aide pour identifier le AWS KMS keys contenu d'un AWS KMS trousseau de clés, voirIdentification AWS KMS keys dans un AWS KMS porte-clés.

L'exemple suivant utilise la version 4. x du AWS Encryption SDK pour .NET pour créer un AWS KMS trousseau de clés avec une clé génératrice et des clés supplémentaires. Pour un exemple complet, consultez AwsKmsMultiKeyringExample.cs.

```
// Instantiate the AWS Encryption SDK and material provider
var mpl = new MaterialProviders(new MaterialProvidersConfig());
var esdk =  new ESDK(new AwsEncryptionSdkConfig());

string generatorKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
List<string> additionalKey = new List<string> { "alias/exampleAlias" };

// Instantiate the keyring input object
var kmsEncryptKeyringInput = new CreateAwsKmsMultiKeyringInput
{
    Generator = generatorKey,
    KmsKeyIds = additionalKey
};

var kmsEncryptKeyring =
 materialProviders.CreateAwsKmsMultiKeyring(kmsEncryptKeyringInput);
```

JavaScript Browser

Lorsque vous spécifiez un AWS KMS key pour un jeu de clés de chiffrement dans le Kit SDK de chiffrement AWS pour JavaScript, vous pouvez utiliser n'importe quel identifiant de clé valide : un ID de clé, un ARN de clé, un nom d'alias ou un ARN d'alias. Pour obtenir de l'aide pour identifier le AWS KMS keys contenu d'un AWS KMS trousseau de clés, voirIdentification AWS KMS keys dans un AWS KMS porte-clés.

Pour un exemple complet, consultez kms_simple.ts dans le référentiel de. Kit SDK de chiffrement AWS pour JavaScript GitHub

```
const clientProvider = getClient(KMS, { credentials })
const generatorKeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
const additionalKey = 'alias/exampleAlias'

const keyring = new KmsKeyringBrowser({
  clientProvider,
  generatorKeyId,
  keyIds: [additionalKey]
```

```
})
```

JavaScript Node.js

Lorsque vous spécifiez un AWS KMS key pour un jeu de clés de chiffrement dans le Kit SDK de chiffrement AWS pour JavaScript, vous pouvez utiliser n'importe quel identifiant de clé valide : un ID de clé, un ARN de clé, un nom d'alias ou un ARN d'alias. Pour obtenir de l'aide pour identifier le AWS KMS keys contenu d'un AWS KMS trousseau de clés, voirIdentification AWS KMS keys dans un AWS KMS porte-clés.

Pour un exemple complet, consultez kms_simple.ts dans le référentiel de. Kit SDK de chiffrement AWS pour JavaScript GitHub

```
const generatorKeyId = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

const additionalKey = 'alias/exampleAlias'

const keyring = new KmsKeyringNode({
  generatorKeyId,
  keyIds: [additionalKey]
})
```

Java

Pour créer un AWS KMS porte-clés contenant une ou plusieurs AWS KMS clés Kit SDK de chiffrement AWS pour Java, créez un porte-clés multiple. Kit SDK de chiffrement AWS pour Java Il comprend un porte-clés multiple juste pour les AWS KMS clés.

Lorsque vous spécifiez un AWS KMS key pour un jeu de clés de chiffrement dans le Kit SDK de chiffrement AWS pour Java, vous pouvez utiliser n'importe quel identifiant de clé valide : un ID de clé, un ARN de clé, un nom d'alias ou un ARN d'alias. Pour obtenir de l'aide pour identifier le AWS KMS keys contenu d'un AWS KMS trousseau de clés, voirIdentification AWS KMS keys dans un AWS KMS porte-clés.

Pour un exemple complet, consultez le BasicEncryptionKeyringExamplefichier .java dans le Kit SDK de chiffrement AWS pour Java référentiel dans GitHub.

```
 // Instantiate the AWS Encryption SDK and material providers
final AwsCrypto crypto = AwsCrypto.builder().build();
final MaterialProviders materialProviders = MaterialProviders.builder()
            .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
```

```
            .build();

String generatorKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
List<String> additionalKey = Collections.singletonList("alias/exampleAlias");

// Create the AWS KMS keyring
final CreateAwsKmsMultiKeyringInput keyringInput =
 CreateAwsKmsMultiKeyringInput.builder()
        .generator(generatorKey)
        .kmsKeyIds(additionalKey)
        .build();
final IKeyring kmsKeyring = matProv.CreateAwsKmsMultiKeyring(keyringInput);
```

Création d'un AWS KMS trousseau de clés pour le déchiffrement

Vous spécifiez également un AWS KMS trousseau de clés lors du déchiffrement du message chiffré renvoyé. AWS Encryption SDK Si le jeu de clés de déchiffrement le spécifie AWS KMS keys, seules ces clés d'encapsulation AWS Encryption SDK seront utilisées pour déchiffrer les clés de données chiffrées du message chiffré. (Vous pouvez également utiliser un trousseau de clés de AWS KMS découverte, qui n'en spécifie aucun AWS KMS keys.)

Lors du déchiffrement, il AWS Encryption SDK recherche dans le AWS KMS trousseau de clés un élément capable de déchiffrer l'une AWS KMS key des clés de données cryptées. Plus précisément, le modèle suivant est AWS Encryption SDK utilisé pour chaque clé de données chiffrée d'un message chiffré.

- Il AWS Encryption SDK obtient l'ARN de la clé AWS KMS key qui a chiffré la clé de données à partir des métadonnées du message crypté.
- Il AWS Encryption SDK recherche dans le trousseau de clés de déchiffrement un ARN AWS KMS key avec une clé correspondante.
- S'il trouve un AWS KMS key ARN correspondant dans le trousseau de clés, il AWS Encryption SDK demande AWS KMS à utiliser la clé KMS pour déchiffrer la clé de données chiffrée.
- Dans le cas contraire, il passe à la clé de données chiffrée suivante, le cas échéant.

Il AWS Encryption SDK ne tente jamais de déchiffrer une clé de données chiffrée à moins AWS KMS key que l'ARN de la clé chiffrée ne soit inclus dans le trousseau de clés de déchiffrement. Si le trousseau de clés de déchiffrement n'inclut pas les ARN de l' AWS KMS keys une des clés de

données chiffrées, l'appel de déchiffrement AWS Encryption SDK échoue sans jamais appeler. AWS KMS

À partir de la version 1.7. x, lors du déchiffrement d'une clé de données cryptée, transmet AWS Encryption SDK toujours l'ARN de la clé AWS KMS key au `KeyId` paramètre de l'opération de déchiffrement. AWS KMS L'identifier AWS KMS key lors du déchiffrement est une AWS KMS bonne pratique qui garantit que vous déchiffrez la clé de données cryptée avec la clé d'encapsulation que vous souhaitez utiliser.

Un appel de déchiffrement avec un AWS KMS trousseau de clés réussit lorsqu'au moins un AWS KMS key élément du trousseau de clés de déchiffrement peut déchiffrer l'une des clés de données chiffrées du message chiffré. En outre, le mandataire doit disposer de l'autorisation `kms:Decrypt` pour cette AWS KMS key. Ce comportement vous permet de chiffrer des données AWS KMS keys sous plusieurs comptes Régions AWS et de fournir un jeu de clés de déchiffrement plus limité adapté à un compte, une région, un utilisateur, un groupe ou un rôle en particulier.

Lorsque vous spécifiez un AWS KMS key dans un trousseau de clés de déchiffrement, vous devez utiliser son ARN clé. Dans le cas contraire, le n' AWS KMS key est pas reconnu. Pour obtenir de l'aide pour trouver l'ARN de la clé, consultez la section Trouver l'ID et l'ARN de la clé dans le guide du AWS Key Management Service développeur.

> **Note**
>
> Si vous réutilisez un porte-clés de chiffrement pour le déchiffrement, assurez-vous que les AWS KMS keys du porte-clés sont identifiées par leurs ARN de clé.

Par exemple, le jeu de AWS KMS clés suivant inclut uniquement la clé supplémentaire qui a été utilisée dans le jeu de clés de chiffrement. Toutefois, au lieu de faire référence à la clé supplémentaire par son alias`alias/exampleAlias`, l'exemple utilise l'ARN de la clé supplémentaire comme l'exigent les appels de déchiffrement.

Vous pouvez utiliser ce porte-clés pour déchiffrer un message qui a été chiffré sous la clé du générateur et la clé supplémentaire, à condition que vous soyez autorisé à utiliser la clé supplémentaire pour déchiffrer les données.

C

```
const char * additional_key = "arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321"
```

```
struct aws_cryptosdk_keyring *kms_decrypt_keyring =
       Aws::Cryptosdk::KmsKeyring::Builder().Build(additional_key);
```

C# / .NET

Comme ce trousseau de clés de déchiffrement ne comprend qu'une seule AWS KMS clé, l'exemple utilise la `CreateAwsKmsKeyring()` méthode avec une instance de son `CreateAwsKmsKeyringInput` objet. Pour créer un AWS KMS porte-clés avec une seule AWS KMS clé, vous pouvez utiliser un porte-clés à une ou plusieurs touches. Pour plus de détails, consultez Chiffrer des données dans le AWS Encryption SDK pour .NET. L'exemple suivant utilise la version 4. x du AWS Encryption SDK pour que .NET crée un AWS KMS trousseau de clés pour le déchiffrement.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk =  new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

string additionalKey = "arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321";

// Instantiate a KMS keyring for one AWS KMS key.
var kmsDecryptKeyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = additionalKey
};

var kmsDecryptKeyring =
 materialProviders.CreateAwsKmsKeyring(kmsDecryptKeyringInput);
```

JavaScript Browser

```
const clientProvider = getClient(KMS, { credentials })
const additionalKey = 'arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321'

const keyring = new KmsKeyringBrowser({ clientProvider, keyIds: [additionalKey] })
```

JavaScript Node.js

```
const additionalKey = 'arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321'

const keyring = new KmsKeyringNode({ keyIds: [additionalKey] })
```

Java

Comme ce trousseau de clés de déchiffrement ne comprend qu'une seule AWS KMS clé, l'exemple utilise la `CreateAwsKmsKeyring()` méthode avec une instance de son `CreateAwsKmsKeyringInput` objet. Pour créer un AWS KMS porte-clés avec une seule AWS KMS clé, vous pouvez utiliser un porte-clés à une ou plusieurs touches.

```
// Instantiate the AWS Encryption SDK and material providers
final AwsCrypto crypto = AwsCrypto.builder().build();
final MaterialProviders materialProviders = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();

String additionalKey = "arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321";

// Create a AwsKmsKeyring
CreateAwsKmsKeyringInput kmsDecryptKeyringInput = CreateAwsKmsKeyringInput.builder()
        .generator(additionalKey)
        .kmsClient(KmsClient.create())
        .build();
IKeyring kmsKeyring = materialProviders.CreateAwsKmsKeyring(kmsDecryptKeyringInput);
```

Vous pouvez également utiliser un AWS KMS trousseau de clés qui spécifie une clé génératrice pour le déchiffrement, telle que la suivante. Lors du déchiffrement, le AWS Encryption SDK ignore la distinction entre les clés du générateur et les clés supplémentaires. Il peut utiliser n'importe lequel des éléments spécifiés AWS KMS keys pour déchiffrer une clé de données cryptée. L'appel à ne AWS KMS réussit que lorsque l'appelant est autorisé à l'utiliser pour déchiffrer AWS KMS key les données.

C

```
struct aws_cryptosdk_keyring *kms_decrypt_keyring =
```

```
      Aws::Cryptosdk::KmsKeyring::Builder().Build(generator_key, {additional_key,
 other_key});
```

C# / .NET

L'exemple suivant utilise la version 4. x du AWS Encryption SDK pour .NET.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk =  new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

string generatorKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

// Instantiate a KMS keyring for one AWS KMS key.
var kmsDecryptKeyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = generatorKey
};

var kmsDecryptKeyring =
 materialProviders.CreateAwsKmsKeyring(kmsDecryptKeyringInput);
```

JavaScript Browser

```
const clientProvider = getClient(KMS, { credentials })

const keyring = new KmsKeyringBrowser({
  clientProvider,
  generatorKeyId,
  keyIds: [additionalKey, otherKey]
})
```

JavaScript Node.js

```
const keyring = new KmsKeyringNode({
  generatorKeyId,
  keyIds: [additionalKey, otherKey]
})
```

Java

```
// Instantiate the AWS Encryption SDK and material providers
final AwsCrypto crypto = AwsCrypto.builder().build();
final MaterialProviders materialProviders = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();

String generatorKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

// Create a AwsKmsKeyring
CreateAwsKmsKeyringInput kmsDecryptKeyringInput = CreateAwsKmsKeyringInput.builder()
        .generator(generatorKey)
        .kmsClient(KmsClient.create())
        .build();
IKeyring kmsKeyring = materialProviders.CreateAwsKmsKeyring(kmsDecryptKeyringInput);
```

Contrairement à un jeu de clés de chiffrement qui utilise tous les éléments spécifiés AWS KMS keys, vous pouvez déchiffrer un message chiffré à l'aide d'un jeu de clés de déchiffrement AWS KMS keys qui inclut des éléments non liés au message chiffré et AWS KMS keys que l'appelant n'est pas autorisé à utiliser. Si un appel de déchiffrement envoyé à AWS KMS échoue, par exemple lorsque le mandataire ne dispose pas de l'autorisation requise, le kit AWS Encryption SDK se contente de passer à la clé de données chiffrée suivante.

Utilisation d'un porte-clés AWS KMS Discovery

Lors du déchiffrement, il est recommandé de spécifier les clés d'encapsulation qu'ils peuvent utiliser. AWS Encryption SDK Pour suivre cette bonne pratique, utilisez un jeu de clés de AWS KMS déchiffrement qui limite les clés AWS KMS d'encapsulage 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.

AWS Encryption SDK Il fournit un porte-clés de AWS KMS découverte standard et un porte-clés de découverte pour les clés AWS KMS multirégionales. Pour plus d'informations sur l'utilisation de clés multirégionales avec le AWS Encryption SDK, voirUtilisation 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. L'exception est le Kit SDK de chiffrement

AWS pour C cas où l'opération de chiffrement ignore un jeu de clés de découverte standard, mais échoue si vous spécifiez un jeu de clés de découverte multirégional, seul ou dans un jeu de clés multiclés.

Lors du déchiffrement, un jeu de clés de découverte permet de demander AWS Encryption SDK AWS KMS à déchiffrer n'importe quelle clé de données cryptée en utilisant AWS KMS key celle qui l'a chiffrée, indépendamment de qui la possède ou qui y a 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. 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.

AWS Encryption SDK Il fournit un porte-clés AWS KMS Discovery 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 AWS KMS key n'importe quel élément utilisé pour chiffrer une clé de données dans le message chiffré, juste pour que AWS KMS key l'appelant soit autorisé à l'utiliser pour le déchiffrer. Il est possible qu'il ne s'agisse pas de la AWS KMS key que le mandataire 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 AWS KMS jeu de clés de découverte peut être sensiblement plus lent que les autres car il AWS Encryption SDK tente 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 afin de limiter le nombre de clés KMS pouvant être utilisées à celles Comptes AWS des partitions spécifiées. Les filtres de découverte sont pris en charge dans les versions 1.7. x et versions

ultérieures du AWS Encryption SDK. 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.

Le code suivant instancie un jeu de clés de AWS KMS découverte avec un filtre de découverte qui limite les clés KMS AWS Encryption SDK pouvant être utilisées à celles de la `aws` partition et à l'exemple de compte 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 des régions chinoises, 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.

C

Pour obtenir un exemple complet, veuillez consulter : kms_discovery.cpp.

```
std::shared_ptr<KmsKeyring::> discovery_filter(
    KmsKeyring::DiscoveryFilter::Builder("aws")
        .AddAccount("111122223333")
        .Build());

struct aws_cryptosdk_keyring *kms_discovery_keyring =
 Aws::Cryptosdk::KmsKeyring::Builder()
       .BuildDiscovery(discovery_filter));
```

C# / .NET

L'exemple suivant utilise la version 4. x du AWS Encryption SDK pour .NET.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk =  new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

List<string> account = new List<string> { "111122223333" };

// In a discovery keyring, you specify an AWS KMS client and a discovery filter,
// but not a AWS KMS key
var kmsDiscoveryKeyringInput = new CreateAwsKmsDiscoveryKeyringInput
{
```

```
    KmsClient = new AmazonKeyManagementServiceClient(),
    DiscoveryFilter = new DiscoveryFilter()
    {
        AccountIds = account,
        Partition = "aws"
    }
};

var kmsDiscoveryKeyring =
 materialProviders.CreateAwsKmsDiscoveryKeyring(kmsDiscoveryKeyringInput);
```

JavaScript Browser

Dans JavaScript, vous devez spécifier explicitement la propriété de découverte.

```
const clientProvider = getClient(KMS, { credentials })

const discovery = true
const keyring = new KmsKeyringBrowser(clientProvider, {
    discovery,
    discoveryFilter: { accountIDs: [111122223333], partition: 'aws' }
})
```

JavaScript Node.js

Dans JavaScript, vous devez spécifier explicitement la propriété de découverte.

```
const discovery = true

const keyring = new KmsKeyringNode({
    discovery,
    discoveryFilter: { accountIDs: ['111122223333'], partition: 'aws' }
})
```

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);
```

Utilisation d'un porte-clés de découverte AWS KMS régional

Un trousseau de découverte AWS KMS régional est un trousseau de clés qui ne spécifie pas les ARN des clés KMS. Au lieu de cela, il permet AWS Encryption SDK 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, il AWS Encryption SDK 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 messages 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 trousseau de découverte régional Kit SDK de chiffrement AWS pour C tente de déchiffrer uniquement avec des clés KMS dans la région spécifiée. Lorsque vous utilisez un trousseau de clés de découverte dans Kit SDK de chiffrement AWS pour JavaScript et AWS Encryption SDK pour .NET, vous configurez la région sur le AWS KMS client. Ces AWS Encryption SDK

implémentations ne filtrent pas les clés KMS par région, mais AWS KMS échoueront 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. Les filtres de découverte sont pris en charge dans les versions 1.7. x et versions ultérieures du AWS Encryption SDK.

Par exemple, le code suivant crée un trousseau de clés de découverte AWS KMS régional avec un filtre de découverte. Ce porte-clés limite les deux clés KMS du AWS Encryption SDK compte 111122223333 dans la région de l'ouest des États-Unis (Oregon) (us-west-2).

C

Pour afficher ce porte-clés ainsi que la méthode `create_kms_client` dans un exemple pratique, consultez kms_discovery.cpp.

```
std::shared_ptr<KmsKeyring::DiscoveryFilter> discovery_filter(
    KmsKeyring::DiscoveryFilter::Builder("aws")
        .AddAccount("111122223333")
        .Build());

struct aws_cryptosdk_keyring *kms_regional_keyring =
 Aws::Cryptosdk::KmsKeyring::Builder()

 .WithKmsClient(create_kms_client(Aws::Region::US_WEST_2)).BuildDiscovery(discovery_filter))
```

C# / .NET

Le AWS Encryption SDK for .NET ne possède pas de trousseau de clés de découverte régional dédié. Cependant, vous pouvez utiliser plusieurs techniques pour limiter le nombre de clés KMS utilisées lors du déchiffrement dans une région donnée.

Le moyen le plus efficace de limiter le nombre de régions dans un jeu de clés de découverte consiste à utiliser un jeu de clés de découverte prenant en compte plusieurs régions, même si vous avez chiffré les données uniquement à l'aide de clés à région unique. Lorsqu'il rencontre des clés à région unique, le trousseau de clés compatible avec plusieurs régions n'utilise aucune fonctionnalité multirégionale.

Le trousseau de clés renvoyé par la `CreateAwsKmsMrkDiscoveryKeyring()` méthode filtre les clés KMS par région avant l'appel AWS KMS. Il envoie une demande de déchiffrement AWS

KMS uniquement lorsque la clé de données chiffrée a été chiffrée par une clé KMS dans la région spécifiée par le `Region` paramètre de l'`CreateAwsKmsMrkDiscoveryKeyringInput`objet.

Les exemples suivants utilisent la version 4. x du AWS Encryption SDK pour .NET.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk =  new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

List<string> account = new List<string> { "111122223333" };

// Create the discovery filter
var filter = DiscoveryFilter = new DiscoveryFilter
{
    AccountIds = account,
    Partition = "aws"
};

var regionalDiscoveryKeyringInput = new CreateAwsKmsMrkDiscoveryKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(RegionEndpoint.USWest2),
    Region = RegionEndpoint.USWest2,
    DiscoveryFilter = filter
};

var kmsRegionalDiscoveryKeyring =
 materialProviders.CreateAwsKmsMrkDiscoveryKeyring(regionalDiscoveryKeyringInput);
```

Vous pouvez également limiter les clés KMS à une Région AWS valeur particulière en spécifiant une région dans votre instance du AWS KMS client (AmazonKeyManagementServiceClient). Toutefois, cette configuration est moins efficace et potentiellement plus coûteuse que l'utilisation d'un trousseau de clés de découverte prenant en compte plusieurs régions. Au lieu de filtrer les clés KMS par région avant d'appeler AWS KMS, le service AWS Encryption SDK for .NET appelle AWS KMS chaque clé de données chiffrée (jusqu'à ce qu'il en déchiffre une) et s'appuie sur cette méthode AWS KMS pour limiter les clés KMS qu'il utilise à la région spécifiée.

L'exemple suivant utilise la version 4. x du AWS Encryption SDK pour .NET.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk =  new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());
```

```
List<string> account = new List<string> { "111122223333" };

// Create the discovery filter,
// but not a AWS KMS key
var createRegionalDiscoveryKeyringInput = new CreateAwsKmsDiscoveryKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(RegionEndpoint.USWest2),
    DiscoveryFilter = new DiscoveryFilter()
    {
        AccountIds = account,
        Partition = "aws"
    }
};

var kmsRegionalDiscoveryKeyring =
 materialProviders.CreateAwsKmsDiscoveryKeyring(createRegionalDiscoveryKeyringInput);
```

JavaScript Browser

```
const clientProvider = getClient(KMS, { credentials })

const discovery = true
const clientProvider = limitRegions(['us-west-2'], getKmsClient)
const keyring = new KmsKeyringBrowser(clientProvider, {
    discovery,
    discoveryFilter: { accountIDs: ['111122223333'], partition: 'aws' }
})
```

JavaScript Node.js

Pour afficher ce porte-clés, ainsi que `limitRegions` et la fonction, dans un exemple pratique, veuillez consulter kms_regional_discovery.ts.

```
const discovery = true
const clientProvider = limitRegions(['us-west-2'], getKmsClient)
const keyring = new KmsKeyringNode({
    clientProvider,
    discovery,
    discoveryFilter: { accountIDs: ['111122223333'], partition: 'aws' }
})
```

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);
```

Kit SDK de chiffrement AWS pour JavaScript Il exporte également une `excludeRegions` fonction pour Node.js et le navigateur. Cette fonction crée un jeu de clés de découverte AWS KMS régional qui omet certaines AWS KMS keys régions. L'exemple suivant crée un trousseau de clés de découverte AWS KMS régional qui peut être utilisé AWS KMS keys dans le compte 111122223333 partout Région AWS sauf dans l'est des États-Unis (Virginie du Nord) (us-east-1).

Il Kit SDK de chiffrement AWS pour C n'existe pas de méthode analogue, mais vous pouvez en implémenter une en créant une méthode personnalisée ClientSupplier.

Cet exemple montre le code pour Node.js.

```
const discovery = true
const clientProvider = excludeRegions(['us-east-1'], getKmsClient)
const keyring = new KmsKeyringNode({
    clientProvider,
    discovery,
    discoveryFilter: { accountIDs: [111122223333], partition: 'aws' }
})
```

AWS KMS Porte-clés hiérarchiques

> **Important**
>
> Le trousseau de clés AWS KMS hiérarchique n'est pris en charge que par la version 4. x du AWS Encryption SDK pour .NET et la version 3. x du Kit SDK de chiffrement AWS pour Java.

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 cryptez ou déchiffrez des données. 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 de branche 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 données unique pour chiffrer chaque message et chiffre chaque clé 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 a `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 se compose de deux partitions, l'une pour les opérations de chiffrement et l'autre pour les opérations de déchiffrement. La partition de chiffrement stocke les éléments de clé de branche assemblés à partir de la clé de branche active et les réutilise pour toutes les opérations de chiffrement jusqu'à l'expiration de la limite de cache. La partition de déchiffrement stocke les matériaux de clé de branche assemblés pour les autres versions de clé de branche identifiées lors des opérations de déchiffrement. La partition de déchiffrement peut stocker plusieurs versions de documents clés de branche actifs à la fois. Lorsqu'elle est configurée pour utiliser un fournisseur d'ID de clé de branche pour un environnement mutualisé, la partition de chiffrement peut également stocker plusieurs versions de matériaux de clé de branche à la fois. Pour plus d’informations, consultez Utilisation du trousseau de clés hiérarchique dans les environnements multilocataires.

> **Note**
>
> Toutes les mentions du porte-clés hiérarchique dans le AWS Encryption SDK font référence au porte-clés AWS KMS hiérarchique.

Rubriques

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 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 5.
2. S'il n'existe aucun matériel de clé de branche valide, le trousseau de clés hiérarchique interroge le magasin de clés de branche pour trouver la clé de branche active.
    a. Le magasin de clés de branche 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 de branche 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 chiffrement utilise les matériaux de chiffrement pour chiffrer les données. Pour plus d'informations, consultez Comment AWS Encryption SDK crypte les données.

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. La méthode de déchiffrement identifie la clé de données chiffrée à partir du message chiffré et la transmet au trousseau de clés hiérarchique.
2. Le trousseau hiérarchique désérialise 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 plus d'informations, consultez 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 de branche pour trouver la clé de branche correspondant à la version de clé de branche identifiée à l'étape 2.

   a. Le magasin de clés de branche 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 de branche 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.

Le procédé de déchiffrement utilise le matériel de déchiffrement et la clé de données en texte brut pour déchiffrer le message chiffré. Pour plus d'informations, voir Comment AWS Encryption SDK déchiffre un message chiffré.

Prérequis

AWS Encryption SDK Cela ne nécessite pas de Compte AWS et ne dépend d'aucun Service AWS. Toutefois, le trousseau de clés hiérarchique dépend d'Amazon AWS KMS DynamoDB.

Pour utiliser un trousseau de clés hiérarchique, vous avez besoin d'un chiffrement symétrique AWS KMS key avec les autorisations KMS:Decrypt. Vous pouvez également utiliser une clé multirégionale de chiffrement symétrique. Pour 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.

Avant de pouvoir créer et utiliser un trousseau de clés hiérarchique, vous devez créer votre magasin de clés de succursale et le remplir avec votre première clé de branche active.

Étape 1 : Configuration d'un nouveau service de stockage de clés

Le service de magasin de clés fournit plusieurs opérations d'API, telles que `CreateKeyStore` et`CreateKey`, pour vous aider à assembler les prérequis relatifs au jeu de clés hiérarchique et à gérer le magasin de clés de votre succursale.

L'exemple suivant crée un service de stockage de clés. Vous devez spécifier un nom de table DynamoDB qui servira de nom à votre magasin de clés de branche, un nom logique pour le magasin de clés de branche et l'ARN de la clé KMS qui identifie la clé KMS qui protégera vos clés de branche.

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. Le nom logique du magasin de clés peut être identique au nom de votre table DynamoDB, mais ce n'est pas obligatoire. Nous vous recommandons de spécifier le nom de votre table DynamoDB comme nom de table logique lorsque vous configurez votre service de banque de clés pour la première fois. Vous devez toujours spécifier le même nom de table logique. Si le nom de votre banque de clés de branche change après la restauration de votre table DynamoDB à partir d'une sauvegarde, le nom de la banque de clés logique correspond au nom de la table DynamoDB que vous spécifiez pour garantir que le trousseau de clés hiérarchique peut toujours accéder à votre magasin de clés de succursale.

> **Note**
>
> Le nom logique du magasin de clés est inclus dans le contexte de chiffrement de toutes les opérations d'API du service de magasin de clés appelées AWS KMS. Le contexte de

chiffrement n'est pas secret, ses valeurs, y compris le nom du magasin de clés logiques, apparaissent en texte clair dans les journaux. AWS CloudTrail

C# / .NET

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

Java

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

Étape 2 : Appelez **`CreateKeyStore`** pour créer un magasin de clés de succursale

L'opération suivante crée le magasin de clés de branche qui conservera et protégera vos clés de branche.

C# / .NET

```
var createKeyStoreOutput = keystore.CreateKeyStore(new CreateKeyStoreInput());
```

Java

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

L'`CreateKeyStore`opération crée une table DynamoDB avec le nom de table que vous avez spécifié à l'étape 1 et les valeurs obligatoires suivantes.

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

> **Note**
>
> Vous pouvez créer manuellement la table DynamoDB qui sert de magasin de clés de branche au lieu d'utiliser l'opération. `CreateKeyStore` Si vous choisissez de créer manuellement le magasin de clés de branche, vous devez spécifier les valeurs de chaîne suivantes pour la partition et les clés de tri :
>
> - Clé de partition: `branch-key-id`
> - Clé de tri: `type`

Étape 3 : Appelez **`CreateKey`** pour créer une nouvelle clé de branche active

L'opération suivante crée une nouvelle clé de branche active à l'aide de la clé KMS que vous avez spécifiée à l'étape 1 et ajoute la clé de branche active à la table DynamoDB que vous avez créée à l'étape 2.

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

- Identifiant de clé de branche : définit une personnalisation`branch-key-id`.

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

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
});
```

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();
```

Tout d'abord, l'`CreateKey`opération génère les valeurs suivantes.

- Un identifiant unique universel (UUID) de version 4 pour le `branch-key-id` (sauf si vous avez spécifié un identifiant personnalisé`branch-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'`CreateKey`opération 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 branch key store",
      "kms-arn" : the KMS key ARN,
      "hierarchy-version" : "1",
      "aws-crypto-ec:contextKey": "contextValue"
   },
   "KeyId": "the KMS key ARN you specified in Step 1",
```

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

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

Enfin, l'`CreateKey`opération 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"
}
```

Création d'un trousseau de clés hiérarchique

Pour initialiser le trousseau de clés hiérarchique, vous devez fournir les valeurs suivantes :

- Un nom de magasin clé de succursale

  Nom de la table DynamoDB que vous avez créée pour servir de magasin de clés de succursale.

-

  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. Cette valeur doit être supérieure à zéro. Lorsque la limite de cache TTL expire, l'entrée est expulsée du cache local.

- Un identifiant de clé de branche

  Le `branch-key-id` qui identifie la clé de succursale active dans votre magasin de clés de succursale.

> **Note**
>
> Pour initialiser le trousseau de clés hiérarchique pour une utilisation multilocataire, vous devez spécifier un fournisseur d'ID de clé de branche au lieu d'un. `branch-key-id` Pour plus d'informations, consultez Utilisation du trousseau de clés hiérarchique dans les environnements multilocataires.

- (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 type de cache définit le modèle de threading. Le trousseau de clés hiérarchique fournit trois types de cache compatibles avec les environnements multilocataires : Default,, MultiThreaded. StormTracking

  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.

  Default (Recommended)

  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 clé de branche expire, le cache par défaut empêche plusieurs threads d'appeler AWS KMS et Amazon DynamoDB en notifiant à un thread que l'entrée de matériel 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.

  Pour initialiser votre trousseau de clés hiérarchique avec un cache par défaut, spécifiez la valeur suivante :

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

  C#//.NET

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

```
};
```

Java

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

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 initialiser le trousseau de clés hiérarchique avec le cache par défaut. Pour des personnalisations de cache plus précises, utilisez le StormTracking cache.

MultiThreaded

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 initialiser votre trousseau de clés hiérarchique avec un 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.

C#//.NET

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

Java

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

StormTracking

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 et Amazon DynamoDB en notifiant à un thread que l'entrée de documents 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 initialiser votre trousseau de clés hiérarchique avec un 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.
- 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 un`GetCacheEntry`, 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 : 20 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

C#//.NET

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

Java

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

- (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.

L'exemple suivant initialise un jeu de clés hiérarchique avec une limite de cache TLL de 600 secondes et une capacité d'entrée de 1 000.

C# / .NET

```
// Instantiate the AWS Encryption SDK and material providers
var mpl = new MaterialProviders(new MaterialProvidersConfig());
var esdk =  new ESDK(new AwsEncryptionSdkConfig());

// Instantiate the keyring
var createKeyringInput = new CreateAwsKmsHierarchicalKeyringInput
{
    KeyStore = branchKeyStoreName,
    BranchKeyId = branch-key-id,
    Cache = new CacheType { Default = new DefaultCache{EntryCapacity = 1000} },
    TtlSeconds = 600
};
```

Java

```
final MaterialProviders matProv = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
final CreateAwsKmsHierarchicalKeyringInput keyringInput =
 CreateAwsKmsHierarchicalKeyringInput.builder()
        .keyStore(branchKeyStoreName)
        .branchKeyId(branch-key-id)
        .ttlSeconds(600)
        .cache(CacheType.builder() //OPTIONAL
                .Default(DefaultCache.builder()
                .entryCapacity(1000)
                .build())
        .build();
final Keyring hierarchicalKeyring =
 matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);
```

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. Le trousseau de clés hiérarchique utilise généralement chaque version de clé de branche active 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 dériver de nouvelles clés d'encapsulation. Mais ils peuvent toujours être interrogés et fournir des clés d'encapsulation pour déchiffrer les clés de données qu'ils ont chiffrées lorsqu'ils sont actifs.

Utilisez le `VersionKey` service Key Store 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`.

C# / .NET

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

Java

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

Utilisation du trousseau de clés hiérarchique dans les environnements multilocataires

Vous pouvez utiliser la hiérarchie des clés établie entre les clés de branche actives et leurs clés d'encapsulation dérivées pour prendre en charge les environnements multilocataires en créant une clé de branche pour chaque locataire de votre environnement. Le trousseau de clés hiérarchique chiffre ensuite toutes les données d'un locataire donné avec sa clé de branche distincte. Cela vous permet d'isoler les données des locataires par clé de succursale.

Chaque locataire possède sa propre clé de branche définie par une clé uniquebranch-key-id. Il ne peut y avoir qu'une seule version active de chaque version branch-key-id à la fois.

Avant de pouvoir initialiser votre trousseau de clés hiérarchique pour une utilisation multilocataire, vous devez créer une clé de branche pour chaque locataire et créer un fournisseur d'ID de clé de branche. Utilisez le fournisseur d'identifiant de clé de succursale pour créer un nom convivial branch-key-ids afin de permettre à un locataire de reconnaître plus facilement le nom correctbranch-key-id. Par exemple, le nom convivial vous permet de faire référence à une clé de branche comme tenant1 au 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 message.

Tout d'abord, suivez les étapes 1 et 2 des procédures relatives aux prérequis. Utilisez ensuite les procédures suivantes pour créer une clé de branche pour chaque locataire, créer un fournisseur d'ID de clé de branche et initialiser votre jeu de clés hiérarchique pour une utilisation par plusieurs locataires.

Étape 1 : créer une clé de branche pour chaque locataire de votre environnement

Appelez CreateKey pour chaque locataire.

L'opération suivante crée deux clés de branche à l'aide de la clé KMS que vous avez spécifiée lors de la création de votre service de magasin de clés, et ajoute les clés de branche à la table DynamoDB que vous avez créée pour servir de magasin de clés de branche. La même clé KMS doit protéger toutes les clés de branche.

C# / .NET

```
var branchKeyId1 = keystore.CreateKey(new CreateKeyInput());
```

```
var branchKeyId2 = keystore.CreateKey(new CreateKeyInput());
```

Java

```
CreateKeyOutput branchKeyId1 =
 keystore.CreateKey(CreateKeyInput.builder().build());
CreateKeyOutput branchKeyId2 =
 keystore.CreateKey(CreateKeyInput.builder().build());
```

Étape 2 : créer un fournisseur d'ID de clé de succursale

L'exemple suivant crée un fournisseur d'ID de clé de branche.

C# / .NET

```
var branchKeySupplier =
    new ExampleBranchKeySupplier(branchKeyId1.BranchKeyIdentifier,
 branchKeyId2.BranchKeyIdentifier);
```

Java

```
IBranchKeyIdSupplier branchKeyIdSupplier = new ExampleBranchKeyIdSupplier(
        branchKeyId1.branchKeyIdentifier(), branchKeyId2.branchKeyIdentifier());
```

Étape 3 : Initialisez votre trousseau de clés hiérarchique avec le fournisseur d'ID de clé de branche

Pour initialiser le trousseau de clés hiérarchique, vous devez fournir les valeurs suivantes :

- Un nom de magasin clé de succursale
- Une limite de durée de vie du cache (TTL)
- Un fournisseur d'identifiant de clé de succursale
- (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 type de cache définit le modèle de threading. Le trousseau de clés hiérarchique fournit trois types de cache compatibles avec les environnements multilocataires : Default,, MultiThreaded. StormTracking

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.

Default (Recommended)

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 clé de branche expire, le cache par défaut empêche plusieurs threads d'appeler AWS KMS et Amazon DynamoDB en notifiant à un thread que l'entrée de matériel 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.

Pour initialiser votre trousseau de clés hiérarchique avec un cache par défaut, spécifiez la valeur suivante :

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

C#//.NET

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

Java

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

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 initialiser le trousseau de clés hiérarchique avec le cache par défaut. Pour des personnalisations de cache plus précises, utilisez le StormTracking cache.

MultiThreaded

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 initialiser votre jeu de clés hiérarchique avec un 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.

C#//.NET

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

Java

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

StormTracking

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 et Amazon DynamoDB en notifiant à un thread que l'entrée de documents 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 initialiser votre jeu de clés hiérarchique avec un 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.
- 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 un`GetCacheEntry`, 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 : 20 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

C#//.NET

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

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

Java

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

- (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.

L'exemple suivant initialise un jeu de clés hiérarchique avec le fournisseur d'ID de clé de branche créé à l'étape 2, une limite de cache TLL de 600 secondes et une capacité d'entrée de 1 000.

C# / .NET

```
var createKeyringInput = new CreateAwsKmsHierarchicalKeyringInput
{
    KeyStore = keystore,
    BranchKeyIdSupplier = branchKeySupplier,
    Cache = new CacheType { Default = new DefaultCache{EntryCapacity = 1000} },
    TtlSeconds = 600
};
var keyring = mpl.CreateAwsKmsHierarchicalKeyring(createKeyringInput);
```

Java

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

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

Étape 4 : Créez des noms conviviaux pour chaque clé de branche

L'exemple suivant crée des noms conviviaux pour les deux clés de branche créées à l'étape 1. AWS Encryption SDK Utilise des contextes de chiffrement pour associer le nom convivial que vous définissez au nom associé`branch-key-id`.

C# / .NET

```
// Create encryption contexts for the two branch keys created in Step 1
var encryptionContextA = new Dictionary<string, string>()
{
    // We will encrypt with branchKeyTenantA
    {"tenant", "TenantA"},
    {"encryption", "context"},
    {"is not", "secret"},
    {"but adds", "useful metadata"},
    {"that can help you", "be confident that"},
    {"the data you are handling", "is what you think it is"}
};
var encryptionContextB = new Dictionary<string, string>()
{
    // We will encrypt with branchKeyTenantB
    {"tenant", "TenantB"},
    {"encryption", "context"},
    {"is not", "secret"},
    {"but adds", "useful metadata"},
    {"that can help you", "be confident that"},
    {"the data you are handling", "is what you think it is"}
};
```

```
// Instantiate the AWS Encryption SDK var esdk =  new ESDK(new
 AwsEncryptionSdkConfig());


var encryptInputA = new EncryptInput
{
    Plaintext = plaintext,
    Keyring = keyring,
    // Encrypt with branchKeyId1
    EncryptionContext = encryptionContextA
};

var encryptInputB = new EncryptInput
{
    Plaintext = plaintext,
    Keyring = keyring,
    // Encrypt with branchKeyId2
    EncryptionContext = encryptionContextB
};

var encryptOutput = esdk.Encrypt(encryptInputA);
encryptOutput = esdk.Encrypt(encryptInputB);

// Use the encryption contexts to define friendly names for each branch key
public class ExampleBranchKeySupplier : IBranchKeyIdSupplier
{
    private string branchKeyTenantA;
    private string branchKeyTenantB;

    public ExampleBranchKeySupplier(string branchKeyTenantA, string
 branchKeyTenantB)
    {
        this.branchKeyTenantA = branchKeyTenantA;
        this.branchKeyTenantB = branchKeyTenantB;
    }

    public GetBranchKeyIdOutput GetBranchKeyId(GetBranchKeyIdInput input)
    {
        Dictionary<string, string> encryptionContext = input.EncryptionContext;

        if (!encryptionContext.ContainsKey("tenant"))
        {
            throw new Exception("EncryptionContext invalid, does not contain
 expected tenant key value pair.");
```

```
        }

        string tenant = encryptionContext["tenant"];
        string branchkeyId;

        if (tenant.Equals("TenantA"))
        {
            GetBranchKeyIdOutput output = new GetBranchKeyIdOutput();
            output.BranchKeyId = branchKeyTenantA;
            return output;
        } else if (tenant.Equals("TenantB"))
        {
            GetBranchKeyIdOutput output = new GetBranchKeyIdOutput();
            output.BranchKeyId = branchKeyTenantB;
            return output;
        }
        else
        {
            throw new Exception("Item does not have a valid tenantID.");
        }
    }
}
```

Java

```
// Create encryption context for branchKeyTenantA
Map<String, String> encryptionContextA = new HashMap<>();
encryptionContextA.put("tenant", "TenantA");
encryptionContextA.put("encryption", "context");
encryptionContextA.put("is not", "secret");
encryptionContextA.put("but adds", "useful metadata");
encryptionContextA.put("that can help you", "be confident that");
encryptionContextA.put("the data you are handling", "is what you think it is");

// Create encryption context for branchKeyTenantB
Map<String, String> encryptionContextB = new HashMap<>();
encryptionContextB.put("tenant", "TenantB");
encryptionContextB.put("encryption", "context");
encryptionContextB.put("is not", "secret");
encryptionContextB.put("but adds", "useful metadata");
encryptionContextB.put("that can help you", "be confident that");
encryptionContextB.put("the data you are handling", "is what you think it is");
```

```
// Instantiate the AWS Encryption SDK
final AwsCrypto crypto = AwsCrypto.builder().build();

final CryptoResult<byte[], ?> encryptResultA = crypto.encryptData(keyring,
 plaintext, encryptionContextA);

final CryptoResult<byte[], ?> encryptResultB = crypto.encryptData(keyring,
 plaintext, encryptionContextB);

// Use the encryption contexts to define friendly names for each branch key
public class ExampleBranchKeyIdSupplier implements IBranchKeyIdSupplier {
    private static String branchKeyIdForTenantA;
    private static String branchKeyIdForTenantB;

    public ExampleBranchKeyIdSupplier(String tenant1Id, String tenant2Id) {
        this.branchKeyIdForTenantA = tenant1Id;
        this.branchKeyIdForTenantB = tenant2Id;
    }

    @Override
    public GetBranchKeyIdOutput GetBranchKeyId(GetBranchKeyIdInput input) {

        Map<String, String> encryptionContext = input.encryptionContext();

        if (!encryptionContext.containsKey("tenant"))
        {
            throw new IllegalArgumentException("EncryptionContext invalid, does
 not contain expected tenant key value pair.");
        }

        String tenantKeyId = encryptionContext.get("tenant");
        String branchKeyId;

        if (tenantKeyId.equals("TenantA")) {
            branchKeyId = branchKeyIdForTenantA;
        } else if (tenantKeyId.equals("TenantB")) {
            branchKeyId = branchKeyIdForTenantB;
        } else {
            throw new IllegalArgumentException("Item does not contain valid
 tenant ID");
        }

        return GetBranchKeyIdOutput.builder().branchKeyId(branchKeyId).build();
```

```
    }
}
```

AWS KMS Porte-clés ECDH

> **Important**
>
> Le porte-clés AWS KMS ECDH n'est disponible qu'avec la version 3. x du Kit SDK de chiffrement AWS pour Java. Le porte-clés AWS KMS ECDH est introduit dans la version 1.5.0 de la bibliothèque Material Providers.

Un porte-clés AWS KMS ECDH utilise un accord de clé asymétrique AWS KMS keyspour dériver une clé d'encapsulation symétrique partagée entre deux parties. Tout d'abord, le porte-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 AWS Encryption SDK utilisée (`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 s'assurer que les deux parties utilisent le bon matériel de saisie, elles AWS Encryption SDK utilisent 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 texte chiffré de l'en-tête du message, l'opération échoue. Par exemple, si vous chiffrez des données 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 les données. 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 les données.

Le trousseau de clés AWS KMS ECDH chiffre les données à l'aide d'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

Il AWS Encryption SDK 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 données 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'DeriveSharedSecretautorisation kms : sur la paire de clés KMS asymétrique de l'expéditeur.
- Pour déchiffrer des données à l'aide du schéma d'accord de `KmsPublicKeyDiscovery` clés, vous devez disposer des GetPublicKey autorisations kms : DeriveSharedSecret et kms : sur la paire de clés KMS asymétriques 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 GetPublicKeyopération 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 GetPublicKeyopération 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.

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();
```

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

Lors du déchiffrement, il est recommandé de spécifier les clés qu'ils peuvent utiliser. AWS Encryption SDK 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 jeu de clés AWS KMS ECDH capable de déchiffrer tout message dont la clé publique de la paire de clés KMS spécifiée correspond à la clé publique du destinataire enregistrée dans le texte chiffré du message.

> **Important**
>
> Lorsque vous déchiffrez des messages 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.

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 tout message dont la clé publique de la paire de clés KMS spécifiée correspond à la clé publique du destinataire enregistrée dans le texte chiffré du message.

```
// 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();
```

Porte-clés AES brut

Vous AWS Encryption SDK 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.

Le trousseau de clés AES brut est équivalent et interagit avec la JceMasterKeyclasse du Kit SDK de chiffrement AWS pour Java et la RawMasterKeyclasse du Kit SDK de chiffrement AWS pour Python lorsqu'ils sont utilisés avec une clé de chiffrement AES. Vous pouvez chiffrer des données avec une certaine implémentation et déchiffrer les données avec n'importe quelle autre implémentation à l'aide de la même clé d'encapsulage. Pour plus de détails, consultez Compatibilité du porte-clés.

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 l'en-tête du message chiffré renvoyé par l'opération de chiffrement. 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 de clé et le nom de clé sont équivalents aux champs ID du fournisseur (ou fournisseur) et ID de clé dans le `JceMasterKey` et`RawMasterKey`.
> Les Kit SDK de chiffrement AWS pour C et AWS Encryption SDK pour .NET réservent la valeur de l'espace de noms des `aws-kms` clés pour les clés KMS. N'utilisez pas cette valeur d'espace de noms dans un jeu de clés AES brut ou un jeu de clés RSA brut avec ces bibliothèques.

Si vous créez différents trousseaux de clés pour chiffrer et déchiffrer un message 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 de clé, 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.

C

Pour instancier un trousseau de clés AES brut dans le Kit SDK de chiffrement AWS pour C, utilisez. `aws_cryptosdk_raw_aes_keyring_new()` Pour un exemple complet, consultez raw_aes_keyring.c.

```
struct aws_allocator *alloc = aws_default_allocator();

AWS_STATIC_STRING_FROM_LITERAL(wrapping_key_namespace, "HSM_01");
AWS_STATIC_STRING_FROM_LITERAL(wrapping_key_name, "AES_256_012");

struct aws_cryptosdk_keyring *raw_aes_keyring = aws_cryptosdk_raw_aes_keyring_new(
        alloc, wrapping_key_namespace, wrapping_key_name, aes_wrapping_key,
 wrapping_key_len);
```

C# / .NET

Pour créer un trousseau de clés AES brut dans AWS Encryption SDK .NET, utilisez la `materialProviders.CreateRawAesKeyring()` méthode. Pour un exemple complet, consultez RawAes .cs. KeyringExample

L'exemple suivant utilise la version 4. x du AWS Encryption SDK pour .NET.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk =  new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

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 that determines how your data keys are protected.
var createKeyringInput = new CreateRawAesKeyringInput
{
    KeyNamespace = keyNamespace,
    KeyName = keyName,
    WrappingKey = aesWrappingKey,
    WrappingAlg = AesWrappingAlg.ALG_AES256_GCM_IV12_TAG16
};

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

JavaScript Browser

Le Kit SDK de chiffrement AWS pour JavaScript navigateur obtient ses primitives cryptographiques de l'WebCryptoAPI. Avant de construire le porte-clés, vous devez l'utiliser `RawAesKeyringWebCrypto.importCryptoKey()` pour importer le matériau clé brut dans le WebCrypto backend. Cela garantit que le trousseau de clés est complet même si tous les appels WebCrypto sont asynchrones.

Ensuite, pour instancier un trousseau de clés AES brut, utilisez la méthode. `RawAesKeyringWebCrypto()` Vous devez spécifier l'algorithme d'emballage AES (« suite d'emballage ») en fonction de la longueur de votre matériau clé. Pour un exemple complet, consultez aes_simple.ts (Browser). JavaScript

```
const keyNamespace = 'HSM_01'
const keyName = 'AES_256_012'

const wrappingSuite =
  RawAesWrappingSuiteIdentifier.AES256_GCM_IV12_TAG16_NO_PADDING

/* Import the plaintext AES key into the WebCrypto backend. */
const aesWrappingKey = await RawAesKeyringWebCrypto.importCryptoKey(
  rawAesKey,
  wrappingSuite
)
```

```
const rawAesKeyring = new RawAesKeyringWebCrypto({
  keyName,
  keyNamespace,
  wrappingSuite,
  aesWrappingKey
})
```

JavaScript Node.js

Pour instancier un jeu de clés AES brut dans le fichier Kit SDK de chiffrement AWS pour JavaScript for Node.js, créez une instance de la classe. `RawAesKeyringNode` Vous devez spécifier l'algorithme d'emballage AES (« suite d'emballage ») en fonction de la longueur de votre matériau clé. Pour un exemple complet, consultez aes_simple.ts (Node.js). JavaScript

```
const keyName = 'AES_256_012'
const keyNamespace = 'HSM_01'

const wrappingSuite =
  RawAesWrappingSuiteIdentifier.AES256_GCM_IV12_TAG16_NO_PADDING

const rawAesKeyring = new RawAesKeyringNode({
  keyName,
  keyNamespace,
  aesWrappingKey,
  wrappingSuite,
})
```

Java

Pour instancier un trousseau de clés AES brut dans le Kit SDK de chiffrement AWS pour Java, utilisez. `matProv.CreateRawAesKeyring()`

```
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);
```

Porte-clés RSA bruts

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. Cependant, vous pouvez chiffrer des données avec un jeu de clés RSA brut contenant uniquement une clé publique, et vous pouvez déchiffrer des données avec 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. Certaines implémentations linguistiques du AWS Encryption SDK ne permettent pas de créer un jeu de clés RSA brut avec des clés provenant de paires différentes. D'autres comptent sur vous pour vérifier que vos clés proviennent de la même paire de clés.

Le trousseau de clés RSA brut est équivalent et interagit avec le JceMasterKeyin Kit SDK de chiffrement AWS pour Java et le RawMasterKeyin Kit SDK de chiffrement AWS pour Python lorsqu'il est utilisé avec des clés de chiffrement asymétriques RSA. Vous pouvez chiffrer des données avec une certaine implémentation et déchiffrer les données avec n'importe quelle autre implémentation à l'aide de la même clé d'encapsulage. Pour plus de détails, consultez Compatibilité du porte-clés.

> **Note**
>
> Le porte-clés RSA brut ne prend pas en charge les clés KMS asymétriques. Si vous souhaitez utiliser des clés RSA KMS asymétriques, version 4. x du AWS Encryption SDK pour .NET et la version 3. x des trousseaux de AWS KMS clés de Kit SDK de chiffrement AWS pour Java support qui utilisent le chiffrement symétrique (`SYMMETRIC_DEFAULT`) ou le RSA asymétrique. AWS KMS keys
> Si vous chiffrez des données à l'aide d'un jeu de clés RSA brut qui inclut la clé publique d'une clé RSA KMS, ni le AWS Encryption SDK NI AWS KMS ne peuvent les déchiffrer. Vous ne pouvez pas exporter la clé privée d'une clé KMS AWS KMS asymétrique vers un jeu de clés

RSA brut. L'opération AWS KMS Decrypt ne peut pas déchiffrer le message chiffré renvoyé. AWS Encryption SDK

Lorsque vous créez un jeu de clés RSA brut dans le Kit SDK de chiffrement AWS pour C, veillez à fournir le contenu du fichier PEM qui inclut chaque clé sous forme de chaîne C terminée par un caractère nul, et non sous forme de chemin ou de nom de fichier. Lorsque vous créez un jeu de clés RSA brut JavaScript, soyez conscient des incompatibilités potentielles avec les implémentations d'autres langages.

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 l'en-tête du message chiffré renvoyé par l'opération de chiffrement. 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 de clé et le nom de clé sont équivalents aux champs ID du fournisseur (ou fournisseur) et ID de clé dans le `JceMasterKey` et`RawMasterKey`.
Kit SDK de chiffrement AWS pour C réserve la valeur de l'espace de noms des `aws-kms` clés pour les clés KMS. Ne l'utilisez pas dans un trousseau de clés RAW AES ou dans un trousseau de clés RSA brut avec le. Kit SDK de chiffrement AWS pour C

Si vous créez différents trousseaux de clés pour chiffrer et déchiffrer un message 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` 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 MGF1 avec rembourrage SHA-256. Le mode de rembourrage PKCS1 n'est pris en charge que pour des raisons de rétrocompatibilité.

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

Les exemples suivants montrent 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 MGF1 avec le mode de remplissage SHA-256. Les `RSAPrivateKey` variables `RSAPublicKey` et représentent le matériel clé que vous fournissez.

C

Pour créer un trousseau de clés RSA brut dans le Kit SDK de chiffrement AWS pour C, utilisez. `aws_cryptosdk_raw_rsa_keyring_new`

Lorsque vous créez un jeu de clés RSA brut dans le Kit SDK de chiffrement AWS pour C, veillez à fournir le contenu du fichier PEM qui inclut chaque clé sous forme de chaîne C terminée par un caractère nul, et non sous forme de chemin ou de nom de fichier. Pour un exemple complet, consultez raw_rsa_keyring.c.

```
struct aws_allocator *alloc = aws_default_allocator();

AWS_STATIC_STRING_FROM_LITERAL(key_namespace, "HSM_01");
AWS_STATIC_STRING_FROM_LITERAL(key_name, "RSA_2048_06");

struct aws_cryptosdk_keyring *rawRsaKeyring = aws_cryptosdk_raw_rsa_keyring_new(
```

```
    alloc,
    key_namespace,
    key_name,
    private_key_from_pem,
    public_key_from_pem,
    AWS_CRYPTOSDK_RSA_OAEP_SHA256_MGF1);
```

C# / .NET

Pour instancier un jeu de clés RSA brut dans le domaine .NET, AWS Encryption SDK utilisez la méthode. `materialProviders.CreateRawRsaKeyring()` Pour un exemple complet, voir RawRSA .cs KeyringExample.

L'exemple suivant utilise la version 4. x du AWS Encryption SDK pour .NET.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk =  new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

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 createRawRsaKeyringInput = new CreateRawRsaKeyringInput
{
    KeyNamespace = keyNamespace,
    KeyName = keyName,
    PaddingScheme = PaddingScheme.OAEP_SHA512_MGF1,
    PublicKey = publicKey,
    PrivateKey = privateKey
};

// Create the keyring
var rawRsaKeyring = materialProviders.CreateRawRsaKeyring(createRawRsaKeyringInput);
```

JavaScript Browser

Le Kit SDK de chiffrement AWS pour JavaScript navigateur obtient ses primitives cryptographiques de la WebCryptobibliothèque. Avant de construire le porte-clés, vous devez utiliser `importPublicKey()` et/ou `importPrivateKey()` importer le matériau clé brut dans le WebCrypto backend. Cela garantit que le trousseau de clés est complet même si tous les appels WebCrypto sont asynchrones. L'objet utilisé par les méthodes d'importation inclut l'algorithme d'encapsulage et son mode de remplissage.

Après avoir importé le matériel clé, utilisez la `RawRsaKeyringWebCrypto()` méthode pour instancier le trousseau de clés. Lorsque vous créez un jeu de clés RSA brut JavaScript, soyez conscient des incompatibilités potentielles avec les implémentations d'autres langages.

Pour un exemple complet, consultez rsa_simple.ts (Browser). JavaScript

```
const privateKey = await RawRsaKeyringWebCrypto.importPrivateKey(
  privateRsaJwKKey
)

const publicKey = await RawRsaKeyringWebCrypto.importPublicKey(
  publicRsaJwKKey
)

const keyNamespace = 'HSM_01'
const keyName = 'RSA_2048_06'

const keyring = new RawRsaKeyringWebCrypto({
  keyName,
  keyNamespace,
  publicKey,
  privateKey,
})
```

JavaScript Node.js

Pour instancier un jeu de clés RSA brut dans Kit SDK de chiffrement AWS pour JavaScript Node.js, créez une nouvelle instance de la classe. `RawRsaKeyringNode` Le `wrapKey` paramètre contient la clé publique. Le `unwrapKey` paramètre contient la clé privée. Le `RawRsaKeyringNode` constructeur calcule un mode de remplissage par défaut pour vous, bien que vous puissiez spécifier un mode de remplissage préféré.

Lorsque vous créez un trousseau de clés RSA brut JavaScript, soyez conscient des incompatibilités potentielles avec les implémentations d'autres langages.

Pour un exemple complet, consultez rsa_simple.ts (Node.js). JavaScript

```
const keyNamespace = 'HSM_01'
const keyName = 'RSA_2048_06'

const keyring = new RawRsaKeyringNode({ keyName, keyNamespace, rsaPublicKey,
 rsaPrivateKey})
```

Java

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

Porte-clés ECDH bruts

> **Important**
>
> Le porte-clés Raw ECDH n'est disponible qu'avec la version 3. x du Kit SDK de chiffrement AWS pour Java. Le porte-clés ECDH brut est introduit dans 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 AWS Encryption SDK utilisée (`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 contenu clé. Pour s'assurer que les deux parties utilisent le bon matériel clé, elles AWS Encryption SDK utilisent 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 texte chiffré de l'en-tête du message, l'opération échoue. Par exemple, si vous chiffrez des données 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 les données. 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 les données.

Le trousseau de clés ECDH brut chiffre les données 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. Il AWS Encryption SDK prend en charge les spécifications de cuves elliptiques 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 clés :`RawPrivateKeyToStaticPublicKey`,, et`EphemeralPrivateKeyToStaticPublicKey`. `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 clé permet de chiffrer et de déchiffrer des données.

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`

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);
}
```

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 les messages. Pour déchiffrer les messages 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

PublicKeyDiscoveryclés 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`

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);
}
```

PublicKeyDiscovery

Lors du déchiffrement, il est recommandé de spécifier les clés d'encapsulation qu'ils peuvent utiliser. AWS Encryption SDK Pour suivre cette bonne pratique, utilisez un jeu 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 trousseau de clés ECDH brut capable de déchiffrer tout message dont la clé publique spécifiée correspond à la clé publique du destinataire enregistrée dans le texte chiffré du message. Ce schéma d'accord clé ne peut que déchiffrer les messages.

> **Important**
>
> Lorsque vous déchiffrez des messages 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`

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 message dont la clé publique de la clé privée spécifiée correspond à la clé publique du destinataire enregistrée dans le texte chiffré du message.

```
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);
}
```

Porte-clés multiples

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'encapsulage 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'encapsulage 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'encapsulage du porte-clés multiple. Si vous créez un porte-clés multiple sans porte-clés générateur vous pouvez l'utiliser pour déchiffrer les données, mais pas pour les chiffrer. 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, il AWS Encryption SDK utilise les trousseaux de clés pour essayer de déchiffrer l'une des clés de données cryptées. Les porte-clés sont appelés dans l'ordre dans lequel ils sont spécifiées dans le porte-clés multiple. Le traitement s'arrête dès qu'une clé d'un porte-clés peut déchiffrer une clé de données chiffrée.

À partir de la version 1.7. x, lorsqu'une clé de données cryptée est cryptée sous un trousseau de clés AWS Key Management Service (AWS KMS) (ou un fournisseur de clés principales), le paramètre de l' AWS Encryption SDK opération AWS KMS Decrypt transmet toujours l'ARN de la clé AWS KMS key au `KeyId` paramètre de l'opération de déchiffrement. Il s'agit d'une AWS KMS bonne pratique qui garantit que vous déchiffrez la clé de données cryptée avec la clé d'encapsulation que vous souhaitez utiliser.

Pour obtenir un exemple pratique d'un porte-clés multiple, veuillez consulter :

- C : multi_keyring.cpp
- C#//.NET : .cs MultiKeyringExample
- JavaScript Node.js : multi_keyring.ts
- JavaScript Navigateur : multi_keyring.ts

- Java : MultiKeyringExample.java

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.

C

```
/* Define an AWS KMS keyring. For details, see string.cpp */
struct aws_cryptosdk_keyring *kms_keyring =
 Aws::Cryptosdk::KmsKeyring::Builder().Build(example_key);

// Define a Raw AES keyring. For details, see raw_aes_keyring.c */
struct aws_cryptosdk_keyring *aes_keyring = aws_cryptosdk_raw_aes_keyring_new(
        alloc, wrapping_key_namespace, wrapping_key_name, wrapping_key,
 AWS_CRYPTOSDK_AES256);
```

C# / .NET

```
// Define an AWS KMS keyring. For details, see AwsKmsKeyringExample.cs.
var kmsKeyring = materialProviders.CreateAwsKmsKeyring(createKmsKeyringInput);

// Define a Raw AES keyring. For details, see RawAESKeyringExample.cs.
var aesKeyring = materialProviders.CreateRawAesKeyring(createAesKeyringInput);
```

JavaScript Browser

```
const clientProvider = getClient(KMS, { credentials })

// Define an AWS KMS keyring. For details, see kms_simple.ts.
const kmsKeyring = new KmsKeyringBrowser({ generatorKeyId: exampleKey })

// Define a Raw AES keyring. For details, see aes_simple.ts.
const aesKeyring = new RawAesKeyringWebCrypto({ keyName, keyNamespace,
 wrappingSuite, masterKey })
```

JavaScript Node.js

```
// Define an AWS KMS keyring. For details, see kms_simple.ts.
const kmsKeyring = new KmsKeyringNode({ generatorKeyId: exampleKey })
```

```
// Define a Raw AES keyring. For details, see raw_aes_keyring_node.ts.
const aesKeyring = new RawAesKeyringNode({ keyName, keyNamespace, wrappingSuite,
 unencryptedMasterKey })
```

Java

```
// Define 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);

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

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.

C

Dans le constructeur d'un porte-clés multiple en C, vous spécifiez uniquement son porte-clés générateur.

```
struct aws_cryptosdk_keyring *multi_keyring = aws_cryptosdk_multi_keyring_new(alloc,
 kms_keyring);
```

Pour ajouter un porte-clés enfant à votre porte-clés multiple, utilisez la méthode `aws_cryptosdk_multi_keyring_add_child`. Vous devez appeler la méthode une fois pour chaque porte-clés enfant que vous ajoutez.

```
// Add the Raw AES keyring (C only)
aws_cryptosdk_multi_keyring_add_child(multi_keyring, aes_keyring);
```

C# / .NET

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

```
var createMultiKeyringInput = new CreateMultiKeyringInput
{
    Generator = kmsKeyring,
    ChildKeyrings = new List<IKeyring>() {aesKeyring}
};

var multiKeyring = materialProviders.CreateMultiKeyring(createMultiKeyringInput);
```

JavaScript Browser

JavaScript les porte-clés multiples sont immuables. Le constructeur de JavaScript porte-clés multiples vous permet de spécifier le porte-clés générateur et plusieurs porte-clés enfants.

```
const clientProvider = getClient(KMS, { credentials })

const multiKeyring = new MultiKeyringWebCrypto(generator: kmsKeyring, children:
 [aesKeyring]);
```

JavaScript Node.js

JavaScript les porte-clés multiples sont immuables. Le constructeur de JavaScript porte-clés multiples vous permet de spécifier le porte-clés générateur et plusieurs porte-clés enfants.

```
const multiKeyring = new MultiKeyringNode(generator: kmsKeyring, children:
 [aesKeyring]);
```

Java

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

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

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

Langages de programmation AWS Encryption SDK

Le kit AWS Encryption SDK est disponible pour les langages de programmation suivants. Toutes les implémentations de langage 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. En outre, lors du chiffrement et du déchiffrement, vous devez utiliser des porte-clés compatibles, ou des clés principales et des fournisseurs de clés principales. Pour plus d'informations, consultez the section called “Compatibilité du porte-clés”.

Rubriques

Kit SDK de chiffrement AWS pour C

Le kit Kit SDK de chiffrement AWS pour C est conçu pour fournir une bibliothèque de chiffrement côté client pour les développeurs qui écrivent des applications en C. Il sert également de base pour les implémentations du kit AWS Encryption SDK dans les langages de programmation de niveau plus élevé.

Comme toutes les implémentations du kit AWS Encryption SDK, le kit Kit SDK de chiffrement AWS pour C propose les fonctions avancées en matière de protection des données. Il s'agit notamment du chiffrement d'enveloppe, de données authentifiées supplémentaires (données AAD) et de suites d'algorithmes de clés symétriques sécurisées, authentifiées, comme les clés AES-GCM 256 bits avec dérivation de clés et signature.

Toutes les implémentations spécifiques à un langage du kit AWS Encryption SDK sont complètement interopérables. Par exemple, vous pouvez chiffrer les données avec leKit SDK de chiffrement AWS pour Cet déchiffrer avectoute implémentation linguistique prise en charge, y compris leAWSCLI de chiffrement.

LeKit SDK de chiffrement AWS pour Cnécessite leAWS SDK for C++Interaction avecAWS Key Management Service(AWS KMS). Vous devez l'utiliser uniquement si vous utilisez l'option optionnelleAWS KMSPorte-clés. Néanmoins, le kit AWS Encryption SDK ne nécessite aucun autre service AWS KMS ou AWS.

En savoir plus

- Pour plus de détails sur la programmation avec leKit SDK de chiffrement AWS pour C, voir leExemples de C, leexemplesdans leaws-encryption-sdkDépôt -csurGitHub, et leKit SDK de chiffrement AWS pour CDocumentation sur les API.
- Pour une discussion sur la façon d'utiliser leKit SDK de chiffrement AWS pour Cpour chiffrer les données afin de pouvoir les déchiffrer en plusieursRégions AWS, voirComment déchiffrer des textes de chiffrement dans plusieurs régions avec leAWS Encryption SDKen Cdans leAWSBlog sur la sécurité.

Rubriques

Installation du Kit SDK de chiffrement AWS pour C

Installez la dernière version duKit SDK de chiffrement AWS pour C.

> **Note**
>
> Toutes les versions duKit SDK de chiffrement AWS pour Cles versions antérieures à 2.0.0 se trouvent dans leend-of-supportphase.
> Vous pouvez la mettre à jour en toute sécurité depuis la version 2.0xet plus tard vers la dernière version duKit SDK de chiffrement AWS pour Csans aucune modification du code ou des données. Cependant,nouvelles fonctions de sécuritéintroduit dans la version 2.0.xne sont pas rétrocompatibles. Pour effectuer une mise à jour à partir de versions antérieures à 1.7xvers la version 2.0.xet plus tard, vous devez d'abord la mettre à jour vers la dernière version 1.xversion duKit SDK de chiffrement AWS pour C. Pour plus d'informations, consultez.Migrer votreAWS Encryption SDK

Vous trouverez des instructions détaillées pour l'installation et la construction duKit SDK de chiffrement AWS pour Cdans lefichier READMEde laaws-encryption-sdk-créférentiel. Il inclut des instructions pour créer sur les plateformes Amazon Linux, Ubuntu, macOS et Windows.

Avant de commencer, décidez si vous souhaitez utiliserAWS KMSporte-clésdans leAWS Encryption SDK. Si vous utilisez unAWS KMSporte-clés, vous devez installer leAWS SDK for C++. Dans laAWSUn SDK est nécessaire pour interagir avecAWS Key Management Service(AWS KMS). Lorsque vous utilisezAWS KMSporte-clés, lesAWS Encryption SDKles usagesAWS KMSpour générer et protéger les clés de chiffrement qui protègent vos données.

Vous n'avez pas besoin d'installer leAWS SDK for C++si vous utilisez un autre type de trousseau de clés, tel qu'un trousseau AES brut, un trousseau RSA brut ou un trousseau multiple qui n'inclut pasAWS KMSporte-clés. Toutefois, lorsque vous utilisez un type de trousseau de clés brut, vous devez générer et protéger vos propres clés d'encapsulation brutes.

Pour vous aider à choisir les types de trousseaux de clés à utiliser, voirthe section called “Choisir un porte-clés”.

Si vous rencontrez des difficultés lors de l'installation,signaler un problèmedans le`aws-encryption-sdk-c`référentiel ou utilisez l'un des liens de commentaires sur cette page.

Utilisation des Kit SDK de chiffrement AWS pour C

Cette rubrique explique certaines fonctions du kit Kit SDK de chiffrement AWS pour C qui ne sont pas prises en charge par d'autres implémentations de langage de programmation.

Les exemples de cette section montrent comment utiliserversion 2.0.h/24, j/7et plus tard duKit SDK de chiffrement AWS pour C. Pour obtenir des exemples utilisant des versions antérieures, recherchez votre version dans leVersionsListe deréférentiel aws-cryption-sdk-csur GitHub.

Pour plus de détails sur la programmation avec le kit Kit SDK de chiffrement AWS pour C, consultez les exemples C, les exemples du référentiel aws-encryption-sdk-c sur GitHub et la documentation de l'API Kit SDK de chiffrement AWS pour C.

Voir aussi : Utilisation des porte-clés

Rubriques

- Modèles de chiffrement et de déchiffrement des données
- Comptage des références

Modèles de chiffrement et de déchiffrement des données

Lorsque vous utilisez le kit Kit SDK de chiffrement AWS pour C, vous suivez un modèle similaire à ceci : créez un porte-clés, créez un CMM qui utilise le porte-clés, créez une session qui utilise le CMM (et le porte-clés), puis traitez la session.

1. Chargez les chaînes d'erreur.

   Appelez leaws_cryptosdk_load_error_strings()dans votre code C ou C++. Il charge des informations d'erreur très utiles pour le débogage.

   Vous n'avez besoin de l'appeler qu'une seule fois, par exemple dans votremainméthode.

   ```
   /* Load error strings for debugging */
   aws_cryptosdk_load_error_strings();
   ```

2. Créez un porte-clés.

   Configurez votre porte-clés avec les clés d'encapsulage que vous souhaitez utiliser pour chiffrer vos clés de données. Cet exemple utilise unAWS KMSPorte-clésavec uneAWS KMS key, mais vous pouvez utiliser n'importe quel type de porte-clés à sa place.

   Pour identifier unAWS KMS keydans un porte-clés de chiffrement dans leKit SDK de chiffrement AWS pour C, spécifiez unARN de cléouARN d'alias. Dans un porte-clés de déchiffrement, vous devez utiliser un ARN de clé. Pour plus d'informations, consultez Identification AWS KMS keys dans un AWS KMS porte-clés.

   ```
   const char * KEY_ARN = "arn:aws:kms:us-
   west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
   struct aws_cryptosdk_keyring *kms_keyring =
          Aws::Cryptosdk::KmsKeyring::Builder().Build(KEY_ARN);
   ```

3. Créez une session.

   Dans le kit Kit SDK de chiffrement AWS pour C, vous utilisez une session pour chiffrer un seul message en texte brut ou déchiffrer un seul message en texte chiffré, quelle que soit sa taille. La session conserve l'état du message tout au long de son traitement.

   Configurez votre session avec un allocateur, un porte-clés et un mode : AWS_CRYPTOSDK_ENCRYPT ou AWS_CRYPTOSDK_DECRYPT. Si vous devez modifier le mode de la session, utilisez la méthode aws_cryptosdk_session_reset.

Lorsque vous créez une session avec un porte-clés, leKit SDK de chiffrement AWS pour Ccrée automatiquement un gestionnaire de matériaux de chiffrement (CMM) par défaut pour vous. Vous n'avez pas besoin de créer, de maintenir ou de détruire cet objet.

Par exemple, la session suivante utilise l'allocateur et le porte-clés qui a été défini à l'étape 1. Lorsque vous chiffrez des données, le mode est `AWS_CRYPTOSDK_ENCRYPT`.

```
struct aws_cryptosdk_session * session =
 aws_cryptosdk_session_new_from_keyring_2(allocator, AWS_CRYPTOSDK_ENCRYPT,
 kms_keyring);
```

4. Chiffrez ou déchiffrez les données.

Pour traiter les données de la session, utilisez la méthode `aws_cryptosdk_session_process`. Si le tampon d'entrée est suffisamment grand pour contenir tout le texte brut et que le tampon de sortie est suffisamment grand pour contenir le texte chiffré entier, vous pouvez appeler`aws_cryptosdk_session_process_full`. Toutefois, si vous avez besoin de gérer les données de streaming, vous pouvez appeler`aws_cryptosdk_session_process`en boucle. Pour de plus amples informations, vous pouvez consulter l'exemple file_streaming.cpp. L`aws_cryptosdk_session_process_full`est introduit dansAWS Encryption SDKversions 1.9.h/24, j/7et 2.2.h/24, j/7.

Lorsque la session est configurée pour chiffrer des données, les champs de texte brut décrivent les champs d'entrée et les champs du texte chiffré décrivent la sortie. Le champ `plaintext` contient le message que vous souhaitez chiffrer et le champ `ciphertext` obtient le message chiffré renvoyé par la méthode de chiffrement

```
/* Encrypting data */
aws_cryptosdk_session_process_full(session,
                                   ciphertext,
                                   ciphertext_buffer_size,
                                   &ciphertext_length,
                                   plaintext,
                                   plaintext_length)
```

Lorsque la session est configurée pour déchiffrer des données, les champs de texte chiffré décrivent les champs d'entrée et les champs du texte brut décrivent la sortie. Le champ

`ciphertext` contient le message chiffré renvoyé par la méthode de chiffrement, et le champ `plaintext` obtient le message en texte brut renvoyé par la méthode de déchiffrement.

Pour déchiffrer les données, appelez la méthode `aws_cryptosdk_session_process_full`.

```
/* Decrypting data */
aws_cryptosdk_session_process_full(session,
                                   plaintext,
                                   plaintext_buffer_size,
                                   &plaintext_length,
                                   ciphertext,
                                   ciphertext_length)
```

Comptage des références

Pour éviter les fuites de mémoire, assurez-vous de libérer vos références à tous les objets que vous créez lorsque vous en avez terminé avec eux. Dans le cas contraire, des fuites de mémoire se produisent. Le kit SDK fournit des méthodes pour faciliter cette tâche.

Chaque fois que vous créez un objet parent avec l'un des objets enfants suivants, l'objet parent obtient et conserve une référence à l'objet enfant, comme suit :

- Un porte-clés, tel que la création d'une session avec un porte-clés
- Un par défautGestionnaire de matériaux de chiffrement(CMM), tel que la création d'une session ou d'un CMM personnalisé avec un CMM par défaut
- Un cache de clés de données, tel que la création d'un CMM de mise en cache avec un porte-clés et un cache

Sauf si vous avez besoin d'une référence indépendante à l'objet enfant, vous pouvez libérer votre référence à l'objet enfant dès que vous créez l'objet parent. La référence restante à l'objet enfant est libérée lorsque l'objet parent est détruit. Ce modèle garantit que vous conservez la référence à chaque objet uniquement pendant le délai nécessaire et empêche les fuites de mémoire en raison de références non libérées.

Vous êtes uniquement responsable de la libération des références aux objets enfants que vous créez explicitement. Vous n'êtes pas responsable de la gestion des références aux objets créés par le SDK pour vous. Si le SDK crée un objet, tel que le CMM par défaut que

leaws_cryptosdk_caching_cmm_new_from_keyringajoute à une session, le SDK gère la création et la destruction de l'objet et de ses références.

Dans l'exemple suivant, lorsque vous créez une session avec un porte-clés, la session obtient une référence au porte-clés et conserve cette référence jusqu'à ce que la session soit détruite. Si vous n'avez pas besoin de conserver une référence supplémentaire au porte-clés, vous pouvez utiliser la méthode `aws_cryptosdk_keyring_release` pour libérer l'objet de porte-clés dès que la session est créée. Cette méthode diminue le nombre de références pour le porte-clés. La référence de la session au porte-clés est libérée lorsque vous appelez `aws_cryptosdk_session_destroy` pour détruire la session.

```
// The session gets a reference to the keyring.
struct aws_cryptosdk_session *session =
 aws_cryptosdk_session_new_from_keyring_2(alloc, AWS_CRYPTOSDK_ENCRYPT, keyring);

// After you create a session with a keyring, release the reference to the keyring
 object.
aws_cryptosdk_keyring_release(keyring);
```

Pour des tâches plus complexes, telles que la réutilisation d'un porte-clés pour plusieurs sessions ou la spécification d'une suite d'algorithmes dans un CMM, vous devrez peut-être conserver une référence indépendante à l'objet. Si c'est le cas, n'appelez pas les méthodes de publication immédiatement. Au lieu de cela, libérez vos références lorsque vous n'utilisez plus les objets, en plus de détruire la session.

Cette technique de comptage des références fonctionne également lorsque vous utilisez d'autres CMM, comme le CMM de mise en cache pourMise en cache des clés de données. Lorsque vous créez une CMM de mise en cache à partir d'un cache et d'un porte-clés, le CMM de mise en cache obtient une référence aux deux objets. Sauf si vous en avez besoin pour une autre tâche, vous pouvez libérer vos références indépendantes au cache et au porte-clés dès que le CMM de mise en cache est créé. Ensuite, lorsque vous créez une session avec le CMM de mise en cache, vous pouvez libérer votre référence au CMM de mise en cache.

Notez que vous êtes uniquement responsable de la publication des références aux objets que vous créez explicitement. Les objets que les méthodes créent pour vous, tels que le CMM par défaut qui sous-tend le CMM de mise en cache, sont gérés par la méthode.

```
/ Create the caching CMM from a cache and a keyring.
```

```
struct aws_cryptosdk_cmm *caching_cmm =
 aws_cryptosdk_caching_cmm_new_from_keyring(allocator, cache, kms_keyring, NULL, 60,
 AWS_TIMESTAMP_SECS);

// Release your references to the cache and the keyring.
aws_cryptosdk_materials_cache_release(cache);
aws_cryptosdk_keyring_release(kms_keyring);

// Create a session with the caching CMM.
struct aws_cryptosdk_session *session = aws_cryptosdk_session_new_from_cmm_2(allocator,
 AWS_CRYPTOSDK_ENCRYPT, caching_cmm);

// Release your references to the caching CMM.
aws_cryptosdk_cmm_release(caching_cmm);

// ...

aws_cryptosdk_session_destroy(session);
```

Exemples Kit SDK de chiffrement AWS pour C

Les exemples suivants montrent comment utiliser le kit Kit SDK de chiffrement AWS pour C afin de chiffrer et de déchiffrer des données.

Les exemples de cette section montrent comment utiliser les versions 2.0.h/24, j/7et plus tard duKit SDK de chiffrement AWS pour C. Pour obtenir des exemples utilisant des versions antérieures, recherchez votre version dans leVersionsliste desaws-encryption-sdk-crepository surGitHub.

Lorsque vous installez et générez le kit Kit SDK de chiffrement AWS pour C, le code source de ces éléments et d'autres exemples sont inclus dans le sous-répertoire `examples` et compilés et intégrés dans le répertoire `build`. Vous pouvez également les trouver dans leexemplessous-répertoire duaws-encryption-sdk-crepository surGitHub.

Rubriques

- Chiffrement et déchiffrement de chaînes

Chiffrement et déchiffrement de chaînes

L'exemple suivant montre comment utiliser le kit Kit SDK de chiffrement AWS pour C pour chiffrer et déchiffrer une chaîne.

Cet exemple présente leAWS KMSPorte-clés, un type de porte-clés qui utilise unAWS KMS keydansAWS Key Management Service(AWS KMS)pour générer et chiffrer des clés de données. L'exemple inclut du code écrit en C++. LeKit SDK de chiffrement AWS pour Cnécessite laAWS SDK for C++AppelezAWS KMSlors de l'utilisationAWS KMSporte-clés. Si vous utilisez un porte-clés qui n'interagit pas avecAWS KMS, comme un porte-clés AES brut, un porte-clés RSA brut ou un porte-clés multi-clés qui ne comprend pas deAWS KMSporte-clés, leAWS SDK for C++n'est pas obligatoire.

Pour obtenir de l'aide pour créer unAWS KMS key, voirCréation de clésdans leAWS Key Management ServiceManuel du développeur. Pour obtenir de l'aide pour identifier leAWS KMS keysdans unAWS KMSporte-clés, voirIdentification AWS KMS keys dans un AWS KMS porte-clés.

Consultez l'exemple de code complet : string.cpp

Rubriques

- Chiffrement d'une chaîne
- Déchiffrement d'une chaîne

Chiffrement d'une chaîne

La première partie de cet exemple utilise unAWS KMSporte-clés avec unAWS KMS keypour chiffrer une chaîne en texte brut.

Étape 1. Chargez des chaînes d'erreur.

Appelez le`aws_cryptosdk_load_error_strings()`dans votre code C ou C++. Il charge des informations d'erreur très utiles pour le débogage.

Vous n'avez besoin de l'appeler qu'une seule fois, par exemple dans votre`main`méthode.

```
/* Load error strings for debugging */
aws_cryptosdk_load_error_strings();
```

Étape 2 : Construction du porte-clés.

Création d'unAWS KMSporte-clés pour chiffrement. Le porte-clés de cet exemple est configuré avec unAWS KMS key, mais vous pouvez configurer unAWS KMSporte-clés avec plusieursAWS KMS keys, y comprisAWS KMS keysdans différentsRégions AWSet d'autres comptes.

Pour identifier unAWS KMS keydans un porte-clés de chiffrement dans leKit SDK de chiffrement AWS pour C, spécifiez unARN de cléouARN d'alias. Dans un porte-clés de déchiffrement, vous devez utiliser un ARN de clé. Pour plus d'informations, consultez Identification AWS KMS keys dans un AWS KMS porte-clés.

Identification AWS KMS keys dans un AWS KMS porte-clés

Lorsque vous créez un porte-clés avec plusieursAWS KMS keys, vous spécifiez leAWS KMS keyutilisée pour générer et chiffrer la clé de données en texte brut et, si vous le souhaitez, un ensemble de supplémentairesAWS KMS keysqui chiffrent la même clé de données en texte brut. Dans ce cas, vous spécifiez uniquement la du générateurAWS KMS key.

Avant d'exécuter ce code, remplacez l'exemple d'ARN de clé par un ARN valide.

```
const char * key_arn = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

struct aws_cryptosdk_keyring *kms_keyring =
       Aws::Cryptosdk::KmsKeyring::Builder().Build(key_arn);
```

Étape 3 : Créez une session.

Créez une session à l'aide de l'allocateur, d'un énumérateur de mode et du porte-clés.

Chaque session nécessite un mode : soit `AWS_CRYPTOSDK_ENCRYPT` pour chiffrer, soit `AWS_CRYPTOSDK_DECRYPT` pour déchiffrer. Pour modifier le mode d'une session existante, utilisez la méthode `aws_cryptosdk_session_reset`.

Après avoir créé une session avec le porte-clés, vous pouvez libérer votre référence au porte-clés à l'aide de la méthode fournie par le kit SDK. La session conserve une référence à l'objet de porte-clés pendant sa durée de vie. Les références aux objets de porte-clés et de session sont libérées lorsque vous détruisez la session. Cette technique de comptage de référence permet d'éviter les fuites de mémoire et d'empêcher la libération des objets pendant leur utilisation.

```
struct aws_cryptosdk_session *session =
       aws_cryptosdk_session_new_from_keyring_2(alloc, AWS_CRYPTOSDK_ENCRYPT,
 kms_keyring);

/* When you add the keyring to the session, release the keyring object */
aws_cryptosdk_keyring_release(kms_keyring);
```

Étape 4 : Définition du contexte de chiffrement.

Un contexte de chiffrement est un ensemble de données authentifiées supplémentaires non secrètes et arbitraires. Lorsque vous fournissez un contexte de chiffrement lors du chiffrement, le kit AWS Encryption SDK le lie de façon cryptographique au texte chiffré de sorte que le même contexte de chiffrement est requis pour déchiffrer les données. L'utilisation d'un contexte de chiffrement est facultative, mais nous vous le recommandons dans le cadre des bonnes pratiques.

Tout d'abord, créez une table de hachage qui inclut le contexte de chiffrement des chaînes.

```
/* Allocate a hash table for the encryption context */
int set_up_enc_ctx(struct aws_allocator *alloc, struct aws_hash_table *my_enc_ctx)

// Create encryption context strings
AWS_STATIC_STRING_FROM_LITERAL(enc_ctx_key1, "Example");
AWS_STATIC_STRING_FROM_LITERAL(enc_ctx_value1, "String");
AWS_STATIC_STRING_FROM_LITERAL(enc_ctx_key2, "Company");
AWS_STATIC_STRING_FROM_LITERAL(enc_ctx_value2, "MyCryptoCorp");

// Put the key-value pairs in the hash table
aws_hash_table_put(my_enc_ctx, enc_ctx_key1, (void *)enc_ctx_value1, &was_created)
aws_hash_table_put(my_enc_ctx, enc_ctx_key2, (void *)enc_ctx_value2, &was_created)
```

Obtenez un pointeur réversible dans le contexte de chiffrement de la session. Ensuite, utilisez la fonction `aws_cryptosdk_enc_ctx_clone` pour copier le contexte de chiffrement dans la session. Conservez la copie dans `my_enc_ctx` afin de pouvoir valider la valeur après le déchiffrement des données.

Le contexte de chiffrement fait partie de la session. Il ne s'agit pas d'un paramètre transmis à la fonction du processus de session. Cela garantit que le même contexte de chiffrement est utilisé pour chaque segment d'un message, même si la fonction du processus de session est appelée plusieurs fois pour chiffrer l'intégralité du message.

```
struct aws_hash_table *session_enc_ctx =
 aws_cryptosdk_session_get_enc_ctx_ptr_mut(session);

aws_cryptosdk_enc_ctx_clone(alloc, session_enc_ctx, my_enc_ctx)
```

Étape 5 : Chiffrement de la chaîne.

Pour chiffrer la chaîne en texte brut, utilisez la méthode `aws_cryptosdk_session_process_full` avec la session dans le mode de chiffrement. Cette méthode, introduite dansAWS Encryption SDKversions 1.9.h/24, j/7et 2.2.h/24, j/7, est conçu pour le chiffrement et le déchiffrement sans streaming. Pour gérer les données en streaming, appelez le`aws_cryptosdk_session_process`en boucle.

Lors du chiffrement, les champs de texte brut sont des champs d'entrée ; les champs de texte chiffré sont des champs de sortie. Une fois le traitement terminé, le champ `ciphertext_output` contient le message chiffré, y compris le texte chiffré réel, des clés de données chiffrées et le contexte de chiffrement. Vous pouvez déchiffrer ce message à l'aide du kit AWS Encryption SDK pour n'importe quel langage de programmation pris en charge.

```
/* Gets the length of the plaintext that the session processed */
size_t ciphertext_len_output;
if (AWS_OP_SUCCESS != aws_cryptosdk_session_process_full(session,
                                  ciphertext_output,
                                  ciphertext_buf_sz_output,
                                  &ciphertext_len_output,
                                  plaintext_input,
                                  plaintext_len_input)) {
    aws_cryptosdk_session_destroy(session);
    return 8;
}
```

Étape 6 : Nettoyez la session.

La dernière étape détruit la session, y compris les références au CMM et le porte-clés.

Si vous préférez, au lieu de détruire la session, vous pouvez la réutiliser avec le même porte-clés et la même clé CMM pour déchiffrer la chaîne, ou pour chiffrer ou pour chiffrer ou déchiffrer d'autres messages. Pour utiliser la session pour le déchiffrement, utilisez la méthode `aws_cryptosdk_session_reset` pour modifier le mode vers `AWS_CRYPTOSDK_DECRYPT`.

Déchiffrement d'une chaîne

La deuxième partie de cet exemple consiste à déchiffrer un message chiffré qui contient le texte chiffré de la chaîne d'origine.

Étape 1 : Chargez des chaînes d'erreur.

Appelez l'`aws_cryptosdk_load_error_strings()`dans votre code C ou C++. Il charge des informations d'erreur très utiles pour le débogage.

Vous n'avez besoin de l'appeler qu'une seule fois, par exemple dans votre`main`méthode.

```
/* Load error strings for debugging */
aws_cryptosdk_load_error_strings();
```

Étape 2 : Construction du porte-clés.

Lorsque vous déchiffrer des données dans AWS KMS, vous transmettez message chiffré renvoyé par l'API de chiffrement. LeAPI de déchiffrementne prend pas deAWS KMS keyen entrée. En revanche, AWS KMS utilise la même AWS KMS key afin de déchiffrer le texte chiffré utilisé pour le chiffrement. Cependant, leAWS Encryption SDKvous permet de spécifier unAWS KMSporte-clés avecAWS KMS keyslors du chiffrement et du déchiffrement.

Lors du déchiffrement, vous pouvez configurer un porte-clés avec uniquement les AWS KMS keys que vous souhaitez utiliser pour déchiffrer le message chiffré. Par exemple, vous pouvez créer un porte-clés avec uniquement la AWS KMS key qui est utilisée par un rôle dans votre organisation. LeAWS Encryption SDKn'utilisera jamais unAWS KMS keysauf si elle s'affiche dans le porte-clés de déchiffrement. Si le kit SDK ne peut pas déchiffrer les clés de données chiffrées à l'aide de AWS KMS keys du porte-clés que vous fournissez, soit parce qu'aucune des AWS KMS keys du porte-clés n'a été utilisée pour chiffrer les clés de données, soit parce que le mandataire n'a pas l'autorisation d'utiliser les AWS KMS keys du porte-clés pour le déchiffrement, l'appel de déchiffrement échoue.

Lorsque vous spécifiez unAWS KMS keypour un porte-clés de déchiffrement, vous devez utiliser sonARN de clé. ARN d'aliassont autorisés uniquement dans les porte-clés de chiffrement. Pour obtenir de l'aide pour identifier leAWS KMS keysdans unAWS KMSporte-clés, voirIdentification AWS KMS keys dans un AWS KMS porte-clés.

Dans cet exemple, nous spécifions un porte-clés configuré avec le mêmeAWS KMS keyutilisée pour chiffrer la chaîne. Avant d'exécuter ce code, remplacez l'exemple d'ARN de clé par un ARN valide.

```
const char * key_arn = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
```

```
struct aws_cryptosdk_keyring *kms_keyring =
        Aws::Cryptosdk::KmsKeyring::Builder().Build(key_arn);
```

Étape 3 : Créez une session.

Créez une session à l'aide de l'allocateur et du porte-clés. Pour configurer la session pour le déchiffrement, configurez la session avec le mode `AWS_CRYPTOSDK_DECRYPT`.

Après avoir créé une session avec un porte-clés, vous pouvez libérer votre référence au porte-clés à l'aide de la méthode fournie par le kit SDK. La session conserve une référence à l'objet de porte-clés pendant sa durée de vie, et la session et le porte-clés sont tous les deux libérés lorsque vous détruisez la session. Cette technique de comptage de référence permet d'éviter les fuites de mémoire et d'empêcher la libération des objets pendant leur utilisation.

```
struct aws_cryptosdk_session *session =
 aws_cryptosdk_session_new_from_keyring_2(alloc, AWS_CRYPTOSDK_DECRYPT,
 kms_keyring);

/* When you add the keyring to the session, release the keyring object */
aws_cryptosdk_keyring_release(kms_keyring);
```

Étape 4 : Déchiffrement de la chaîne.

Pour déchiffrer la chaîne, utilisez la méthode `aws_cryptosdk_session_process_full` avec la session qui est configurée pour le déchiffrement. Cette méthode, introduite dansAWS Encryption SDKversions 1.9.h/24, j/7et 2.2.h/24, j/7, est conçu pour le chiffrement et le déchiffrement sans streaming. Pour gérer les données en streaming, appelez le`aws_cryptosdk_session_process`en boucle.

Lors du déchiffrement, les champs de texte chiffré sont des champs d'entrée et les champs de texte brut sont des champs de sortie. Le champ `ciphertext_input` contient le message chiffré renvoyé par la méthode de chiffrement. Une fois le traitement terminé, le champ `plaintext_output` contient la chaîne en texte brut (déchiffrée).

```
size_t plaintext_len_output;

if (AWS_OP_SUCCESS != aws_cryptosdk_session_process_full(session,
                                    plaintext_output,
                                    plaintext_buf_sz_output,
                                    &plaintext_len_output,
                                    ciphertext_input,
```

```
                                    ciphertext_len_input)) {
    aws_cryptosdk_session_destroy(session);
    return 13;
}
```

Étape 5 : Vérifiez le contexte de chiffrement.

Assurez-vous que le contexte de chiffrement réel, celui qui a été utilisé pour déchiffrer le message, contient le contexte de chiffrement que vous avez fourni lors du chiffrement du message. Le contexte de chiffrement réel peut inclure des paires supplémentaires. En effet, le gestionnaire de matériaux cryptographiques (CMM) peut ajouter des paires au contexte de chiffrement fourni avant de chiffrer le message.

Dans le kit Kit SDK de chiffrement AWS pour C, vous n'êtes pas tenu de fournir un contexte de chiffrement lors du déchiffrement dans la mesure où le contexte de chiffrement est inclus dans le message chiffré renvoyé par le kit SDK. Toutefois, avant qu'elle ne renvoie le message en texte brut, votre fonction de déchiffrement doit vérifier que toutes les paires du contexte de chiffrement fourni apparaissent dans le contexte de chiffrement qui a été utilisé pour déchiffrer le message.

Tout d'abord, obtenez un pointeur en lecture seule à la table de hachage de la session. Cette table de hachage contient le contexte de chiffrement qui a été utilisé pour déchiffrer le message.

```
const struct aws_hash_table *session_enc_ctx =
 aws_cryptosdk_session_get_enc_ctx_ptr(session);
```

Ensuite, parcourez le contexte de chiffrement dans la table de hachage `my_enc_ctx` que vous avez copiée lors du chiffrement. Vérifiez que chaque paire de la table de hachage `my_enc_ctx` qui a été utilisée pour chiffrer s'affiche dans la table de hachage `session_enc_ctx` qui a été utilisée pour déchiffrer. Si une clé est manquante ou si cette clé a une valeur différente, arrêtez le traitement et écrivez un message d'erreur.

```
for (struct aws_hash_iter iter = aws_hash_iter_begin(my_enc_ctx); !
aws_hash_iter_done(&iter);
      aws_hash_iter_next(&iter)) {
     struct aws_hash_element *session_enc_ctx_kv_pair;
     aws_hash_table_find(session_enc_ctx, iter.element.key,
 &session_enc_ctx_kv_pair)

    if (!session_enc_ctx_kv_pair ||
        !aws_string_eq(
```

```
            (struct aws_string *)iter.element.value, (struct aws_string
 *)session_enc_ctx_kv_pair->value)) {
        fprintf(stderr, "Wrong encryption context!\n");
        abort();
    }
}
```

Étape 6 : Nettoyez la session.

Après avoir vérifié le contexte de chiffrement, vous pouvez détruire la session ou la réutiliser. Si vous devez reconfigurer la session, utilisez la`aws_cryptosdk_session_reset`méthode.

```
aws_cryptosdk_session_destroy(session);
```

AWS Encryption SDKpour .NET

AWS Encryption SDKfor .NET est une bibliothèque de chiffrement côté client 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 du langage de programmation AWS Encryption SDK sont totalement interopérables. Toutefois, si vous chiffrez des données à l'aide du contexte de chiffrement requis CMM dans la version 4. x du AWS Encryption SDK pour .NET, vous ne pouvez le déchiffrer qu'avec la version 4. x du AWS Encryption SDK pour .NET ou version 3. x duKit SDK de chiffrement AWS pour Java.

> **Note**
>
> La version 4.0.0 de AWS Encryption SDK for .NET s'écarte de la spécification du AWS Encryption SDK message. Par conséquent, les messages chiffrés par la version 4.0.0 ne peuvent être déchiffrés que par la version 4.0.0 ou ultérieure pour .NET. AWS Encryption SDK Ils ne peuvent être déchiffrés par aucune autre implémentation de langage de programmation.
> La version 4.0.1 de AWS Encryption SDK for .NET écrit des messages conformément à la spécification des AWS Encryption SDK messages et est interopérable avec d'autres implémentations de langages de programmation. Par défaut, la version 4.0.1 peut lire les messages chiffrés par la version 4.0.0. Toutefois, si vous ne souhaitez pas déchiffrer les messages chiffrés par la version 4.0.0, vous pouvez spécifier la

`NetV4_0_0_RetryPolicy`propriété pour empêcher le client de lire ces messages. Pour plus d'informations, consultez les notes de mise à jour de la version 4.0.1 dans le aws-encryption-sdk-dafny référentiel sur. GitHub

Le AWS Encryption SDK pour .NET se distingue de certaines des autres implémentations du langage de programmation de la AWS Encryption SDK manière suivante :

- Aucune prise en charge de la mise en cache des clés de données

> **Note**
>
> La version 4. x of the AWS Encryption SDK for .NET prend en charge le keyring AWS KMS hiérarchique, une solution alternative de mise en cache des matériaux cryptographiques.

- Pas de support pour le streaming de données
- Aucune journalisation ou trace de pile AWS Encryption SDK depuis .NET
- Nécessite le AWS SDK for .NET

Le AWS Encryption SDK pour .NET inclut toutes les fonctionnalités de sécurité introduites dans les versions 2.0. x et versions ultérieures d'autres implémentations linguistiques duAWS Encryption SDK. Toutefois, si vous utilisez le AWS Encryption SDK pour .NET pour déchiffrer des données chiffrées par une version antérieure à la version 2.0. version x, autre implémentation linguistique duAWS Encryption SDK, vous devrez peut-être ajuster votre politique d'engagement. Pour plus de détails, consultez Comment définir votre politique d'engagement.

Le AWS Encryption SDK for .NET est un produit de AWS Encryption SDK in Dafny, un langage de vérification formel dans lequel vous écrivez des spécifications, le code pour les implémenter et les preuves pour les tester. Le résultat est une bibliothèque qui implémente les fonctionnalités du AWS Encryption SDK dans un framework garantissant l'exactitude fonctionnelle.

En savoir plus

- Pour des exemples montrant comment configurer des options dans leAWS Encryption SDK, telles que la spécification d'une suite d'algorithmes alternative, la limitation des clés de données chiffrées et l'utilisation de clés AWS KMS multirégionales, voirConfiguration du AWS Encryption SDK.
- Pour plus de détails sur la programmation avec le AWS Encryption SDK pour .NET, consultez le `aws-encryption-sdk-net`répertoire du aws-encryption-sdk-dafny référentiel sur GitHub.

Rubriques

Installation du AWS Encryption SDK pour .NET

Le AWS Encryption SDK pour .NET est disponible sous forme de `AWS.Cryptography.EncryptionSDK`package dans NuGet. Pour plus de détails sur l'installation et la génération du fichier AWS Encryption SDK pour .NET, consultez le fichier README.md dans le référentiel. `aws-encryption-sdk-net`

Version 3.x

> Version 3. x of the AWS Encryption SDK for .NET prend en charge .NET Framework 4.5.2 à 4.8 uniquement sous Windows. Il prend en charge .NET Core 3.0+, .NET 5.0 et versions ultérieures sur tous les systèmes d'exploitation pris en charge.

Version 4.x

> La version 4. x of the AWS Encryption SDK for .NET prend en charge .NET 6.0 et .NET Framework net48 et versions ultérieures.

AWS Encryption SDKPour .NET, les touches « AWS SDK for .NET même si vous n'utilisez pas AWS Key Management Service (AWS KMS) » sont requises. Il est installé avec le NuGet package. Toutefois, sauf si vous utilisez AWS KMS des clés, AWS Encryption SDK car .NET ne nécessite pas d'Compte AWSAWSinformations d'identification ou d'interaction avec un AWS service. Pour obtenir de l'aide pour configurer un AWS compte si vous en avez besoin, consultezUtilisation de l'AWS Encryption SDK avec AWS KMS.

Débogage du AWS Encryption SDK pour .NET

Le AWS Encryption SDK for .NET ne génère aucun journal. Les exceptions dans le AWS Encryption SDK fichier for .NET génèrent un message d'exception, mais aucune trace de pile.

Pour vous aider à déboguer, veillez à activer la connexion auAWS SDK for .NET. Les journaux et les messages d'erreur du AWS SDK for .NET peuvent vous aider à distinguer les erreurs survenant dans le fichier .NET AWS SDK for .NET de celles qui se produisent dans le AWS Encryption SDK fichier .NET. Pour obtenir de l'aide AWS SDK for .NET concernant la journalisation, consultez AWSLogginle guide du AWS SDK for .NET développeur. (Pour consulter le sujet, développez la section Ouvrir pour afficher le contenu du .NET Framework.)

AWS KMSporte-clés dans le AWS Encryption SDK for .NET

Les AWS KMS trousseaux de clés de base AWS Encryption SDK pour .NET n'utilisent qu'une seule clé KMS. Ils ont également besoin d'un AWS KMS client, ce qui vous permet de configurer le client pour Région AWS la clé KMS.

Pour créer un AWS KMS porte-clés avec une ou plusieurs clés enveloppantes, utilisez un porte-clés à plusieurs clés. AWS Encryption SDKPour .NET, il existe un porte-clés spécial qui accepte une ou plusieurs AWS KMS clés, et un porte-clés standard qui accepte un ou plusieurs porte-clés de n'importe quel type compatible. Certains programmeurs préfèrent utiliser une méthode à plusieurs jeux de clés pour créer tous leurs trousseaux de clés, et AWS Encryption SDK for .NET prend en charge cette stratégie.

Le AWS Encryption SDK for .NET fournit des porte-clés simples et des porte-clés multiples de base pour tous les cas d'utilisation courants, y compris les clés multirégionales. AWS KMS

Par exemple, pour créer un AWS KMS porte-clés avec une seule AWS KMS clé, vous pouvez utiliser la `CreateAwsKmsKeyring()` méthode.

Version 3.x

L'exemple suivant utilise la version 3. x de AWS Encryption SDK pour .NET afin de créer un AWS KMS client par défaut pour la région contenant la clé spécifiée.

```
// Instantiate the AWS Encryption SDK and material providers
var encryptionSdk = AwsEncryptionSdkFactory.CreateDefaultAwsEncryptionSdk();
var materialProviders =

 AwsCryptographicMaterialProvidersFactory.CreateDefaultAwsCryptographicMaterialProviders();

string keyArn = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
```

```
// Instantiate the keyring input object
var kmsKeyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = keyArn
};

// Create the keyring
var keyring = materialProviders.CreateAwsKmsKeyring(kmsKeyringInput);
```

Version 4.x

L'exemple suivant utilise la version 4. x du AWS Encryption SDK pour .NET afin de créer un AWS KMS client pour la région contenant la clé spécifiée.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk =  new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

string keyArn = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

// Instantiate the keyring input object
var createKeyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = kmsArn
};

// Create the keyring
var kmsKeyring = mpl.CreateAwsKmsKeyring(createKeyringInput);
```

Pour créer un trousseau de clés avec une ou plusieurs AWS KMS clés, utilisez `CreateAwsKmsMultiKeyring()` cette méthode. Cet exemple utilise deux AWS KMS touches. Pour spécifier une clé KMS, utilisez uniquement le `Generator` paramètre. Le `KmsKeyIds` paramètre qui spécifie des clés KMS supplémentaires est facultatif.

La saisie de ce trousseau de clés ne nécessite aucun AWS KMS client. Il AWS Encryption SDK utilise plutôt le AWS KMS client par défaut pour chaque région représentée par une clé KMS dans le trousseau de clés. Par exemple, si la clé KMS identifiée par la valeur du `Generator` paramètre se trouve dans la région USA Ouest (Oregon) (`us-west-2`), un AWS KMS client par défaut est AWS

Encryption SDK créé pour `us-west-2` cette région. Si vous devez personnaliser le AWS KMS client, utilisez `CreateAwsKmsKeyring()` cette méthode.

L'exemple suivant utilise la version 4. x du AWS Encryption SDK pour .NET et la `CreateAwsKmsKeyring()` méthode de personnalisation du AWS KMS client.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk =  new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

string generatorKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
List<string> additionalKeys = new List<string> { "arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321" };

// Instantiate the keyring input object
var createEncryptKeyringInput = new CreateAwsKmsMultiKeyringInput
{
    Generator = generatorKey,
    KmsKeyIds = additionalKeys
};

var kmsEncryptKeyring =
 materialProviders.CreateAwsKmsMultiKeyring(createEncryptKeyringInput);
```

La version 4. x of the AWS Encryption SDK for .NET prend en charge les AWS KMS ensembles de clés utilisant le chiffrement symétrique (`SYMMETRIC_DEFAULT`) ou des clés RSA KMS asymétriques. AWS KMSles porte-clés créés avec des clés RSA KMS asymétriques ne peuvent contenir qu'une seule paire de clés.

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 porte-clés. Pour déchiffrer avec un trousseau de AWS KMS clés RSA asymétrique, vous devez disposer de l'autorisation KMS:Decrypt.

Pour créer un trousseau de AWS KMS clés RSA asymétrique, vous devez fournir l'ARN de la clé publique et de la clé privée à partir de votre clé RSA KMS asymétrique. La clé publique doit être

codée au format PEM. L'exemple suivant crée un AWS KMS trousseau de clés avec une paire de clés RSA asymétrique.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk =  new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

var publicKey = new MemoryStream(Encoding.UTF8.GetBytes(AWS KMS RSA public key));

// Instantiate the keyring input object
var createKeyringInput = new CreateAwsKmsRsaKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = AWS KMS RSA private key ARN,
    PublicKey = publicKey,
    EncryptionAlgorithm = EncryptionAlgorithmSpec.RSAES_OAEP_SHA_256
};

// Create the keyring
var kmsRsaKeyring = mpl.CreateAwsKmsRsaKeyring(createKeyringInput);
```

Contextes de chiffrement requis dans la version 4.x

Avec la version 4. x du AWS Encryption SDK pour .NET, vous pouvez utiliser le contexte de chiffrement requis CMM pour exiger des contextes de chiffrement dans vos opérations cryptographiques. Un contexte de chiffrement est un ensemble de paires clé-valeur non secrètes. Le contexte de chiffrement est lié cryptographiquement aux données chiffrées, de sorte que le même contexte de chiffrement est requis pour déchiffrer le champ. Lorsque vous utilisez le contexte de chiffrement requis CMM, vous pouvez spécifier une ou plusieurs clés de contexte de chiffrement requises (clés obligatoires) qui doivent être incluses dans tous les appels de chiffrement et de déchiffrement.

> **Note**
>
> Le contexte de chiffrement requis CMM n'est interopérable qu'avec la version 3. x duKit SDK de chiffrement AWS pour Java. Il n'est interopérable avec aucune autre implémentation de langage de programmation. Si vous chiffrez des données à l'aide du contexte de chiffrement requis CMM, vous ne pouvez les déchiffrer qu'avec la version 3. x de la version 4 de l'Kit SDK de chiffrement AWS pour Javaor. x du AWS Encryption SDK pour .NET.

Lors du chiffrement, il AWS Encryption SDK vérifie que toutes les clés de contexte de chiffrement requises sont incluses dans le contexte de chiffrement que vous avez spécifié. Les AWS Encryption SDK signes indiquent les contextes de chiffrement que vous avez spécifiés. Seules les paires clé-valeur qui ne sont pas des clés obligatoires sont sérialisées et stockées en texte clair dans l'en-tête du message chiffré renvoyé par l'opération de chiffrement.

Lors du déchiffrement, vous devez fournir un contexte de chiffrement contenant toutes les paires clé-valeur représentant les clés requises. AWS Encryption SDKUtilise ce contexte de chiffrement et les paires clé-valeur stockées dans l'en-tête du message chiffré pour reconstruire le contexte de chiffrement d'origine que vous avez spécifié lors de l'opération de chiffrement. S'il est AWS Encryption SDK impossible de reconstruire le contexte de chiffrement d'origine, l'opération de déchiffrement échoue. Si vous fournissez une paire clé-valeur contenant la clé requise avec une valeur incorrecte, le message chiffré ne peut pas être déchiffré. Vous devez fournir la même paire clé-valeur que celle spécifiée lors du chiffrement.

> **⚠ Important**
>
> Réfléchissez bien aux valeurs que vous choisissez pour les clés requises dans votre contexte de chiffrement. Vous devez être en mesure de fournir à nouveau les mêmes clés et les valeurs correspondantes lors du déchiffrement. Si vous ne parvenez pas à reproduire les clés requises, le message chiffré ne peut pas être déchiffré.

L'exemple suivant initialise un AWS KMS trousseau de clés avec le contexte de chiffrement requis CMM.

```
var encryptionContext = new Dictionary<string, string>()
{
    {"encryption", "context"},
    {"is not", "secret"},
    {"but adds", "useful metadata"},
    {"that can help you", "be confident that"},
    {"the data you are handling", "is what you think it is"}
};

// Instantiate the AWS Encryption SDK and material providers
var esdk =  new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());

// Instantiate the keyring input object
```

```
var createKeyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = kmsKey
};

// Create the keyring
var kmsKeyring = mpl.CreateAwsKmsKeyring(createKeyringInput);

var createCMMInput = new CreateRequiredEncryptionContextCMMInput
{
    UnderlyingCMM = mpl.CreateDefaultCryptographicMaterialsManager(new
 CreateDefaultCryptographicMaterialsManagerInput{Keyring = kmsKeyring}),
    // If you pass in a keyring but no underlying cmm, it will result in a failure
 because only cmm is supported.
    RequiredEncryptionContextKeys = new List<string>(encryptionContext.Keys)
};

// Create the required encryption context CMM
var requiredEcCMM = mpl.CreateRequiredEncryptionContextCMM(createCMMInput);
```

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

AWS Encryption SDKpour des exemples .NET

Les exemples suivants montrent les modèles de codage de base que vous utilisez lors de la programmation avec le AWS Encryption SDK pour .NET. Plus précisément, vous instanciez la bibliothèque AWS Encryption SDK et les fournisseurs de matériaux. Ensuite, avant d'appeler chaque méthode, vous instanciez un objet qui définit l'entrée de la méthode. Cela ressemble beaucoup au modèle de codage que vous utilisez dans leAWS SDK for .NET.

Pour des exemples montrant comment configurer des options dans leAWS Encryption SDK, telles que la spécification d'une suite d'algorithmes alternative, la limitation des clés de données chiffrées et l'utilisation de clés AWS KMS multirégionales, voirConfiguration du AWS Encryption SDK.

Pour plus d'exemples de programmation avec le AWS Encryption SDK pour .NET, consultez les exemples dans le `aws-encryption-sdk-net` répertoire du `aws-encryption-sdk-dafny` référentiel sur GitHub.

Chiffrer des données dans le AWS Encryption SDK pour .NET

Cet exemple montre le schéma de base pour le chiffrement des données. Il chiffre un petit fichier avec des clés de données protégées par une clé AWS KMS d'encapsulation.

Étape 1 : Instancier la bibliothèque AWS Encryption SDK et la bibliothèque des fournisseurs de matériaux.

Commencez par instancier la bibliothèque AWS Encryption SDK et les fournisseurs de matériaux. Vous allez utiliser les méthodes décrites dans le AWS Encryption SDK pour chiffrer et déchiffrer les données. Vous allez utiliser les méthodes de la bibliothèque des fournisseurs de matériel pour créer les porte-clés qui spécifient les clés qui protègent vos données.

La façon dont vous instanciez la bibliothèque AWS Encryption SDK et la bibliothèque des fournisseurs de matériaux diffère selon les versions 3. x et 4. x du AWS Encryption SDK pour .NET. Toutes les étapes suivantes sont identiques pour les deux versions 3. x et 4. x du AWS Encryption SDK pour .NET.

Version 3.x

```
// Instantiate the AWS Encryption SDK and material providers
var encryptionSdk = AwsEncryptionSdkFactory.CreateDefaultAwsEncryptionSdk();
var materialProviders =

 AwsCryptographicMaterialProvidersFactory.CreateDefaultAwsCryptographicMaterialProviders(
```

Version 4.x

```
// Instantiate the AWS Encryption SDK and material providers
var esdk =  new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());
```

Étape 2 : Créez un objet de saisie pour le trousseau de clés.

Chaque méthode qui crée un trousseau de clés possède une classe d'objet d'entrée correspondante. Par exemple, pour créer l'objet d'entrée pour la `CreateAwsKmsKeyring()` méthode, créez une instance de la `CreateAwsKmsKeyringInput` classe.

Même si l'entrée de ce trousseau de clés ne spécifie pas de clé de générateur, la clé KMS unique spécifiée par le `KmsKeyId` paramètre est la clé de générateur. Il génère et chiffre la clé de données qui chiffre les données.

Cet objet d'entrée nécessite un AWS KMS client pour Région AWS la clé KMS. Pour créer un AWS KMS client, instanciez la `AmazonKeyManagementServiceClient` classe dans le. AWS SDK for .NET L'appel du `AmazonKeyManagementServiceClient()` constructeur sans paramètres crée un client avec les valeurs par défaut.

Dans un AWS KMS trousseau de clés utilisé pour le chiffrement avec .NET, vous pouvez identifier AWS Encryption SDK les clés KMS à l'aide de l'ID de clé, de l'ARN de la clé, du nom de l'alias ou de l'ARN de l'alias. Dans un AWS KMS trousseau de clés utilisé pour le déchiffrement, vous devez utiliser un ARN de clé pour identifier chaque clé KMS. Si vous prévoyez de réutiliser votre jeu de clés de chiffrement pour le déchiffrer, utilisez un identifiant ARN pour toutes les clés KMS.

```
string keyArn = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

// Instantiate the keyring input object
var kmsKeyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = keyArn
};
```

Étape 3 : Créez le porte-clés.

Pour créer le trousseau de clés, appelez la méthode du trousseau avec l'objet de saisie du trousseau de clés. Cet exemple utilise la `CreateAwsKmsKeyring()` méthode, qui n'utilise qu'une seule clé KMS.

```
var keyring = materialProviders.CreateAwsKmsKeyring(kmsKeyringInput);
```

Étape 4 : définir un contexte de chiffrement.

Un contexte de chiffrement est un élément facultatif mais fortement recommandé des opérations cryptographiques dans leAWS Encryption SDK. Vous pouvez définir une ou plusieurs paires clé-valeur non secrètes.

> **Note**
>
> Avec la version 4. x de AWS Encryption SDK pour .NET, vous pouvez exiger un contexte de chiffrement dans toutes les demandes de chiffrement avec le contexte de chiffrement requis CMM.

```
// Define the encryption context
var encryptionContext = new Dictionary<string, string>()
{
    {"purpose", "test"}
};
```

Étape 5 : Créez l'objet d'entrée pour le chiffrement.

Avant d'appeler la `Encrypt()` méthode, créez une instance de la `EncryptInput` classe.

```
string plaintext = File.ReadAllText("C:\\Documents\\CryptoTest\\TestFile.txt");

// Define the encrypt input
var encryptInput = new EncryptInput
{
    Plaintext = plaintext,
    Keyring = keyring,
    EncryptionContext = encryptionContext
};
```

Étape 6 : Chiffrez le texte en clair.

Utilisez la `Encrypt()` méthode du AWS Encryption SDK pour chiffrer le texte en clair à l'aide du trousseau de clés que vous avez défini.

La `EncryptOutput` `Encrypt()` méthode renvoie des méthodes pour obtenir le message crypté (`Ciphertext`), le contexte de chiffrement et la suite d'algorithmes.

```
var encryptOutput = encryptionSdk.Encrypt(encryptInput);
```

Étape 7 : Obtenez le message crypté.

La `Decrypt()` méthode décrite dans AWS Encryption SDK for .NET prend le `Ciphertext` membre de l'`EncryptOutput`instance.

Le `Ciphertext` membre de l'`EncryptOutput`objet est le message chiffré, un objet portable qui inclut les données chiffrées, les clés de données chiffrées et les métadonnées, y compris le contexte de chiffrement. Vous pouvez stocker le message crypté en toute sécurité pendant une période prolongée ou le soumettre à la `Decrypt()` méthode pour récupérer le texte en clair.

```
var encryptedMessage = encryptOutput.Ciphertext;
```

Déchiffrement en mode strict dans le fichier pour .NET AWS Encryption SDK

Les meilleures pratiques recommandent de spécifier les clés que vous utilisez pour déchiffrer les données, une option connue sous le nom de mode strict. AWS Encryption SDKUtilise uniquement les clés KMS que vous spécifiez dans votre trousseau de clés pour déchiffrer le texte chiffré. Les clés de votre jeu de clés de déchiffrement doivent inclure au moins l'une des clés qui ont chiffré les données.

Cet exemple montre le schéma de base du déchiffrement en mode strict avec le AWS Encryption SDK pour .NET.

Étape 1 : Instancier la bibliothèque AWS Encryption SDK et les fournisseurs de matériaux.

```
// Instantiate the AWS Encryption SDK and material providers
var esdk =  new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());
```

Étape 2 : Créez l'objet de saisie pour votre trousseau de clés.

Pour définir les paramètres de la méthode keyring, créez un objet d'entrée. Chaque méthode de porte-clés de AWS Encryption SDK for .NET possède un objet d'entrée correspondant. Comme cet exemple utilise la `CreateAwsKmsKeyring()` méthode pour créer le trousseau de clés, il instancie la `CreateAwsKmsKeyringInput` classe pour l'entrée.

Dans un trousseau de clés de déchiffrement, vous devez utiliser un ARN pour identifier les clés KMS.

```
string keyArn = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

// Instantiate the keyring input object
var kmsKeyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = keyArn
};
```

Étape 3 : Créez le porte-clés.

Pour créer le trousseau de clés de déchiffrement, cet exemple utilise la `CreateAwsKmsKeyring()` méthode et l'objet d'entrée du trousseau de clés.

```
var keyring = materialProviders.CreateAwsKmsKeyring(kmsKeyringInput);
```

Étape 4 : Créez l'objet d'entrée à déchiffrer.

Pour créer l'objet d'entrée pour la `Decrypt()` méthode, instanciez la `DecryptInput` classe.

Le `Ciphertext` paramètre du `DecryptInput()` constructeur prend le `Ciphertext` membre de l'`EncryptOutput`objet renvoyé par la `Encrypt()` méthode. La `Ciphertext` propriété représente le message chiffré, qui inclut les données chiffrées, les clés de données chiffrées et les métadonnées AWS Encryption SDK nécessaires pour déchiffrer le message.

Avec la version 4. x de AWS Encryption SDK pour .NET, vous pouvez utiliser le `EncryptionContext` paramètre facultatif pour spécifier votre contexte de chiffrement dans la `Decrypt()` méthode.

Utilisez le `EncryptionContext` paramètre pour vérifier que le contexte de chiffrement utilisé pour chiffrer est inclus dans le contexte de chiffrement utilisé pour déchiffrer le texte chiffré. AWS Encryption SDKajoute des paires au contexte de chiffrement, y compris la signature numérique si vous utilisez une suite d'algorithmes avec signature, telle que la suite d'algorithmes par défaut.

```
var encryptedMessage = encryptOutput.Ciphertext;

var decryptInput = new DecryptInput
{
    Ciphertext = encryptedMessage,
    Keyring = keyring,
    EncryptionContext = encryptionContext // OPTIONAL
};
```

Étape 5 : Déchiffrez le texte chiffré.

```
var decryptOutput = encryptionSdk.Decrypt(decryptInput);
```

Étape 6 : Vérifiez le contexte de chiffrement — Version 3. x

La `Decrypt()` méthode de la version 3. x de AWS Encryption SDK for .NET ne prend pas de contexte de chiffrement. Il obtient les valeurs du contexte de chiffrement à partir des métadonnées du message chiffré. Toutefois, avant de renvoyer ou d'utiliser le texte en clair, il est recommandé de vérifier que le contexte de chiffrement utilisé pour déchiffrer le texte chiffré inclut le contexte de chiffrement que vous avez fourni lors du chiffrement.

Vérifiez que le contexte de chiffrement utilisé pour chiffrer est inclus dans le contexte de chiffrement utilisé pour déchiffrer le texte chiffré. AWS Encryption SDKajoute des paires au contexte de chiffrement, y compris la signature numérique si vous utilisez une suite d'algorithmes avec signature, telle que la suite d'algorithmes par défaut.

```
// Verify the encryption context
string contextKey = "purpose";
string contextValue = "test";

if (!decryptOutput.EncryptionContext.TryGetValue(contextKey, out var
 decryptContextValue)
    || !decryptContextValue.Equals(contextValue))
{
    throw new Exception("Encryption context does not match expected values");
}
```

Déchiffrement à l'aide d'un trousseau de découverte dans le for .NET AWS Encryption SDK

Plutôt que de spécifier les clés KMS pour le déchiffrement, vous pouvez fournir un jeu de clés de AWS KMS découverte, qui est un jeu de clés qui ne spécifie aucune clé KMS. Un trousseau de découverte permet de AWS Encryption SDK déchiffrer les données en utilisant la clé KMS qui les a chiffrées, à condition que l'appelant ait l'autorisation de déchiffrer la clé. Pour les meilleures pratiques, ajoutez un filtre de découverte qui limite les clés KMS pouvant être utilisées à celles Comptes AWS d'une partition spécifique.

Le jeu de clés AWS Encryption SDK pour .NET fournit un jeu de clés de découverte de base qui nécessite un AWS KMS client et un jeu de clés de découverte multiple qui nécessite que vous en spécifiiez un ou plusieurs. Régions AWS Le client et les régions limitent tous deux les clés KMS qui peuvent être utilisées pour déchiffrer le message chiffré. Les objets d'entrée pour les deux trousses de clés utilisent le filtre de découverte recommandé.

L'exemple suivant montre le modèle de déchiffrement des données à l'aide d'un trousseau de découverte et d'un AWS KMS filtre de découverte.

Étape 1 : Instancier la bibliothèque AWS Encryption SDK et la bibliothèque des fournisseurs de matériaux.

```
// Instantiate the AWS Encryption SDK and material providers
```

```
var esdk =  new ESDK(new AwsEncryptionSdkConfig());
var mpl = new MaterialProviders(new MaterialProvidersConfig());
```

Étape 2 : Créez l'objet de saisie pour le trousseau de clés.

Pour définir les paramètres de la méthode keyring, créez un objet d'entrée. Chaque méthode de porte-clés de AWS Encryption SDK for .NET possède un objet d'entrée correspondant. Comme cet exemple utilise la `CreateAwsKmsDiscoveryKeyring()` méthode pour créer le trousseau de clés, il instancie la `CreateAwsKmsDiscoveryKeyringInput` classe pour l'entrée.

```
List<string> accounts = new List<string> { "111122223333" };

var discoveryKeyringInput = new CreateAwsKmsDiscoveryKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    DiscoveryFilter = new DiscoveryFilter()
    {
        AccountIds = accounts,
        Partition = "aws"
    }
};
```

Étape 3 : Créez le porte-clés.

Pour créer le trousseau de clés de déchiffrement, cet exemple utilise la `CreateAwsKmsDiscoveryKeyring()` méthode et l'objet d'entrée du trousseau de clés.

```
var discoveryKeyring =
 materialProviders.CreateAwsKmsDiscoveryKeyring(discoveryKeyringInput);
```

Étape 4 : Créez l'objet d'entrée à déchiffrer.

Pour créer l'objet d'entrée pour la `Decrypt()` méthode, instanciez la `DecryptInput` classe. La valeur du `Ciphertext` paramètre est le `Ciphertext` membre de l'`EncryptOutput`objet renvoyé par la `Encrypt()` méthode.

Avec la version 4. x de AWS Encryption SDK pour .NET, vous pouvez utiliser le `EncryptionContext` paramètre facultatif pour spécifier votre contexte de chiffrement dans la `Decrypt()` méthode.

Utilisez le `EncryptionContext` paramètre pour vérifier que le contexte de chiffrement utilisé lors du chiffrement est inclus dans le contexte de chiffrement utilisé pour déchiffrer le texte

chiffré. AWS Encryption SDKajoute des paires au contexte de chiffrement, y compris la signature numérique si vous utilisez une suite d'algorithmes avec signature, telle que la suite d'algorithmes par défaut.

```
var ciphertext = encryptOutput.Ciphertext;

var decryptInput = new DecryptInput
{
    Ciphertext = ciphertext,
    Keyring = discoveryKeyring,
    EncryptionContext = encryptionContext // OPTIONAL

};
var decryptOutput = encryptionSdk.Decrypt(decryptInput);
```

Étape 5 : Vérifiez le contexte de chiffrement — Version 3. x

La `Decrypt()` méthode de la version 3. x of the AWS Encryption SDK for .NET ne prend pas en charge de contexte de chiffrement activé`Decrypt()`. Il obtient les valeurs du contexte de chiffrement à partir des métadonnées du message chiffré. Toutefois, avant de renvoyer ou d'utiliser le texte en clair, il est recommandé de vérifier que le contexte de chiffrement utilisé pour déchiffrer le texte chiffré inclut le contexte de chiffrement que vous avez fourni lors du chiffrement.

Vérifiez que le contexte de chiffrement utilisé lors du chiffrement est inclus dans le contexte de chiffrement utilisé pour déchiffrer le texte chiffré. AWS Encryption SDKajoute des paires au contexte de chiffrement, y compris la signature numérique si vous utilisez une suite d'algorithmes avec signature, telle que la suite d'algorithmes par défaut.

```
// Verify the encryption context
string contextKey = "purpose";
string contextValue = "test";

if (!decryptOutput.EncryptionContext.TryGetValue(contextKey, out var
 decryptContextValue)
    || !decryptContextValue.Equals(contextValue))
{
    throw new Exception("Encryption context does not match expected values");
}
```

Kit SDK de chiffrement AWS pour Java

Cette rubrique explique comment installer et utiliser le kit Kit SDK de chiffrement AWS pour Java. Pour plus de détails sur la programmation avec leKit SDK de chiffrement AWS pour Java, consultez le aws-encryption-sdk-javaréférentiel sur GitHub. Pour la documentation de l'API, consultez le Javadoc pour le Kit SDK de chiffrement AWS pour Java.

Rubriques

Prerequisites (Prérequis)

Avant d'installer le kit Kit SDK de chiffrement AWS pour Java, assurez-vous de remplir les conditions prérequises 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.

Bouncy Castle

Le kit Kit SDK de chiffrement AWS pour Java exige Bouncy Castle.

- Les versions Kit SDK de chiffrement AWS pour Java 1.6.1 et ultérieures utilisent Bouncy Castle pour sérialiser et désérialiser des objets de chiffrement. Vous pouvez utiliser Bouncy Castle ou Bouncy Castle FIPS pour répondre à cette exigence. Pour obtenir de l'aide sur l'installation et la configuration de Bouncy Castle FIPS, veuillez consulter la documentation BC FIPS, en particulier les guides de l'utilisateur et les PDF de stratégie de sécurité.
- Les versions antérieures de Kit SDK de chiffrement AWS pour Java utilisent l'API de chiffrement de Bouncy Castle pour Java. Cette exigence n'est satisfaite que par les Bouncy Castle non FIPS.

Si vous n'avez pas Bouncy Castle, accédez aux dernières versions de Bouncy Castle pour télécharger le fichier de fournisseur correspondant à votre JDK. Vous pouvez également utiliser Apache Maven pour obtenir l'artefact pour le fournisseur standard de Bouncy Castle (bcprov-ext-jdk15on) ou l'artefact pour Bouncy Castle FIPS (bc-fips).

AWS SDK for Java

Version 3. x l'Kit SDK de chiffrement AWS pour JavaexigeAWS SDK for Java 2.x, même si vous n'utilisez pas de AWS KMS porte-clés.

Version 2. x ou une version antérieure Kit SDK de chiffrement AWS pour Java ne nécessite pas leAWS SDK for Java. Cependant, AWS SDK for Java il est nécessaire d'utiliser AWS Key Management Service(AWS KMS) comme fournisseur de clé principale. À partir de la Kit SDK de chiffrement AWS pour Java version 2.4.0, il Kit SDK de chiffrement AWS pour Java prend en charge les versions 1.x et 2.x du. AWS SDK for Java AWS Encryption SDKle code pour les versions AWS SDK for Java 1.x et 2.x est interopérable. Par exemple, vous pouvez chiffrer des données avec un AWS Encryption SDK code compatible avec la version AWS SDK for Java 1.x et les déchiffrer à l'aide d'un code compatible AWS SDK for Java 2.x (ou vice versa). Les versions antérieures à Kit SDK de chiffrement AWS pour Java 2.4.0 ne prennent en charge que la version AWS SDK for Java 1.x. Pour plus d'informations sur la mise à jour de votre version duAWS Encryption SDK, consultezMigrer votreAWS Encryption SDK.

Lorsque vous mettez à jour votre Kit SDK de chiffrement AWS pour Java code de la AWS SDK for Java version 1.x àAWS SDK for Java 2.x, remplacez les références à l'`AWSKMSinterface` de la version AWS SDK for Java 1.x par des références à l'`KmsClientinterface` de. AWS SDK for Java 2.x Kit SDK de chiffrement AWS pour JavaNe prend pas en charge l'`KmsAsyncClientinterface`. Mettez également à jour votre code pour utiliser les objets AWS KMS associés dans l'espace de `kmssdkv2` noms, plutôt que dans l'espace de `kms` noms.

Pour installer le AWS SDK for Java, utilisez Apache Maven.

- Pour importer tout le AWS SDK for Java en tant que dépendance, déclarez-le dans votre fichier `pom.xml`.
- Pour créer une dépendance uniquement pour le AWS KMS module dans la version AWS SDK for Java 1.x, suivez les instructions pour spécifier des modules particuliers et définissez la valeur sur`artifactId`. `aws-java-sdk-kms`
- Pour créer une dépendance uniquement pour le AWS KMS module dans la version AWS SDK for Java 2.x, suivez les instructions pour spécifier des modules particuliers. Réglez le `groupId` to `software.amazon.awssdk` et le `artifactId` to`kms`.

Pour plus de modifications, consultez la section Quelles sont les différences entre la version AWS SDK for Java 1.x et la version 2.x dans le manuel du AWS SDK for Java 2.x développeur.

Les exemples Java présentés dans le Guide du AWS Encryption SDK développeur utilisent leAWS SDK for Java 2.x.

Installation

Installez la dernière version duKit SDK de chiffrement AWS pour Java.

> **Note**
>
> Toutes les versions Kit SDK de chiffrement AWS pour Java antérieures à la version 2.0.0 sont en end-of-supportphase.
> Vous pouvez effectuer la mise à jour en toute sécurité à partir de la version 2.0. x et versions ultérieures vers la dernière version du Kit SDK de chiffrement AWS pour Java sans aucune modification du code ou des données. Cependant, de nouvelles fonctionnalités de sécurité ont été introduites dans la version 2.0. x ne sont pas rétrocompatibles. Pour effectuer une mise à jour à partir de versions antérieures à 1.7. x vers la version 2.0. x et versions ultérieures, vous devez d'abord effectuer la mise à jour vers la dernière version 1. version x duAWS Encryption SDK. Pour plus de détails, consultez Migrer votreAWS Encryption SDK.

Vous pouvez installer le kit Kit SDK de chiffrement AWS pour Java de l'une des manières suivantes :

Manuellement

Pour installerKit SDK de chiffrement AWS pour Java, cloner ou télécharger le aws-encryption-sdk-java GitHubréférentiel.

Utilisation d'Apache Maven

Kit SDK de chiffrement AWS pour Java est disponible via Apache Maven avec la définition de dépendance suivante.

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-encryption-sdk-java</artifactId>
  <version>3.0.0</version>
</dependency>
```

Après avoir installé le SDK, commencez par consulter l'exemple de code Java présenté dans ce guide et le Javadoc activé. GitHub

AWS KMSporte-clés dans le Kit SDK de chiffrement AWS pour Java

Version 3. x of Kit SDK de chiffrement AWS pour Java utilise des trousseaux de clés pour chiffrer les enveloppes. Les AWS KMS porte-clés de base du clavier ne nécessitent Kit SDK de chiffrement AWS pour Java qu'une seule clé KMS. Ils ont également besoin d'un AWS KMS client, ce qui vous permet de configurer le client pour Région AWS la clé KMS.

Pour créer un AWS KMS porte-clés avec une ou plusieurs clés enveloppantes, utilisez un porte-clés à plusieurs clés. Kit SDK de chiffrement AWS pour JavaIl possède un porte-clés spécial qui prend une ou plusieurs AWS KMS clés, et un porte-clés standard qui accepte un ou plusieurs porte-clés de n'importe quel type compatible. Certains programmeurs préfèrent utiliser une méthode à plusieurs jeux de clés pour créer tous leurs trousseaux de clés, et ils soutiennent cette stratégie. Kit SDK de chiffrement AWS pour Java

Kit SDK de chiffrement AWS pour JavaIl fournit des porte-clés simples et des porte-clés multiples de base pour tous les cas d'utilisation courants, y compris les clés multirégionales. AWS KMS

Par exemple, pour créer un AWS KMS trousseau de clés avec une seule AWS KMS clé, vous pouvez utiliser la méthode`CreateAwsKmsKeyring()`].

```
// Instantiate the AWS Encryption SDK and material providers
final AwsCrypto crypto = AwsCrypto.builder().build();
final MaterialProviders materialProviders = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();

// Create the keyring
CreateAwsKmsKeyringInput kmsKeyringInput = CreateAwsKmsKeyringInput.builder()
        .kmsKeyId(keyArn)
        .kmsClient(KmsClient.create())
        .build();
IKeyring kmsKeyring = materialProviders.CreateAwsKmsKeyring(kmsKeyringInput);
```

Pour créer un trousseau de clés avec une ou plusieurs AWS KMS clés, utilisez `CreateAwsKmsMultiKeyring()` cette méthode. Cet exemple utilise deux clés KMS. Pour spécifier une clé KMS, utilisez uniquement le `generator` paramètre. Le `msKeyIds` paramètre qui spécifie des clés KMS supplémentaires est facultatif.

La saisie de ce trousseau de clés ne nécessite aucun AWS KMS client. Il AWS Encryption SDK utilise plutôt le AWS KMS client par défaut pour chaque région représentée par une clé KMS dans le trousseau de clés. Par exemple, si la clé KMS identifiée par la valeur du `Generator` paramètre se trouve dans la région USA Ouest (Oregon) (`us-west-2`), un AWS KMS client par défaut est AWS Encryption SDK créé pour `us-west-2` cette région. Si vous devez personnaliser le AWS KMS client, utilisez `CreateAwsKmsKeyring()` cette méthode.

```
// Instantiate the AWS Encryption SDK and material providers
final AwsCrypto crypto = AwsCrypto.builder().build();
final MaterialProviders materialProviders = MaterialProviders.builder()
            .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
            .build();

String generatorKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
List<String> additionalKey = Collections.singletonList("arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321");
// Create the keyring
final CreateAwsKmsMultiKeyringInput keyringInput =
 CreateAwsKmsMultiKeyringInput.builder()
        .generator(generatorKey)
        .kmsKeyIds(additionalKey)
        .build();
final IKeyring kmsKeyring = matProv.CreateAwsKmsMultiKeyring(keyringInput);
```

Kit SDK de chiffrement AWS pour Javaprend en charge les AWS KMS trousseaux de clés qui utilisent un chiffrement symétrique (`SYMMETRIC_DEFAULT`) ou des clés RSA KMS asymétriques. AWS KMSles porte-clés créés avec des clés RSA KMS asymétriques ne peuvent contenir qu'une seule paire de clés.

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 porte-clés. Pour déchiffrer avec un trousseau de AWS KMS clés RSA asymétrique, vous devez disposer de l'autorisation KMS:Decrypt.

Pour créer un trousseau de AWS KMS clés RSA asymétrique, vous devez fournir l'ARN de la clé publique et de la clé privée à partir de votre clé RSA KMS asymétrique. La clé publique doit être codée au format PEM. L'exemple suivant crée un AWS KMS trousseau de clés avec une paire de clés RSA asymétrique.

```
// Instantiate the AWS Encryption SDK and material providers
final AwsCrypto crypto = AwsCrypto.builder()
        // Specify algorithmSuite without asymmetric signing here
        //
        // ALG_AES_128_GCM_IV12_TAG16_NO_KDF("0x0014"),
        // ALG_AES_192_GCM_IV12_TAG16_NO_KDF("0x0046"),
        // ALG_AES_256_GCM_IV12_TAG16_NO_KDF("0x0078"),
        // ALG_AES_128_GCM_IV12_TAG16_HKDF_SHA256("0x0114"),
        // ALG_AES_192_GCM_IV12_TAG16_HKDF_SHA256("0x0146"),
        // ALG_AES_256_GCM_IV12_TAG16_HKDF_SHA256("0x0178")

 .withEncryptionAlgorithm(CryptoAlgorithm.ALG_AES_256_GCM_IV12_TAG16_HKDF_SHA256)
        .build();

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

// Create a KMS RSA keyring.
//    This keyring takes in:
//     - kmsClient
//     - kmsKeyId: Must be an ARN representing an asymmetric RSA KMS key
//     - publicKey: A ByteBuffer of a UTF-8 encoded PEM file representing the public
//                  key for the key passed into kmsKeyId
//     - encryptionAlgorithm: Must be either RSAES_OAEP_SHA_256 or RSAES_OAEP_SHA_1
final CreateAwsKmsRsaKeyringInput createAwsKmsRsaKeyringInput =
        CreateAwsKmsRsaKeyringInput.builder()
                .kmsClient(KmsClient.create())
                .kmsKeyId(rsaKeyArn)
                .publicKey(publicKey)
                .encryptionAlgorithm(EncryptionAlgorithmSpec.RSAES_OAEP_SHA_256)
                .build();
IKeyring awsKmsRsaKeyring =
 matProv.CreateAwsKmsRsaKeyring(createAwsKmsRsaKeyringInput);
```

Contextes de chiffrement requis dans la version 3.x

Avec la version 3. x duKit SDK de chiffrement AWS pour Java, vous pouvez utiliser le contexte de chiffrement requis CMM pour exiger des contextes de chiffrement dans vos opérations cryptographiques. Un contexte de chiffrement est un ensemble de paires clé-valeur non secrètes. Le contexte de chiffrement est lié cryptographiquement aux données chiffrées, de sorte que le même contexte de chiffrement est requis pour déchiffrer le champ. Lorsque vous utilisez le contexte de

chiffrement requis CMM, vous pouvez spécifier une ou plusieurs clés de contexte de chiffrement requises (clés obligatoires) qui doivent être incluses dans tous les appels de chiffrement et de déchiffrement.

> **Note**
>
> Le contexte de chiffrement requis CMM n'est interopérable qu'avec la version 4. x du AWS Encryption SDK pour .NET. Il n'est interopérable avec aucune autre implémentation de langage de programmation. Si vous chiffrez des données à l'aide du contexte de chiffrement requis CMM, vous ne pouvez les déchiffrer qu'avec la version 3. x de la version 4 de l'Kit SDK de chiffrement AWS pour Javaor. x du AWS Encryption SDK pour .NET.

Lors du chiffrement, il AWS Encryption SDK vérifie que toutes les clés de contexte de chiffrement requises sont incluses dans le contexte de chiffrement que vous avez spécifié. Les AWS Encryption SDK signes indiquent les contextes de chiffrement que vous avez spécifiés. Seules les paires clé-valeur qui ne sont pas des clés obligatoires sont sérialisées et stockées en texte clair dans l'en-tête du message chiffré renvoyé par l'opération de chiffrement.

Lors du déchiffrement, vous devez fournir un contexte de chiffrement contenant toutes les paires clé-valeur représentant les clés requises. AWS Encryption SDKUtilise ce contexte de chiffrement et les paires clé-valeur stockées dans l'en-tête du message chiffré pour reconstruire le contexte de chiffrement d'origine que vous avez spécifié lors de l'opération de chiffrement. S'il est AWS Encryption SDK impossible de reconstruire le contexte de chiffrement d'origine, l'opération de déchiffrement échoue. Si vous fournissez une paire clé-valeur contenant la clé requise avec une valeur incorrecte, le message chiffré ne peut pas être déchiffré. Vous devez fournir la même paire clé-valeur que celle spécifiée lors du chiffrement.

> **Important**
>
> Réfléchissez bien aux valeurs que vous choisissez pour les clés requises dans votre contexte de chiffrement. Vous devez être en mesure de fournir à nouveau les mêmes clés et les valeurs correspondantes lors du déchiffrement. Si vous ne parvenez pas à reproduire les clés requises, le message crypté ne peut pas être déchiffré.

L'exemple suivant initialise un AWS KMS trousseau de clés avec le contexte de chiffrement requis CMM.

```
// Instantiate the AWS Encryption SDK
final AwsCrypto crypto = AwsCrypto.builder()
        .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
        .build();

// Create your encryption context
final Map<String, String> encryptionContext = new HashMap<>();
encryptionContext.put("encryption", "context");
encryptionContext.put("is not", "secret");
encryptionContext.put("but adds", "useful metadata");
encryptionContext.put("that can help you", "be confident that");
encryptionContext.put("the data you are handling", "is what you think it is");

// Create a list of required encryption contexts
final List<String> requiredEncryptionContextKeys = Arrays.asList("encryption",
 "context");

// Create the keyring
final MaterialProviders materialProviders = MaterialProviders.builder()
        .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
        .build();
final CreateAwsKmsKeyringInput keyringInput = CreateAwsKmsKeyringInput.builder()
        .kmsKeyId(keyArn)
        .kmsClient(KmsClient.create())
        .build();
IKeyring kmsKeyring = materialProviders.CreateAwsKmsKeyring(keyringInput);

// Create the required encryption context CMM
ICryptographicMaterialsManager cmm =
    materialProviders.CreateDefaultCryptographicMaterialsManager(
        CreateDefaultCryptographicMaterialsManagerInput.builder()
            .keyring(kmsKeyring)
            .build()
    );
ICryptographicMaterialsManager requiredCMM =
    materialProviders.CreateRequiredEncryptionContextCMM(
        CreateRequiredEncryptionContextCMMInput.builder()
            .requiredEncryptionContextKeys(requiredEncryptionContextKeys)
            .underlyingCMM(cmm)
            .build()
        );
```

Exemples Kit SDK de chiffrement AWS pour Java

Les exemples suivants montrent comment utiliser le kit Kit SDK de chiffrement AWS pour Java afin de chiffrer et de déchiffrer des données. Ces exemples montrent comment utiliser la version 3. x et versions ultérieures duKit SDK de chiffrement AWS pour Java. Version 3. x of the Kit SDK de chiffrement AWS pour Java remplace les fournisseurs de clés principales par des porte-clés. Pour les exemples utilisant des versions antérieures, recherchez votre version dans la liste des versions du aws-encryption-sdk-javaréférentiel sur GitHub.

Rubriques

- Chiffrement et déchiffrement de chaînes
- Chiffrement et déchiffrement de flux d'octets
- Chiffrer et déchiffrer des flux d'octets à l'aide d'un jeu de clés multiples

Chiffrement et déchiffrement de chaînes

L'exemple suivant montre comment utiliser la version 3. x de Kit SDK de chiffrement AWS pour Java pour chiffrer et déchiffrer des chaînes. Avant d'utiliser la chaîne, convertissez-la en un tableau d'octets.

Cet exemple utilise un AWS KMSporte-clés. Lorsque vous chiffrez avec un AWS KMS trousseau de clés, vous pouvez utiliser un ID de clé, un ARN de clé, un nom d'alias ou un ARN d'alias pour identifier les clés KMS. Lors du déchiffrement, vous devez utiliser un ARN de clé pour identifier les clés KMS.

Lorsque vous appelez la méthode `encryptData()`, elle renvoie un message chiffré (`CryptoResult`) qui inclut le texte chiffré, les clés de données chiffrées et le contexte de chiffrement. Lorsque vous appelez `getResult` sur l'objet `CryptoResult`, il renvoie une version de chaîne codée en base 64 du message chiffré que vous pouvez transmettre à la méthode `decryptData()`.

De même, lorsque vous appelez`decryptData()`, l'`CryptoResult`objet renvoyé contient le message en texte brut et un AWS KMS key identifiant. Avant que votre application renvoie le texte brut, vérifiez que l'ID AWS KMS key et le contexte de chiffrement du message chiffré sont ceux que vous attendez.

```
// Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
```

```
package com.amazonaws.crypto.keyrings;

import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CommitmentPolicy;
import com.amazonaws.encryptionsdk.CryptoResult;
import software.amazon.cryptography.materialproviders.IKeyring;
import software.amazon.cryptography.materialproviders.MaterialProviders;
import
 software.amazon.cryptography.materialproviders.model.CreateAwsKmsMultiKeyringInput;
import software.amazon.cryptography.materialproviders.model.MaterialProvidersConfig;

import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Collections;
import java.util.Map;

/**
 * Encrypts and then decrypts data using an AWS KMS Keyring.
 *
 * <p>Arguments:
 *
 * <ol>
 *   <li>Key ARN: For help finding the Amazon Resource Name (ARN) of your AWS KMS
 customer master
 *       key (CMK), see 'Viewing Keys' at
 *       http://docs.aws.amazon.com/kms/latest/developerguide/viewing-keys.html
 * </ol>
 */
public class BasicEncryptionKeyringExample {

  private static final byte[] EXAMPLE_DATA = "Hello
 World".getBytes(StandardCharsets.UTF_8);

  public static void main(final String[] args) {
    final String keyArn = args[0];

    encryptAndDecryptWithKeyring(keyArn);
  }

  public static void encryptAndDecryptWithKeyring(final String keyArn) {
    // 1. Instantiate the SDK
    // This builds the AwsCrypto client with the RequireEncryptRequireDecrypt
 commitment policy,
```

```
    // which means this client only encrypts using committing algorithm suites and
 enforces
    // that the client will only decrypt encrypted messages that were created with a
 committing
    // algorithm suite.
    // This is the default commitment policy if you build the client with
    // `AwsCrypto.builder().build()`
    // or `AwsCrypto.standard()`.
    final AwsCrypto crypto =
        AwsCrypto.builder()
            .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
            .build();

    // 2. Create the AWS KMS keyring.
    // This example creates a multi keyring, which automatically creates the KMS
 client.
    final MaterialProviders materialProviders =
        MaterialProviders.builder()
            .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
            .build();
    final CreateAwsKmsMultiKeyringInput keyringInput =
        CreateAwsKmsMultiKeyringInput.builder().generator(keyArn).build();
    final IKeyring kmsKeyring =
 materialProviders.CreateAwsKmsMultiKeyring(keyringInput);

    // 3. Create an encryption context
    // We recommend using an encryption context whenever possible
    // to protect integrity. This sample uses placeholder values.
    // For more information see:
    // blogs.aws.amazon.com/security/post/Tx2LZ6WBJJANTNW/How-to-Protect-the-Integrity-
of-Your-Encrypted-Data-by-Using-AWS-Key-Management
    final Map<String, String> encryptionContext =
        Collections.singletonMap("ExampleContextKey", "ExampleContextValue");

    // 4. Encrypt the data
    final CryptoResult<byte[], ?> encryptResult =
        crypto.encryptData(kmsKeyring, EXAMPLE_DATA, encryptionContext);
    final byte[] ciphertext = encryptResult.getResult();

    // 5. Decrypt the data
    final CryptoResult<byte[], ?> decryptResult =
        crypto.decryptData(
            kmsKeyring,
            ciphertext,
```

```
            // Verify that the encryption context in the result contains the
            // encryption context supplied to the encryptData method
            encryptionContext);

    // 6. Verify that the decrypted plaintext matches the original plaintext
    assert Arrays.equals(decryptResult.getResult(), EXAMPLE_DATA);
  }
}
```

Chiffrement et déchiffrement de flux d'octets

L'exemple suivant montre comment utiliser le kit AWS Encryption SDK pour chiffrer et déchiffrer des flux d'octets.

Cet exemple utilise un trousseau de clés AES brut.

Lors du chiffrement, cet exemple utilise la `AwsCrypto.builder() .withEncryptionAlgorithm()` méthode pour spécifier une suite d'algorithmes sans signatures numériques. Lors du déchiffrement, afin de garantir que le texte chiffré n'est pas signé, cet exemple utilise la méthode. `createUnsignedMessageDecryptingStream()` La `createUnsignedMessageDecryptingStream()` méthode échoue si elle rencontre un texte chiffré avec une signature numérique.

Si vous chiffrez avec la suite d'algorithmes par défaut, qui inclut les signatures numériques, utilisez plutôt `createDecryptingStream()` cette méthode, comme indiqué dans l'exemple suivant.

```
// Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

package com.amazonaws.crypto.keyrings;

import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CommitmentPolicy;
import com.amazonaws.encryptionsdk.CryptoAlgorithm;
import com.amazonaws.encryptionsdk.CryptoInputStream;
import com.amazonaws.encryptionsdk.jce.JceMasterKey;
import com.amazonaws.util.IOUtils;
import software.amazon.cryptography.materialproviders.IKeyring;
import software.amazon.cryptography.materialproviders.MaterialProviders;
import software.amazon.cryptography.materialproviders.model.AesWrappingAlg;
import software.amazon.cryptography.materialproviders.model.CreateRawAesKeyringInput;
import software.amazon.cryptography.materialproviders.model.MaterialProvidersConfig;
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.security.SecureRandom;
import java.util.Collections;
import java.util.Map;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;


/**
 * <p>
 * Encrypts and then decrypts a file under a random key.
 *
 * <p>
 * Arguments:
 * <ol>
 * <li>Name of file containing plaintext data to encrypt
 * </ol>
 *
 * <p>
 * This program demonstrates using a standard Java {@link SecretKey} object as a {@link
 IKeyring} to
 * encrypt and decrypt streaming data.
 */
public class FileStreamingKeyringExample {
    private static String srcFile;

    public static void main(String[] args) throws IOException {
        srcFile = args[0];

        // In this example, we generate a random key. In practice,
        // you would get a key from an existing store
        SecretKey cryptoKey = retrieveEncryptionKey();

        // Create a Raw Aes Keyring using the random key and an AES-GCM encryption
 algorithm
        final MaterialProviders materialProviders = MaterialProviders.builder()
                .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
                .build();
        final CreateRawAesKeyringInput keyringInput =
 CreateRawAesKeyringInput.builder()
```

```
                .wrappingKey(ByteBuffer.wrap(cryptoKey.getEncoded()))
                .keyNamespace("Example")
                .keyName("RandomKey")
                .wrappingAlg(AesWrappingAlg.ALG_AES128_GCM_IV12_TAG16)
                .build();
        IKeyring keyring = materialProviders.CreateRawAesKeyring(keyringInput);

        // Instantiate the SDK.
        // This builds the AwsCrypto client with the RequireEncryptRequireDecrypt
commitment policy,
        // which means this client only encrypts using committing algorithm suites and
enforces
        // that the client will only decrypt encrypted messages that were created with
a committing
        // algorithm suite.
        // This is the default commitment policy if you build the client with
        // `AwsCrypto.builder().build()`
        // or `AwsCrypto.standard()`.
        // This example encrypts with an algorithm suite that doesn't include signing
for faster decryption,
        // since this use case assumes that the contexts that encrypt and decrypt are
equally trusted.
        final AwsCrypto crypto = AwsCrypto.builder()
                .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)

.withEncryptionAlgorithm(CryptoAlgorithm.ALG_AES_256_GCM_HKDF_SHA512_COMMIT_KEY)
                .build();

        // Create an encryption context to identify the ciphertext
        Map<String, String> context = Collections.singletonMap("Example",
"FileStreaming");

        // Because the file might be too large to load into memory, we stream the data,
instead of
        //loading it all at once.
        FileInputStream in = new FileInputStream(srcFile);
        CryptoInputStream<JceMasterKey> encryptingStream =
crypto.createEncryptingStream(keyring, in, context);

        FileOutputStream out = new FileOutputStream(srcFile + ".encrypted");
        IOUtils.copy(encryptingStream, out);
        encryptingStream.close();
        out.close();
```

```
        // Decrypt the file. Verify the encryption context before returning the
 plaintext.
        // Since the data was encrypted using an unsigned algorithm suite, use the
 recommended
        // createUnsignedMessageDecryptingStream method, which only accepts unsigned
 messages.
        in = new FileInputStream(srcFile + ".encrypted");
        CryptoInputStream<JceMasterKey> decryptingStream =
 crypto.createUnsignedMessageDecryptingStream(keyring, in);
        // Does it contain the expected encryption context?
        if
 (!"FileStreaming".equals(decryptingStream.getCryptoResult().getEncryptionContext().get("Exampl
 {
            throw new IllegalStateException("Bad encryption context");
        }

        // Write the plaintext data to disk.
        out = new FileOutputStream(srcFile + ".decrypted");
        IOUtils.copy(decryptingStream, out);
        decryptingStream.close();
        out.close();
    }

    /**
     * In practice, this key would be saved in a secure location.
     * For this demo, we generate a new random key for each operation.
     */
    private static SecretKey retrieveEncryptionKey() {
        SecureRandom rnd = new SecureRandom();
        byte[] rawKey = new byte[16]; // 128 bits
        rnd.nextBytes(rawKey);
        return new SecretKeySpec(rawKey, "AES");
    }
}
```

Chiffrer et déchiffrer des flux d'octets à l'aide d'un jeu de clés multiples

L'exemple suivant montre comment utiliser le AWS Encryption SDK avec un porte-clés multiple. Lorsque vous utilisez un porte-clés multiple pour chiffrer des données, toutes les clés d'encapsulage de tous ses porte-clés sont capables de déchiffrer les données. Cet exemple utilise un AWS KMSporte-clés et un porte-clés RSA brut comme porte-clés enfants.

Cet exemple chiffre avec la suite d'algorithmes par défaut, qui inclut une signature numérique. Lors du streaming, il AWS Encryption SDK publie du texte brut après des contrôles d'intégrité, mais avant que la signature numérique ne soit vérifiée. Pour éviter d'utiliser le texte en clair tant que la signature n'est pas vérifiée, cet exemple met le texte en clair en mémoire tampon et ne l'écrit sur le disque que lorsque le déchiffrement et la vérification sont terminés.

```
// Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

package com.amazonaws.crypto.keyrings;

import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CommitmentPolicy;
import com.amazonaws.encryptionsdk.CryptoOutputStream;
import com.amazonaws.util.IOUtils;
import software.amazon.cryptography.materialproviders.IKeyring;
import software.amazon.cryptography.materialproviders.MaterialProviders;
import
 software.amazon.cryptography.materialproviders.model.CreateAwsKmsMultiKeyringInput;
import software.amazon.cryptography.materialproviders.model.CreateMultiKeyringInput;
import software.amazon.cryptography.materialproviders.model.CreateRawRsaKeyringInput;
import software.amazon.cryptography.materialproviders.model.MaterialProvidersConfig;
import software.amazon.cryptography.materialproviders.model.PaddingScheme;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.nio.ByteBuffer;
import java.security.GeneralSecurityException;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.util.Collections;

/**
 * <p>
 * Encrypts a file using both AWS KMS Key and an asymmetric key pair.
 *
 * <p>
 * Arguments:
 * <ol>
 * <li>Key ARN: For help finding the Amazon Resource Name (ARN) of your AWS KMS key,
```

```
 *   see 'Viewing Keys' at http://docs.aws.amazon.com/kms/latest/developerguide/
viewing-keys.html
 *
 * <li>Name of file containing plaintext data to encrypt
 * </ol>
 * <p>
 * You might use AWS Key Management Service (AWS KMS) for most encryption and
 decryption operations, but
 * still want the option of decrypting your data offline independently of AWS KMS. This
 sample
 * demonstrates one way to do this.
 * <p>
 * The sample encrypts data under both an AWS KMS key and an "escrowed" RSA key pair
 * so that either key alone can decrypt it. You might commonly use the AWS KMS key for
 decryption. However,
 * at any time, you can use the private RSA key to decrypt the ciphertext independent
 of AWS KMS.
 * <p>
 * This sample uses the RawRsaKeyring to generate a RSA public-private key pair
 * and saves the key pair in memory. In practice, you would store the private key in a
 secure offline
 * location, such as an offline HSM, and distribute the public key to your development
 team.
 */
public class EscrowedEncryptKeyringExample {
    private static ByteBuffer publicEscrowKey;
    private static ByteBuffer privateEscrowKey;

    public static void main(final String[] args) throws Exception {
        // This sample generates a new random key for each operation.
        // In practice, you would distribute the public key and save the private key in
 secure
        // storage.
        generateEscrowKeyPair();

        final String kmsArn = args[0];
        final String fileName = args[1];

        standardEncrypt(kmsArn, fileName);
        standardDecrypt(kmsArn, fileName);

        escrowDecrypt(fileName);
    }
```

```
    private static void standardEncrypt(final String kmsArn, final String fileName)
throws Exception {
        // Encrypt with the KMS key and the escrowed public key
        // 1. Instantiate the SDK
        // This builds the AwsCrypto client with the RequireEncryptRequireDecrypt
commitment policy,
        // which means this client only encrypts using committing algorithm suites and
enforces
        // that the client will only decrypt encrypted messages that were created with
a committing
        // algorithm suite.
        // This is the default commitment policy if you build the client with
        // `AwsCrypto.builder().build()`
        // or `AwsCrypto.standard()`.
        final AwsCrypto crypto = AwsCrypto.builder()
                .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
                .build();

        // 2. Create the AWS KMS keyring.
        // This example creates a multi keyring, which automatically creates the KMS
client.
        final MaterialProviders matProv = MaterialProviders.builder()
                .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
                .build();
        final CreateAwsKmsMultiKeyringInput keyringInput =
CreateAwsKmsMultiKeyringInput.builder()
                .generator(kmsArn)
                .build();
        IKeyring kmsKeyring = matProv.CreateAwsKmsMultiKeyring(keyringInput);

        // 3. Create the Raw Rsa Keyring with Public Key.
        final CreateRawRsaKeyringInput encryptingKeyringInput =
CreateRawRsaKeyringInput.builder()
                .keyName("Escrow")
                .keyNamespace("Escrow")
                .paddingScheme(PaddingScheme.OAEP_SHA512_MGF1)
                .publicKey(publicEscrowKey)
                .build();
        IKeyring rsaPublicKeyring =
matProv.CreateRawRsaKeyring(encryptingKeyringInput);

        // 4. Create the multi-keyring.
        final CreateMultiKeyringInput createMultiKeyringInput =
CreateMultiKeyringInput.builder()
```

```
                .generator(kmsKeyring)
                .childKeyrings(Collections.singletonList(rsaPublicKeyring))
                .build();
        IKeyring multiKeyring = matProv.CreateMultiKeyring(createMultiKeyringInput);

        // 5. Encrypt the file
        // To simplify this code example, we omit the encryption context. Production
code should always
        // use an encryption context.
        final FileInputStream in = new FileInputStream(fileName);
        final FileOutputStream out = new FileOutputStream(fileName + ".encrypted");
        final CryptoOutputStream<?> encryptingStream =
crypto.createEncryptingStream(multiKeyring, out);

        IOUtils.copy(in, encryptingStream);
        in.close();
        encryptingStream.close();
    }

    private static void standardDecrypt(final String kmsArn, final String fileName)
throws Exception {
        // Decrypt with the AWS KMS key and the escrow public key.

        // 1. Instantiate the SDK.
        // This builds the AwsCrypto client with the RequireEncryptRequireDecrypt
commitment policy,
        // which means this client only encrypts using committing algorithm suites and
enforces
        // that the client will only decrypt encrypted messages that were created with
a committing
        // algorithm suite.
        // This is the default commitment policy if you build the client with
        // `AwsCrypto.builder().build()`
        // or `AwsCrypto.standard()`.
        final AwsCrypto crypto = AwsCrypto.builder()
                .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
                .build();

        // 2. Create the AWS KMS keyring.
        // This example creates a multi keyring, which automatically creates the KMS
client.
        final MaterialProviders matProv = MaterialProviders.builder()
                .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
                .build();
```

```
        final CreateAwsKmsMultiKeyringInput keyringInput =
CreateAwsKmsMultiKeyringInput.builder()
                .generator(kmsArn)
                .build();
        IKeyring kmsKeyring = matProv.CreateAwsKmsMultiKeyring(keyringInput);

        // 3. Create the Raw Rsa Keyring with Public Key.
        final CreateRawRsaKeyringInput encryptingKeyringInput =
CreateRawRsaKeyringInput.builder()
                .keyName("Escrow")
                .keyNamespace("Escrow")
                .paddingScheme(PaddingScheme.OAEP_SHA512_MGF1)
                .publicKey(publicEscrowKey)
                .build();
        IKeyring rsaPublicKeyring =
matProv.CreateRawRsaKeyring(encryptingKeyringInput);

        // 4. Create the multi-keyring.
        final CreateMultiKeyringInput createMultiKeyringInput =
CreateMultiKeyringInput.builder()
                .generator(kmsKeyring)
                .childKeyrings(Collections.singletonList(rsaPublicKeyring))
                .build();
        IKeyring multiKeyring = matProv.CreateMultiKeyring(createMultiKeyringInput);

        // 5. Decrypt the file
        // To simplify this code example, we omit the encryption context. Production
code should always
        // use an encryption context.
        final FileInputStream in = new FileInputStream(fileName + ".encrypted");
        final FileOutputStream out = new FileOutputStream(fileName + ".decrypted");
        // Since we are using a signing algorithm suite, we avoid streaming decryption
directly to the output file,
        // to ensure that the trailing signature is verified before writing any
untrusted plaintext to disk.
        final ByteArrayOutputStream plaintextBuffer = new ByteArrayOutputStream();
        final CryptoOutputStream<?> decryptingStream =
crypto.createDecryptingStream(multiKeyring, plaintextBuffer);
        IOUtils.copy(in, decryptingStream);
        in.close();
        decryptingStream.close();
        final ByteArrayInputStream plaintextReader = new
ByteArrayInputStream(plaintextBuffer.toByteArray());
        IOUtils.copy(plaintextReader, out);
```

```
        out.close();
    }

    private static void escrowDecrypt(final String fileName) throws Exception {
        // You can decrypt the stream using only the private key.
        // This method does not call AWS KMS.

        // 1. Instantiate the SDK
        final AwsCrypto crypto = AwsCrypto.standard();

        // 2. Create the Raw Rsa Keyring with Private Key.
        final MaterialProviders matProv = MaterialProviders.builder()
                .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
                .build();
        final CreateRawRsaKeyringInput encryptingKeyringInput =
CreateRawRsaKeyringInput.builder()
                .keyName("Escrow")
                .keyNamespace("Escrow")
                .paddingScheme(PaddingScheme.OAEP_SHA512_MGF1)
                .publicKey(publicEscrowKey)
                .privateKey(privateEscrowKey)
                .build();
        IKeyring escrowPrivateKeyring =
matProv.CreateRawRsaKeyring(encryptingKeyringInput);


        // 3. Decrypt the file
        // To simplify this code example, we omit the encryption context. Production
code should always
        // use an encryption context.
        final FileInputStream in = new FileInputStream(fileName + ".encrypted");
        final FileOutputStream out = new FileOutputStream(fileName + ".deescrowed");
        final CryptoOutputStream<?> decryptingStream =
crypto.createDecryptingStream(escrowPrivateKeyring, out);
        IOUtils.copy(in, decryptingStream);
        in.close();
        decryptingStream.close();

    }

    private static void generateEscrowKeyPair() throws GeneralSecurityException {
        final KeyPairGenerator kg = KeyPairGenerator.getInstance("RSA");
        kg.initialize(4096); // Escrow keys should be very strong
        final KeyPair keyPair = kg.generateKeyPair();
```

```
        publicEscrowKey = RawRsaKeyringExample.getPEMPublicKey(keyPair.getPublic());
        privateEscrowKey = RawRsaKeyringExample.getPEMPrivateKey(keyPair.getPrivate());

    }
}
```

Kit SDK de chiffrement AWS pour JavaScript

LeKit SDK de chiffrement AWS pour JavaScriptest conçu pour fournir une bibliothèque de chiffrement côté client pour les développeurs qui écrivent des applications de navigateur web dansJavaScriptou des applications de serveur web dans Node.js.

Comme toutes les implémentations du kit AWS Encryption SDK, le kit Kit SDK de chiffrement AWS pour JavaScript propose les fonctions avancées en matière de protection des données. Il s'agit notamment du chiffrement d'enveloppe, de données authentifiées supplémentaires (données AAD) et de suites d'algorithmes de clés symétriques sécurisées, authentifiées, comme les clés AES-GCM 256 bits avec dérivation de clés et signature.

Toutes les implémentations spécifiques à la langue du AWS Encryption SDK sont conçues pour être interopérables, sous réserve des contraintes de la langue. Pour de plus amples informations sur les contraintes de langue pour JavaScript, veuillez consulter the section called “Compatibilité”.

En savoir plus

- Pour plus de détails sur la programmation avec leKit SDK de chiffrement AWS pour JavaScript, voir leaws-encryption-sdkjavascriptrepository surGitHub.
- Pour obtenir des exemples de programmation, voirthe section called “Exemples”et l'example-browseretexample-nodemodules dans leaws-encryption-sdkjavascriptrepository.
- Pour un exemple réel d'utilisation de laKit SDK de chiffrement AWS pour JavaScriptpour chiffrer des données dans une application Web, voirComment activer le chiffrement dans un navigateur avec leKit SDK de chiffrement AWS pour JavaScriptet Node.jsdans leAWSblogs de sécurité.

Rubriques

- Compatibilité du kit Kit SDK de chiffrement AWS pour JavaScript
- Installation du Kit SDK de chiffrement AWS pour JavaScript
- Modules dans le kit Kit SDK de chiffrement AWS pour JavaScript
- Exemples Kit SDK de chiffrement AWS pour JavaScript

Compatibilité du kit Kit SDK de chiffrement AWS pour JavaScript

Le kit Kit SDK de chiffrement AWS pour JavaScript est conçu pour être interopérable avec d'autres implémentations de langage du kit AWS Encryption SDK. Dans la plupart des cas, vous pouvez chiffrer des données avec le kit Kit SDK de chiffrement AWS pour JavaScript et les déchiffrer avec n'importe quelle autre implémentation de langage, y compris l'interface de ligne de commande AWS Encryption SDK. Et vous pouvez utiliser le kit Kit SDK de chiffrement AWS pour JavaScript pour déchiffrer les messages chiffrés produits par d'autres implémentations de langages du kit AWS Encryption SDK.

Toutefois, lorsque vous utilisez le kit Kit SDK de chiffrement AWS pour JavaScript, vous devez être conscient de certains problèmes de compatibilité dans l'implémentation de langage du kit JavaScript et dans les navigateurs web.

En outre, lorsque vous utilisez différentes implémentations de langage, assurez-vous de configurer les fournisseurs de clés principales, clés principales et porte-clés compatibles. Pour plus d'informations, consultez Compatibilité du porte-clés.

Compatibilité Kit SDK de chiffrement AWS pour JavaScript

L'implémentation JavaScript du kit AWS Encryption SDK diffère des autres implémentations de langage de la manière suivante :

- L'opération de chiffrement du kit Kit SDK de chiffrement AWS pour JavaScript ne renvoie pas de texte chiffré non encadré. Cependant, le kit Kit SDK de chiffrement AWS pour JavaScript déchiffre le texte chiffré encadré et non encadré renvoyé par d'autres implémentations de langage du kit AWS Encryption SDK.
- À partir de Node.js version 12.9.0, Node.js prend en charge les options d'encapsulage de clé RSA suivantes :
  - OAEP avec SHA1, SHA256, SHA384 ou SHA512
  - OAEP avec SHA1 et MGF1 avec SHA1
  - PKCS1v15
- Avant la version 12.9.0, Node.js ne prend en charge que les options d'encapsulage de clé RSA suivantes :
  - OAEP avec SHA1 et MGF1 avec SHA1
  - PKCS1v15

Compatibilité des navigateurs

Certains navigateurs web ne prennent pas en charge les opérations de chiffrement de base requises par Kit SDK de chiffrement AWS pour JavaScript. Vous pouvez compenser certaines opérations manquantes en configurant un repli pour leWebCryptoAPI implémentée par le navigateur.

Limites du navigateur web

Les limitations suivantes sont communes à tous les navigateurs web :

- LeWebCryptoL'API ne prend pas en charge l'encapsulage de clé PKCS1v15.
- Les navigateurs ne prennent pas en charge les clés 192 bits.

Opérations de chiffrement requises

Le kit Kit SDK de chiffrement AWS pour JavaScript nécessite les opérations suivantes dans les navigateurs web. Si un navigateur ne prend pas en charge ces opérations, il est incompatible avec le kit Kit SDK de chiffrement AWS pour JavaScript.

- Le navigateur doit inclure le kit `crypto.getRandomValues()`, qui est une méthode pour générer des valeurs cryptographiquement aléatoires. Pour de plus amples informations sur les versions de navigateur web qui prennent en charge`crypto.getRandomValues()`, voirPuis-je utiliser crypto.getRandomValues() ?.

Repli requis

Le kit Kit SDK de chiffrement AWS pour JavaScript nécessite les bibliothèques et opérations suivantes dans les navigateurs web. Si vous prenez en charge un navigateur web qui ne répond pas à ces exigences, vous devez configurer une solution de secours. Sinon, les tentatives d'utilisation du kit Kit SDK de chiffrement AWS pour JavaScript avec le navigateur échoueront.

- LeWebCryptoL'API, qui effectue des opérations cryptographiques de base dans les applications web, n'est pas disponible pour tous les navigateurs. Pour de plus amples informations sur les versions de navigateur web qui prennent en charge le chiffrement web, veuillez consulter Puis-je utiliser le chiffrement web ?.
- Les versions modernes du navigateur web Safari ne prennent pas en charge le chiffrement AES-GCM de zéro octets, requis par le kit AWS Encryption SDK. Si le navigateur implémente l'optionWebCryptomais ne peut pas utiliser AES-GCM pour chiffrer zéro octet,Kit SDK de

chiffrement AWS pour JavaScriptutilise la bibliothèque de secours uniquement pour le chiffrement zéro octet. Il utilise leWebCryptoAPI pour toutes les autres opérations.

Pour configurer une solution de secours pour l'une ou l'autre limitation, ajoutez les instructions suivantes à votre code. Dans la fonction configureFallback, spécifiez une bibliothèque qui prend en charge les fonctions manquantes. L'exemple suivant repose sur Microsoft ResearchJavaScriptBibliothèque de chiffrement (`msrcrypto`), mais vous pouvez le remplacer par une bibliothèque compatible. Pour un exemple complet, veuillez consulter fallback.ts.

```
import { configureFallback } from '@aws-crypto/client-browser'
configureFallback(msrCrypto)
```

Installation du Kit SDK de chiffrement AWS pour JavaScript

Le kit Kit SDK de chiffrement AWS pour JavaScript se compose d'une collection de modules interdépendants. Plusieurs des modules ne sont que des collections de modules conçus pour fonctionner ensemble. Certains modules sont conçus pour fonctionner indépendamment. Quelques modules sont obligatoires pour toutes les implémentations ; d'autres ne sont obligatoires que pour des cas particuliers. Pour plus d'informations sur les modules duAWS Encryption SDKpour JavaScript, consultezModules dans le kit Kit SDK de chiffrement AWS pour JavaScriptet le`README.md`fichier dans chacun des modules duaws-encryption-sdk-javascriptréférentiel sur GitHub.

> **Note**
>
> Toutes les versions duKit SDK de chiffrement AWS pour JavaScriptles versions antérieures à 2.0.0 se trouvent dans leend-of-supportphase.
> Vous pouvez la mettre à jour en toute sécurité depuis la version 2.0xet plus tard vers la dernière version duKit SDK de chiffrement AWS pour JavaScriptsans aucune modification du code ou des données. Cependant,nouvelles fonctions de sécuritéintroduit dans la version 2.0.xne sont pas rétrocompatibles. Pour effectuer une mise à jour depuis des versions antérieures à 1.7.xvers la version 2.0.xet plus tard, vous devez d'abord la mettre à jour vers la version la plus récente.xversion duKit SDK de chiffrement AWS pour JavaScript. Pour plus d'informations, consultez.Migrer votreAWS Encryption SDK

Pour installer les modules, utilisez le gestionnaire de packages npm.

Par exemple, pour installer le module `client-node`, qui inclut tous les modules dont vous avez besoin pour programmer avec le kit Kit SDK de chiffrement AWS pour JavaScript dans Node.js, utilisez la commande suivante.

```
npm install @aws-crypto/client-node
```

Pour installer le module `client-browser`, qui inclut tous les modules que vous devez programmer avec le Kit SDK de chiffrement AWS pour JavaScript dans le navigateur, utilisez la commande suivante.

```
npm install @aws-crypto/client-browser
```

Pour des exemples pratiques d'utilisation duKit SDK de chiffrement AWS pour JavaScript, consultez les exemples dans le`example-node`et`example-browser`des modules dans leaws-encryption-sdk-javascriptréférentiel sur GitHub.

Modules dans le kit Kit SDK de chiffrement AWS pour JavaScript

Les modules dans le kit Kit SDK de chiffrement AWS pour JavaScript facilitent l'installation du code dont vous avez besoin pour vos projets.

Modules pourJavaScriptNode.js

client-node

Comprend tous les modules dont vous avez besoin pour programmer avec le kit Kit SDK de chiffrement AWS pour JavaScript dans Node.js.

caching-materials-managerNœud -

Exporte les fonctions qui prennent en charge la fonction de mise en cache des clé de données dans le kit Kit SDK de chiffrement AWS pour JavaScript dans Node.js.

decrypt-node

Exporte les fonctions qui déchiffrent et vérifient les messages chiffrés représentant les données et les flux de données. Inclus dans le module `client-node`.

encrypt-node

Exporte des fonctions qui chiffrent et signent différents types de données. Inclus dans le module `client-node`.

example-node

Exporte des exemples pratiques de programmation avec le kit Kit SDK de chiffrement AWS pour JavaScript dans Node.js. Comprend des exemples de différents types de porte-clés et de différents types de données.

hkdf-node

Exporte une fonction de dérivation de clé basée sur HMAC (HKDF) que le kit Kit SDK de chiffrement AWS pour JavaScript dans Node.js utilise dans des suites d'algorithmes spécifiques. LeKit SDK de chiffrement AWS pour JavaScriptdans le navigateur utilise la fonction HKDF native dans leWebCryptoAPI.

integration-node

Définit des tests qui vérifient que le kit Kit SDK de chiffrement AWS pour JavaScript dans Node.js est compatible avec d'autres implémentations de langage du kit AWS Encryption SDK.

kms-keyring-node

Exporte les fonctions qui prennent en chargeAWS KMSporte-clés dans Node.js.

raw-aes-keyringNœud -

Exporte les fonctions qui prennent en charge les porte-clés AES brut dans Node.js.

raw-rsa-keyringNœud -

Exporte les fonctions qui prennent en charge les porte-clés RSA brut dans Node.js.

Modules pourJavaScriptNavigateur

client-browser

Comprend tous les modules dont vous avez besoin pour programmer avec le kit Kit SDK de chiffrement AWS pour JavaScript dans le navigateur.

caching-materials-managernavigateur

Exporte les fonctions qui prennent en charge la fonction de mise en cache des clés de données pour JavaScript dans le navigateur.

decrypt-browser

Exporte les fonctions qui déchiffrent et vérifient les messages chiffrés représentant les données et les flux de données.

encrypt-browser

Exporte des fonctions qui chiffrent et signent différents types de données.

example-browser

Exemples pratiques de programmation avec le kit Kit SDK de chiffrement AWS pour JavaScript dans le navigateur. Comprend des exemples de différents types de porte-clés et de différents types de données.

integration-browser

Définit des tests qui vérifient que le script du kit Kit SDK de chiffrement AWS pour Java dans le navigateur est compatible avec d'autres implémentations de langage du kit AWS Encryption SDK.

kms-keyring-browser

Exporte les fonctions qui prennent en chargeAWS KMSPorte-clésdans le navigateur.

raw-aes-keyringnavigateur

Exporte les fonctions qui prennent en charge les porte-clés AES brut dans le navigateur.

raw-rsa-keyringnavigateur

Exporte les fonctions qui prennent en charge les porte-clés RSA brut dans le navigateur.

Modules pour toutes les implémentations

cache-material

Prend en charge la fonction de mise en cache des clés de données. Fournit du code pour assembler les matériaux de chiffrement mis en cache avec chaque clé de données.

kms-keyring

Exporte les fonctions qui prennent en charge les porte-clés KMS.

material-management

Implémente le gestionnaire de matériaux de chiffrement (CMM).

raw-keyring

Exporte les fonctions requises pour les porte-clés AES et RSA brutes.

serialize

Exporte les fonctions que le kit SDK utilise pour sérialiser sa sortie.

web-crypto-backend

Exporte les fonctions qui utilisent leWebCryptoAPI dans leKit SDK de chiffrement AWS pour JavaScriptdans le navigateur.

Exemples Kit SDK de chiffrement AWS pour JavaScript

Les exemples suivants montrent comment utiliser le kit Kit SDK de chiffrement AWS pour JavaScript afin de chiffrer et de déchiffrer des données.

Vous trouverez d'autres exemples d'utilisation duKit SDK de chiffrement AWS pour JavaScriptdans leexample-nodeetexample-browserModules dans le kitaws-encryption-sdk-javascriptrepository surGitHub. Ces exemples de modules ne sont pas installés lorsque vous installez les modules `client-browser` ou `client-node`.

Consultez les exemples de code complets : Nœud :kms_simple.ts, Navigateur :kms_simple.ts

Rubriques

- Chiffrement des données à l'aide d'unAWS KMSPorte-clés
- Déchiffrement des données à l'aide d'unAWS KMSPorte-clés

Chiffrement des données à l'aide d'unAWS KMSPorte-clés

L'exemple suivant montre comment utiliser le kit Kit SDK de chiffrement AWS pour JavaScript pour chiffrer et déchiffrer une chaîne courte ou un tableau d'octets.

Cet exemple présente unAWS KMSPorte-clés, un type de porte-clés qui utilise unAWS KMS keypour générer et chiffrer des clés de données. Pour obtenir de l'aide pour créer unAWS KMS key, voirCréation de clésdans leAWS Key Management ServiceManuel du développeur. Pour obtenir de l'aide pour identifier leAWS KMS keysdans unAWS KMSporte-clés, voirIdentification AWS KMS keys dans un AWS KMS porte-clés

Étape 1 : Construction du porte-clés.

Création d'unAWS KMSporte-clés pour le chiffrement.

Lors du chiffrement avec unAWS KMSporte-clés, vous devez spécifier unclé du générateur, c'est-à-dire unAWS KMS keyqui est utilisé pour générer la clé de données en texte brut et la chiffrer. Vous pouvez également spécifier zéro ou plusieurs clés supplémentaires qui chiffrent la

même clé de données en texte brut. Le porte-clés renvoie la clé de données en texte brut et une copie chiffrée de cette clé de données pour chaque AWS KMS key du porte-clés, y compris la clé de générateur. Pour déchiffrer les données, vous devez déchiffrer l'une des clés de données chiffrées.

Pour spécifier les AWS KMS keys pour un porte- clés de chiffrement dans le kit Kit SDK de chiffrement AWS pour JavaScript, vous pouvez utiliser n'importe quel identificateur de clé AWS KMS pris en charge. Cet exemple utilise une clé de générateur, identifiée par son ARN d'alias, et une clé supplémentaire, identifiée par un ARN de clé.

> **Note**
>
> Si vous envisagez de réutiliser votreAWS KMSpour le déchiffrement, vous devez utiliser les ARN de clé pour identifier lesAWS KMS keysdans le porte-clés.

Avant d'exécuter ce code, remplacez les exemples d'identifiants AWS KMS key par des identifiants valides. Vous devez disposer des autorisations requises pour utiliser les AWS KMS keys dans le porte-clés.

JavaScript Browser

Commencez par fournir vos informations d'identification au navigateur. LeKit SDK de chiffrement AWS pour JavaScriptExemples d'avecwebpack.DefinePlugin, qui remplace les constantes d'informations d'identification par vos informations d'identification réelles. Mais vous pouvez utiliser n'importe quelle méthode pour fournir vos informations d'identification. Ensuite, utilisez les informations d'identification pour créer un client AWS KMS.

```
declare const credentials: {accessKeyId: string, secretAccessKey:string,
 sessionToken:string }

const clientProvider = getClient(KMS, {
  credentials: {
    accessKeyId,
    secretAccessKey,
    sessionToken
  }
})
```

Ensuite, spécifiez leAWS KMS keyspour la clé du générateur et la clé supplémentaire. Ensuite, créez unAWS KMSporte-clés utilisant leAWS KMSclient et leAWS KMS keys.

```
const generatorKeyId = 'arn:aws:kms:us-west-2:111122223333:alias/EncryptDecrypt'
const keyIds = ['arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab']

const keyring = new KmsKeyringBrowser({ clientProvider, generatorKeyId, keyIds })
```

JavaScript Node.js

```
const generatorKeyId = 'arn:aws:kms:us-west-2:111122223333:alias/EncryptDecrypt'
const keyIds = ['arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab']

const keyring = new KmsKeyringNode({ generatorKeyId, keyIds })
```

Étape 2 : Définissez le contexte de chiffrement.

Un contexte de chiffrement est un ensemble de données authentifiées supplémentaires non secrètes et arbitraires. Lorsque vous fournissez un contexte de chiffrement lors du chiffrement, le kit AWS Encryption SDK le lie de façon cryptographique au texte chiffré de sorte que le même contexte de chiffrement est requis pour déchiffrer les données. L'utilisation d'un contexte de chiffrement est facultative, mais nous vous le recommandons dans le cadre des bonnes pratiques.

Créez un objet simple qui inclut les paires de contexte de chiffrement. La clé et la valeur de chaque paire doivent être une chaîne.

JavaScript Browser

```
const context = {
  stage: 'demo',
  purpose: 'simple demonstration app',
  origin: 'us-west-2'
}
```

JavaScript Node.js

```
const context = {
  stage: 'demo',
  purpose: 'simple demonstration app',
  origin: 'us-west-2'
}
```

Étape 3 : Chiffrez les données.

Pour chiffrer les données en texte brut, appelez la fonction `encrypt`. Transmettre dans leAWS KMSle porte-clés, les données en texte brut et le contexte de chiffrement.

La fonction `encrypt` renvoie un message chiffré (`result`) qui contient les données chiffrées, les clés de données chiffrées et les métadonnées importantes, y compris le contexte de chiffrement et la signature.

Vous pouvez déchiffrer ce message à l'aide du kit AWS Encryption SDK pour n'importe quel langage de programmation pris en charge.

JavaScript Browser

```
const plaintext = new Uint8Array([1, 2, 3, 4, 5])

const { result } = await encrypt(keyring, plaintext, { encryptionContext:
 context })
```

JavaScript Node.js

```
const plaintext = 'asdf'

const { result } = await encrypt(keyring, plaintext, { encryptionContext:
 context })
```

Déchiffrement des données à l'aide d'unAWS KMSPorte-clés

Vous pouvez utiliser le kit Kit SDK de chiffrement AWS pour JavaScript pour déchiffrer le message chiffré et récupérer les données d'origine.

Dans cet exemple, nous déchiffrons les données que nous avons chiffrées dans l'exemple the section called “Chiffrement des données à l'aide d'unAWS KMSPorte-clés”.

Étape 1 : Construction du porte-clés.

Pour déchiffrer les données, transmettez le message chiffré (`result`) renvoyé par la fonction `encrypt`. Le message chiffré inclut les données chiffrées, les clés de données chiffrées et les métadonnées importantes, y compris le contexte de chiffrement et la signature.

Vous devez également spécifier unAWS KMSPorte-cléslors du déchiffrement. Vous pouvez utiliser le même porte-clés que celui utilisé pour chiffrer les données ou un autre porte-clés. Pour

réussir, au moins une AWS KMS key dans le porte-clés de déchiffrement doit être en mesure de déchiffrer l'une des clés de données chiffrées du message chiffré. Étant donné qu'aucune clé de données n'est générée, vous n'avez pas besoin de spécifier une clé de générateur dans un porte-clés de déchiffrement. Si vous le faites, la clé de générateur et les clés supplémentaires sont traitées de la même manière.

Pour spécifier unAWS KMS keypour un porte-clés de déchiffrement dans leKit SDK de chiffrement AWS pour JavaScript, vous devez utiliser leARN de clé. Sinon, la AWS KMS key n'est pas reconnue. Pour obtenir de l'aide pour identifier leAWS KMS keysdans unAWS KMSporte-clés, voirIdentification AWS KMS keys dans un AWS KMS porte-clés

> **Note**
>
> Si vous utilisez le même porte-clés pour chiffrer et déchiffrer, utilisez des ARN de clé pour identifier les AWS KMS keys dans le porte-clés.

Dans cet exemple, nous créons un porte-clés qui inclut une seule des AWS KMS keys dans le porte-clés de chiffrement. Avant d'exécuter ce code, remplacez l'exemple d'ARN de clé par un ARN valide. Vous devez bénéficier de l'autorisation `kms:Decrypt` sur la AWS KMS key.

JavaScript Browser

Commencez par fournir vos informations d'identification au navigateur. LeKit SDK de chiffrement AWS pour JavaScriptExemples d'avecwebpack.DefinePlugin, qui remplace les constantes d'informations d'identification par vos informations d'identification réelles. Mais vous pouvez utiliser n'importe quelle méthode pour fournir vos informations d'identification. Ensuite, utilisez les informations d'identification pour créer un client AWS KMS.

```
declare const credentials: {accessKeyId: string, secretAccessKey:string,
 sessionToken:string }

const clientProvider = getClient(KMS, {
  credentials: {
    accessKeyId,
    secretAccessKey,
    sessionToken
  }
})
```

Ensuite, créez unAWS KMSporte-clés utilisant leAWS KMSclient. Cet exemple utilise une seule des AWS KMS keys du porte-clés de chiffrement.

```
const keyIds = ['arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab']

const keyring = new KmsKeyringBrowser({ clientProvider, keyIds })
```

JavaScript Node.js

```
const keyIds = ['arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab']

const keyring = new KmsKeyringNode({ keyIds })
```

Étape 2 : Déchiffrement des données.

Appelez ensuite la fonction `decrypt`. Transmettez le porte-clés de déchiffrement que vous venez de créer (`keyring`) et le message chiffré renvoyé par la fonction `encrypt` (`result`). Le kit AWS Encryption SDK utilise le porte-clés pour déchiffrer l'une des clés de données chiffrées. Il utilise ensuite la clé de données en texte brut pour déchiffrer les données.

Si l'appel réussit, le champ `plaintext` contient les données en texte brut (déchiffrées). Le champ `messageHeader` contient des métadonnées sur le processus de déchiffrement, y compris le contexte de chiffrement utilisé pour déchiffrer les données.

JavaScript Browser

```
const { plaintext, messageHeader } = await decrypt(keyring, result)
```

JavaScript Node.js

```
const { plaintext, messageHeader } = await decrypt(keyring, result)
```

Étape 3 : Vérifiez le contexte de chiffrement.

Le contexte de chiffrement utilisé pour déchiffrer les données est inclus dans l'en-tête de message (`messageHeader`) renvoyé par la fonction `decrypt`. Avant que votre application renvoie les données en texte brut, vérifiez que le contexte de chiffrement que vous avez fourni lors du chiffrement est inclus dans le contexte de chiffrement utilisé lors du déchiffrement. Une

incompatibilité peut indiquer que les données ont été altérées ou que vous n'avez pas déchiffré le bon texte chiffré.

Lors de la vérification du contexte de chiffrement, la correspondance exacte n'est pas obligatoire. Lorsque vous utilisez un algorithme de chiffrement avec signature, le gestionnaire de matériaux de chiffrement (CMM) ajoute la clé de signature publique au contexte de chiffrement avant de chiffrer le message. Cependant, toutes les paires de contexte de chiffrement que vous avez soumises doivent être incluses dans le contexte de chiffrement qui a été renvoyé.

Tout d'abord, récupérez le contexte de chiffrement à partir de l'en-tête du message. Vérifiez ensuite que chaque paire clé-valeur du contexte de chiffrement d'origine (`context`) correspond à une paire clé-valeur dans le contexte de chiffrement renvoyé (`encryptionContext`).

JavaScript Browser

```
const { encryptionContext } = messageHeader

Object
  .entries(context)
  .forEach(([key, value]) => {
    if (encryptionContext[key] !== value) throw new Error('Encryption Context
 does not match expected values')
})
```

JavaScript Node.js

```
const { encryptionContext } = messageHeader

Object
  .entries(context)
  .forEach(([key, value]) => {
    if (encryptionContext[key] !== value) throw new Error('Encryption Context
 does not match expected values')
})
```

Si la vérification du contexte de chiffrement aboutit, vous pouvez renvoyer les données en texte brut.

Kit SDK de chiffrement AWS pour Python

Cette rubrique explique comment installer et utiliser le kit Kit SDK de chiffrement AWS pour Python. Pour plus de détails sur la programmation avec le Kit SDK de chiffrement AWS pour Python, consultez le aws-encryption-sdk-pythonréférentiel sur GitHub. Pour la documentation de l'API, veuillez consulter Lire les documents.

Rubriques

Prérequis

Avant d'installer le Kit SDK de chiffrement AWS 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 les Kit SDK de chiffrement AWS pour Python versions 3.2.0 et ultérieures.
>
> Les versions antérieures du logiciel AWS Encryption SDK prennent en charge Python 2.7, Python 3.4 et versions ultérieures, mais nous vous recommandons d'utiliser la dernière version du AWS Encryption SDK.
>
> Pour télécharger Python, consultez Téléchargements Python.

Outil d'installation pip pour Python

> `pip`est inclus dans Python 3.6 et versions ultérieures, mais vous souhaiterez peut-être le mettre à niveau. Pour plus d'informations sur la mise à niveau ou l'installation`pip`, consultez la section Installation dans la `pip` documentation.

Installation

Installez la dernière version du Kit SDK de chiffrement AWS pour Python.

> **Note**
>
> Toutes les versions Kit SDK de chiffrement AWS pour Python antérieures à 3.0.0 sont en end-of-supportphase.
> Vous pouvez effectuer la mise à jour en toute sécurité à partir de la version 2.0. x et versions ultérieures vers la dernière version du AWS Encryption SDK sans aucune modification du code ou des données. Cependant, de nouvelles fonctionnalités de sécurité ont été introduites dans la version 2.0. x ne sont pas rétrocompatibles. Pour effectuer une mise à jour à partir de versions antérieures à 1.7. x vers la version 2.0. x et versions ultérieures, vous devez d'abord effectuer la mise à jour vers la dernière version 1. version x du AWS Encryption SDK. Pour plus de détails, consultez Migrer votreAWS Encryption SDK.

`pip`À utiliser pour installer le Kit SDK de chiffrement AWS pour Python, comme indiqué dans les exemples suivants.

Pour installer la dernière version

```
pip install aws-encryption-sdk
```

Pour plus d'informations sur l'utilisation de pip pour installer et mettre à niveau les packages, consultez Installation des packages.

Kit SDK de chiffrement AWS pour Python Nécessite la bibliothèque de cryptographie (pyca/cryptography) sur toutes les plateformes. Toutes les versions de installent et compilent `pip` automatiquement la `cryptography` bibliothèque sous Windows. `pip`8.1 et versions ultérieures s'installent et se compilent automatiquement sous `cryptography` Linux. Si vous utilisez une version antérieure de `pip` et que votre environnement Linux ne dispose pas des outils nécessaires pour créer la `cryptography` bibliothèque, vous devez les installer. Pour de plus amples informations, veuillez consulter Building cryptography on Linux.

Les versions 1.10.0 et 2.5.0 de la norme situent la Kit SDK de chiffrement AWS pour Python dépendance cryptographique entre 2.5.0 et 3.3.2. Les autres versions Kit SDK de chiffrement AWS pour Python installent la dernière version de cryptographie. Si vous avez besoin d'une version de cryptographie ultérieure à la version 3.3.2, nous vous recommandons d'utiliser la dernière version majeure du. Kit SDK de chiffrement AWS pour Python

Pour obtenir la dernière version de développement du Kit SDK de chiffrement AWS pour Python, rendez-vous dans le aws-encryption-sdk-pythonréférentiel dans GitHub.

Après avoir installé le Kit SDK de chiffrement AWS pour Python, commencez par consulter l'exemple de code Python présenté dans ce guide.

Exemple de code Kit SDK de chiffrement AWS pour Python

Les exemples suivants montrent comment utiliser le kit Kit SDK de chiffrement AWS pour Python afin de chiffrer et de déchiffrer des données.

Les exemples de cette section vous montrent comment utiliserversion 2.0.h/24, j/7et plus tard duKit SDK de chiffrement AWS pour Python. Pour obtenir des exemples utilisant des versions antérieures, recherchez votre version dans leVersionsliste desaws-encryption-sdk-pythonrepository sur GitHub.

Rubriques

- Chiffrement et déchiffrement de chaînes
- Chiffrement et déchiffrement de flux d'octets
- Chiffrement et déchiffrement des flux d'octets avec plusieurs fournisseurs de clés principales
- Utilisation de la mise en cache de clés de données pour chiffrer des messages

Chiffrement et déchiffrement de chaînes

L'exemple suivant montre comment utiliser le kit AWS Encryption SDK pour chiffrer et déchiffrer des chaînes. Cet exemple utilise uneAWS KMS keydansAWS Key Management Service(AWS KMS)en tant que clé principale.

Lors du chiffrement, le`StrictAwsKmsMasterKeyProvider`constructeur utilise un ID de clé, un ARN de clé, un nom d'alias ou un ARN d'alias. Lors du déchiffrement, ilnécessite un ARN de clé. Dans ce cas, parce que le`keyArn`est utilisé pour le chiffrement et le déchiffrement, sa valeur doit être un ARN clé. Pour plus d'informations sur les identifiants deAWS KMSclés, voirIdentificateurs clésdans leAWS Key Management ServiceManuel du développeur.

```
# Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You
# may not use this file except in compliance with the License. A copy of
# the License is located at
#
```

```
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is
# distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF
# ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.
"""Example showing basic encryption and decryption of a value already in memory."""
import aws_encryption_sdk
from aws_encryption_sdk import CommitmentPolicy


def cycle_string(key_arn, source_plaintext, botocore_session=None):
    """Encrypts and then decrypts a string under an &KMS; key.

    :param str key_arn: Amazon Resource Name (ARN) of the &KMS; key
    :param bytes source_plaintext: Data to encrypt
    :param botocore_session: existing botocore session instance
    :type botocore_session: botocore.session.Session
    """
    # Set up an encryption client with an explicit commitment policy. If you do not
 explicitly choose a
    # commitment policy, REQUIRE_ENCRYPT_REQUIRE_DECRYPT is used by default.
    client =
 aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQU

    # Create an AWS KMS master key provider
    kms_kwargs = dict(key_ids=[key_arn])
    if botocore_session is not None:
        kms_kwargs["botocore_session"] = botocore_session
    master_key_provider =
 aws_encryption_sdk.StrictAwsKmsMasterKeyProvider(**kms_kwargs)

    # Encrypt the plaintext source data
    ciphertext, encryptor_header = client.encrypt(source=source_plaintext,
 key_provider=master_key_provider)

    # Decrypt the ciphertext
    cycled_plaintext, decrypted_header = client.decrypt(source=ciphertext,
 key_provider=master_key_provider)

    # Verify that the "cycled" (encrypted, then decrypted) plaintext is identical to
 the source plaintext
    assert cycled_plaintext == source_plaintext
```

```
    # Verify that the encryption context used in the decrypt operation includes all key
pairs from
    # the encrypt operation. (The SDK can add pairs, so don't require an exact match.)
    #
    # In production, always use a meaningful encryption context. In this sample, we
omit the
    # encryption context (no key pairs).
    assert all(
        pair in decrypted_header.encryption_context.items() for pair in
encryptor_header.encryption_context.items()
    )
```

Chiffrement et déchiffrement de flux d'octets

L'exemple suivant montre comment utiliser le kit AWS Encryption SDK pour chiffrer et déchiffrer des flux d'octets. Cet exemple n'utilise pas AWS. Il utilise un fournisseur de clés principales éphémère et statique.

Lors du chiffrement, cet exemple utilise une suite d'algorithmes alternatifs sanssignatures numériques(`AES_256_GCM_HKDF_SHA512_COMMIT_KEY`). Cette suite d'algorithmes est appropriée lorsque les utilisateurs qui chiffrent et déchiffrent des données sont également fiables. Ensuite, lors du déchiffrement, l'exemple utilise le`decrypt-unsigned`mode streaming, qui échoue s'il rencontre un texte chiffré signé. Le`decrypt-unsigned`le mode streaming est introduit dansAWS Encryption SDKversions 1.9.h/24, j/7et 2.2.h/24, j/7.

```
# Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You
# may not use this file except in compliance with the License. A copy of
# the License is located at
#
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is
# distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF
# ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.
"""Example showing creation and use of a RawMasterKeyProvider."""
import filecmp
import os

import aws_encryption_sdk
```

```
from aws_encryption_sdk.identifiers import Algorithm, CommitmentPolicy,
 EncryptionKeyType, WrappingAlgorithm
from aws_encryption_sdk.internal.crypto.wrapping_keys import WrappingKey
from aws_encryption_sdk.key_providers.raw import RawMasterKeyProvider


class StaticRandomMasterKeyProvider(RawMasterKeyProvider):
    """Randomly generates 256-bit keys for each unique key ID."""

    provider_id = "static-random"

    def __init__(self, **kwargs):  # pylint: disable=unused-argument
        """Initialize empty map of keys."""
        self._static_keys = {}

    def _get_raw_key(self, key_id):
        """Returns a static, randomly-generated symmetric key for the specified key
 ID.
        :param str key_id: Key ID
        :returns: Wrapping key that contains the specified static key
        :rtype: :class:`aws_encryption_sdk.internal.crypto.WrappingKey`
        """
        try:
            static_key = self._static_keys[key_id]
        except KeyError:
            static_key = os.urandom(32)
            self._static_keys[key_id] = static_key
        return WrappingKey(
            wrapping_algorithm=WrappingAlgorithm.AES_256_GCM_IV12_TAG16_NO_PADDING,
            wrapping_key=static_key,
            wrapping_key_type=EncryptionKeyType.SYMMETRIC,
        )


def cycle_file(source_plaintext_filename):
    """Encrypts and then decrypts a file under a custom static master key provider.
    :param str source_plaintext_filename: Filename of file to encrypt
    """
    # Set up an encryption client with an explicit commitment policy. Note that if you
 do not explicitly choose a
    # commitment policy, REQUIRE_ENCRYPT_REQUIRE_DECRYPT is used by default.
    client =
 aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQU
```

```
    # Create a static random master key provider
    key_id = os.urandom(8)
    master_key_provider = StaticRandomMasterKeyProvider()
    master_key_provider.add_master_key(key_id)

    ciphertext_filename = source_plaintext_filename + ".encrypted"
    cycled_plaintext_filename = source_plaintext_filename + ".decrypted"

    # Encrypt the plaintext source data
    # We can use an unsigning algorithm suite here under the assumption that the
contexts that encrypt
    # and decrypt are equally trusted.
    with open(source_plaintext_filename, "rb") as plaintext, open(ciphertext_filename,
"wb") as ciphertext:
        with client.stream(
            algorithm=Algorithm.AES_256_GCM_HKDF_SHA512_COMMIT_KEY,
            mode="e",
            source=plaintext,
            key_provider=master_key_provider,
        ) as encryptor:
            for chunk in encryptor:
                ciphertext.write(chunk)

    # Decrypt the ciphertext
    # We can use the recommended "decrypt-unsigned" streaming mode since we encrypted
with an unsigned algorithm suite.
    with open(ciphertext_filename, "rb") as ciphertext, open(cycled_plaintext_filename,
"wb") as plaintext:
        with client.stream(mode="decrypt-unsigned", source=ciphertext,
key_provider=master_key_provider) as decryptor:
            for chunk in decryptor:
                plaintext.write(chunk)

    # Verify that the "cycled" (encrypted, then decrypted) plaintext is identical to
the source
    # plaintext
    assert filecmp.cmp(source_plaintext_filename, cycled_plaintext_filename)

    # Verify that the encryption context used in the decrypt operation includes all key
pairs from
    # the encrypt operation
    #
    # In production, always use a meaningful encryption context. In this sample, we
omit the
```

```
    # encryption context (no key pairs).
    assert all(
        pair in decryptor.header.encryption_context.items() for pair in
 encryptor.header.encryption_context.items()
    )
    return ciphertext_filename, cycled_plaintext_filename
```

Chiffrement et déchiffrement des flux d'octets avec plusieurs fournisseurs de clés principales

L'exemple suivant vous montre comment utiliser le kit AWS Encryption SDK avec plusieurs fournisseurs de clés principales. Le fait d'utiliser plusieurs fournisseurs de clés principales crée une redondance si un fournisseur de clés principales n'est pas disponible pour le déchiffrement. Cet exemple utilise uneAWS KMS keyet une key pair RSA en tant que clés principales.

Cet exemple de chiffrement est effectué à l'aide dusuite d'algorithmes par défaut, qui comprend unesignature numérique. Lors de la diffusion en continu, leAWS Encryption SDKpublie du texte en clair après des contrôles d'intégrité, mais avant d'avoir vérifié la signature numérique. Pour éviter d'utiliser le texte brut tant que la signature n'est pas vérifiée, cet exemple met en mémoire tampon le texte brut et l'écrit sur le disque uniquement lorsque le déchiffrement et la vérification sont terminés.

```
# Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You
# may not use this file except in compliance with the License. A copy of
# the License is located at
#
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is
# distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF
# ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.
"""Example showing creation of a RawMasterKeyProvider, how to use multiple
master key providers to encrypt, and demonstrating that each master key
provider can then be used independently to decrypt the same encrypted message.
"""
import filecmp
import os

from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import serialization
```

```
from cryptography.hazmat.primitives.asymmetric import rsa

import aws_encryption_sdk
from aws_encryption_sdk.identifiers import CommitmentPolicy, EncryptionKeyType,
 WrappingAlgorithm
from aws_encryption_sdk.internal.crypto.wrapping_keys import WrappingKey
from aws_encryption_sdk.key_providers.raw import RawMasterKeyProvider


class StaticRandomMasterKeyProvider(RawMasterKeyProvider):
    """Randomly generates and provides 4096-bit RSA keys consistently per unique key
 id."""

    provider_id = "static-random"

    def __init__(self, **kwargs):  # pylint: disable=unused-argument
        """Initialize empty map of keys."""
        self._static_keys = {}

    def _get_raw_key(self, key_id):
        """Retrieves a static, randomly generated, RSA key for the specified key id.

        :param str key_id: User-defined ID for the static key
        :returns: Wrapping key that contains the specified static key
        :rtype: :class:`aws_encryption_sdk.internal.crypto.WrappingKey`
        """
        try:
            static_key = self._static_keys[key_id]
        except KeyError:
            private_key = rsa.generate_private_key(public_exponent=65537,
 key_size=4096, backend=default_backend())
            static_key = private_key.private_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PrivateFormat.PKCS8,
                encryption_algorithm=serialization.NoEncryption(),
            )
            self._static_keys[key_id] = static_key
        return WrappingKey(
            wrapping_algorithm=WrappingAlgorithm.RSA_OAEP_SHA1_MGF1,
            wrapping_key=static_key,
            wrapping_key_type=EncryptionKeyType.PRIVATE,
        )
```

```
def cycle_file(key_arn, source_plaintext_filename, botocore_session=None):
    """Encrypts and then decrypts a file using an AWS KMS master key provider and a
 custom static master
    key provider. Both master key providers are used to encrypt the plaintext file, so
 either one alone
    can decrypt it.

    :param str key_arn: Amazon Resource Name (ARN) of the &KMS; key
    (http://docs.aws.amazon.com/kms/latest/developerguide/viewing-keys.html)
    :param str source_plaintext_filename: Filename of file to encrypt
    :param botocore_session: existing botocore session instance
    :type botocore_session: botocore.session.Session
    """
    # "Cycled" means encrypted and then decrypted
    ciphertext_filename = source_plaintext_filename + ".encrypted"
    cycled_kms_plaintext_filename = source_plaintext_filename + ".kms.decrypted"
    cycled_static_plaintext_filename = source_plaintext_filename + ".static.decrypted"

    # Set up an encryption client with an explicit commitment policy. Note that if you
 do not explicitly choose a
    # commitment policy, REQUIRE_ENCRYPT_REQUIRE_DECRYPT is used by default.
    client =
aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQU

    # Create an AWS KMS master key provider
    kms_kwargs = dict(key_ids=[key_arn])
    if botocore_session is not None:
        kms_kwargs["botocore_session"] = botocore_session
    kms_master_key_provider =
aws_encryption_sdk.StrictAwsKmsMasterKeyProvider(**kms_kwargs)

    # Create a static master key provider and add a master key to it
    static_key_id = os.urandom(8)
    static_master_key_provider = StaticRandomMasterKeyProvider()
    static_master_key_provider.add_master_key(static_key_id)

    # Add the static master key provider to the AWS KMS master key provider
    #   The resulting master key provider uses AWS KMS master keys to generate (and
encrypt)
    #   data keys and static master keys to create an additional encrypted copy of each
data key.
    kms_master_key_provider.add_master_key_provider(static_master_key_provider)

    # Encrypt plaintext with both AWS KMS and static master keys
```

```
   with open(source_plaintext_filename, "rb") as plaintext, open(ciphertext_filename,
"wb") as ciphertext:
       with client.stream(source=plaintext, mode="e",
key_provider=kms_master_key_provider) as encryptor:
           for chunk in encryptor:
               ciphertext.write(chunk)

   # Decrypt the ciphertext with only the AWS KMS master key
   # Buffer the data in memory before writing to disk. This ensures verfication of the
digital signature before returning plaintext.
   with open(ciphertext_filename, "rb") as ciphertext,
open(cycled_kms_plaintext_filename, "wb") as plaintext:
       with client.stream(
           source=ciphertext, mode="d",
key_provider=aws_encryption_sdk.StrictAwsKmsMasterKeyProvider(**kms_kwargs)
       ) as kms_decryptor:
            plaintext.write(kms_decryptor.read())

   # Decrypt the ciphertext with only the static master key
   # Buffer the data in memory before writing to disk to ensure verfication of the
signature before returning plaintext.
   with open(ciphertext_filename, "rb") as ciphertext,
open(cycled_static_plaintext_filename, "wb") as plaintext:
       with client.stream(source=ciphertext, mode="d",
key_provider=static_master_key_provider) as static_decryptor:
            plaintext.write(static_decryptor.read())

   # Verify that the "cycled" (encrypted, then decrypted) plaintext is identical to
the source plaintext
   assert filecmp.cmp(source_plaintext_filename, cycled_kms_plaintext_filename)
   assert filecmp.cmp(source_plaintext_filename, cycled_static_plaintext_filename)

   # Verify that the encryption context in the decrypt operation includes all key
pairs from the
   # encrypt operation.
   #
   # In production, always use a meaningful encryption context. In this sample, we
omit the
   # encryption context (no key pairs).
   assert all(
       pair in kms_decryptor.header.encryption_context.items() for pair in
encryptor.header.encryption_context.items()
   )
   assert all(
```

```
        pair in static_decryptor.header.encryption_context.items()
        for pair in encryptor.header.encryption_context.items()
    )
    return (ciphertext_filename, cycled_kms_plaintext_filename,
cycled_static_plaintext_filename)
```

Utilisation de la mise en cache de clés de données pour chiffrer des messages

L'exemple suivant montre comment utiliser la mise en cache de clé de données dans le kit Kit SDK de chiffrement AWS pour Python. Il est conçu pour vous montrer comment configurer une instance decache local(LocalCryptoMaterialsCache) avec la valeur de capacité requise et une instance duMise en cache du gestionnaire de matériaux de chiffrement(mise en cache CMM) avecseuils de sécurité du cache.

Cet exemple très basique crée une fonction qui chiffre une chaîne fixe. Il vous permet de spécifier unAWS KMS key, la taille de cache requise (capacité) et la valeur d'âge maximum. Pour un exemple plus complexe et concret de mise en cache des clés de données, consultez Exemple de code de mise en cache des clés de données.

Bien qu'il soit facultative, cet exemple utilise également un contexte de chiffrement en tant que données authentifiées supplémentaires. Lorsque vous déchiffrez les données qui ont été chiffrées avec un contexte de chiffrement, assurez-vous que votre application vérifie que le contexte de chiffrement est celui que vous attendez avant de renvoyer la clé de données en texte brut à votre mandataire. Un contexte de chiffrement est un élément de bonne pratique de toute opération de chiffrement ou de déchiffrement, mais il joue un rôle spécifique dans la mise en cache des clés de données. Pour plus d'informations, consultez .Contexte de chiffrement : Sélection des entrées de cache.

```
# Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You
# may not use this file except in compliance with the License. A copy of
# the License is located at
#
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is
# distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF
# ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.
"""Example of encryption with data key caching."""
```

```
import aws_encryption_sdk
from aws_encryption_sdk import CommitmentPolicy


def encrypt_with_caching(kms_key_arn, max_age_in_cache, cache_capacity):
    """Encrypts a string using an &KMS; key and data key caching.

    :param str kms_key_arn: Amazon Resource Name (ARN) of the &KMS; key
    :param float max_age_in_cache: Maximum time in seconds that a cached entry can be
 used
    :param int cache_capacity: Maximum number of entries to retain in cache at once
    """
    # Data to be encrypted
    my_data = "My plaintext data"

    # Security thresholds
    #   Max messages (or max bytes per) data key are optional
    MAX_ENTRY_MESSAGES = 100

    # Create an encryption context
    encryption_context = {"purpose": "test"}

    # Set up an encryption client with an explicit commitment policy. Note that if you
 do not explicitly choose a
    # commitment policy, REQUIRE_ENCRYPT_REQUIRE_DECRYPT is used by default.
    client =
 aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQU

    # Create a master key provider for the &KMS; key
    key_provider =
 aws_encryption_sdk.StrictAwsKmsMasterKeyProvider(key_ids=[kms_key_arn])

    # Create a local cache
    cache = aws_encryption_sdk.LocalCryptoMaterialsCache(cache_capacity)

    # Create a caching CMM
    caching_cmm = aws_encryption_sdk.CachingCryptoMaterialsManager(
        master_key_provider=key_provider,
        cache=cache,
        max_age=max_age_in_cache,
        max_messages_encrypted=MAX_ENTRY_MESSAGES,
    )

    # When the call to encrypt data specifies a caching CMM,
```

```
    # the encryption operation uses the data key cache specified
    # in the caching CMM
    encrypted_message, _header = client.encrypt(
        source=my_data, materials_manager=caching_cmm,
encryption_context=encryption_context
    )

    return encrypted_message
```

Interface de ligne de commande AWS Encryption SDK

Dans laAWS Encryption SDKInterface ligne de commande (AWSLe chiffrement (CLI) vous permet d'utiliserAWS Encryption SDKpour crypter et déchiffrer les données de manière interactive sur la ligne de commande et dans des scripts. Vous n'avez pas besoin d'être un expert en chiffrement ou en programmation.

> **Note**
>
> Versions duAWSL'interface CLI de chiffrement antérieure à 4.0.0 se trouve dans leend-of-supportphase.
> Vous pouvez la mettre à jour la version 2.1 en toute sécurité.xet plus tard vers la dernière version duAWSCryptage CLI sans aucune modification du code ou des données. Cependant,nouvelles fonctions de sécuritéintroduite dans la version 2.1.xne sont pas rétrocompatibles. Pour la mise à jour depuis la version 1.7.xou une version antérieure, vous devez d'abord la mettre à jour vers la version 1.xversion duAWSCLI de chiffrement. Pour plus d'informations, consultez.Migrer votreAWS Encryption SDK
> Les nouvelles fonctionnalités de sécurité ont été initialement publiées enAWSChiffrement CLI version 1.7.xet 2.0.x. Cependant,AWSVersion 1.8 de la CLI de chiffrementxremplace la version 1.7.xetAWSChiffrement CLI 2.1.xremplace 2.0.x. Pour plus d'informations, consultez la :avis de sécuritédans leaws-encryption-sdk-cliréférentiel sur GitHub.

Comme toutes les implémentations deAWS Encryption SDK, leAWSEncryption CLI propose des fonctionnalités avancées de protection des données Il s'agit dechiffrement de l'enveloppe, des données authentifiées supplémentaires (AAD) et une clé symétrique sécurisée, authentifiéesuites d'algorithmes, comme la clé de 256 bits avec dérivation de clé,Engagement clé, et signature.

Dans laAWSLa clé de chiffrement CLI est basée sur leKit SDK de chiffrement AWS pour Pythonet prise en charge sur Linux, macOS et Windows. Vous pouvez exécuter des commandes et des scripts

pour chiffrer et déchiffrer vos données dans votre shell préféré sous Linux ou macOS, dans une fenêtre d'invite de commande (cmd.exe) sous Windows et dans un PowerShell console sur n'importe quel système.

Toutes les implémentations spécifiques au langage deAWS Encryption SDK, y compris leAWSLes CLI de chiffrement sont interopérables. Par exemple, vous pouvez crypter des données à l'aide duKit SDK de chiffrement AWS pour Javaet décryptez-le à l'aide duAWSCLI de chiffrement.

Cette rubrique présenteAWSEncryption CLI explique comment l'installer et l'utiliser, et fournit plusieurs exemples pour vous aider à démarrer. Pour un démarrage rapide, voirComment crypter et déchiffrer vos données à l'aide duAWSCLI de chiffrementdans leAWSBlog sur la sécurité. Pour plus d'informations, consultez.Lire la documentation, et rejoignez-nous pour développerAWSCLI de chiffrement dansaws-encryption-sdk-cliréférentiel sur GitHub.

Performances

Dans laAWSLa clé de chiffrement CLI est basée sur leKit SDK de chiffrement AWS pour Python. Chaque fois que vous exécutez l'interface de ligne de commande, vous démarrez une nouvelle instance du moteur d'exécution Python. Pour améliorer les performances, dans la mesure du possible, utilisez une seule commande au lieu d'une série de commandes indépendantes. Par exemple, exécutez une commande qui traite les fichiers d'un répertoire récursivement au lieu d'exécuter des commandes séparées pour chaque fichier.

Rubriques

- Installation de l'interface AWS Encryption SDK de ligne de commande
- Comment utiliser l'interface de ligne de commande AWS de chiffrement
- Exemples deAWSCLI de chiffrement
- Référence des paramètres et de la syntaxe de l'interface de ligne de commande du kit AWS Encryption SDK
- Versions duAWSCLI de chiffrement

Installation de l'interface AWS Encryption SDK de ligne de commande

Cette rubrique explique comment installer la CLI AWS de chiffrement. Pour des informations détaillées, consultez le aws-encryption-sdk-cliréférentiel sur GitHub et lisez les documents.

Rubriques

Installation des éléments prérequis

La CLI de AWS chiffrement est basée sur le Kit SDK de chiffrement AWS pour Python. Pour installer la CLI de AWS chiffrement, vous avez besoin de Python et `pip` de l'outil de gestion de paquets Python. Python et `pip` sont disponibles sur toutes les plateformes prises en charge.

Installez les prérequis suivants avant d'installer la CLI de AWS chiffrement :

Python

Python 3.8 ou version ultérieure est requis par les versions 4.2.0 et ultérieures de la CLI de AWS chiffrement.

Les versions antérieures de la CLI de AWS chiffrement prennent en charge Python 2.7, 3.4 et versions ultérieures, mais nous vous recommandons d'utiliser la dernière version de la CLI de AWS chiffrement.

Python est inclus dans la plupart des installations Linux et macOS, mais vous devez passer à Python 3.6 ou version ultérieure. Nous vous recommandons d'utiliser la dernière version de Python. Sous Windows, vous devez installer Python ; il n'est pas installé par défaut. Pour télécharger et installer Python, consultez la section Téléchargements de Python.

Pour déterminer si Python est installé, dans la ligne de commande, tapez les informations suivantes.

```
python
```

Pour vérifier la version de Python, utilisez le paramètre `-V` (V majuscule).

```
python -V
```

Sous Windows, après avoir installé Python, ajoutez le chemin du `Python.exe` fichier à la valeur de la variable d'environnement Path.

Par défaut, Python est installé dans le répertoire de tous les utilisateurs ou dans un répertoire de profil utilisateur (`$home` ou `%userprofile%`) dans le sous-répertoire `AppData\Local`

`\Programs\Python`. Pour trouver l'emplacement du fichier `Python.exe` sur votre système, consultez l'une des clés de registre suivante. Vous pouvez l'utiliser PowerShell pour effectuer une recherche dans le registre.

```
PS C:\> dir HKLM:\Software\Python\PythonCore\version\InstallPath
# -or-
PS C:\> dir HKCU:\Software\Python\PythonCore\version\InstallPath
```

pip

`pip` est le gestionnaire de packages Python. Pour installer la CLI de AWS chiffrement et ses dépendances, vous devez disposer de la `pip` version 8.1 ou ultérieure. Pour obtenir de l'aide pour installer ou mettre à niveau `pip`, veuillez consulter Installation dans la documentation de `pip`.

Sur les installations Linux, les versions `pip` antérieures à 8.1 ne peuvent pas créer la bibliothèque de cryptographie requise par la CLI de AWS chiffrement. Si vous choisissez de ne pas mettre à jour votre `pip` version, vous pouvez installer les outils de compilation séparément. Pour plus d'informations, consultez Création du chiffrement sous Linux.

AWS Command Line Interface

Le AWS Command Line Interface (AWS CLI) n'est obligatoire que si vous utilisez AWS KMS keys in AWS Key Management Service (AWS KMS) avec la CLI de AWS chiffrement. Si vous utilisez un autre fournisseur de clé principale, ce n' AWS CLI est pas obligatoire.

Pour l'utiliser AWS KMS keys avec la CLI de AWS chiffrement, vous devez installer et configurer le AWS CLI. La configuration met les informations d'identification que vous utilisez pour vous authentifier à la AWS KMS disposition de la CLI de AWS chiffrement.

Installation et mise à jour de la CLI AWS de chiffrement

Installez la dernière version de la CLI de AWS chiffrement. Lorsque vous installez `pip` la CLI de AWS chiffrement, elle installe automatiquement les bibliothèques dont elle a besoin, notamment la Kit SDK de chiffrement AWS pour Pythonbibliothèque de cryptographie Python et le. AWS SDK for Python (Boto3)

> **Note**
>
> Les versions de la CLI de AWS chiffrement antérieures à la version 4.0.0 sont en end-of-supportcours de phase.

Vous pouvez effectuer la mise à jour en toute sécurité à partir de la version 2.1. x et versions ultérieures vers la dernière version de la CLI de AWS chiffrement sans aucune modification du code ou des données. Cependant, de nouvelles fonctionnalités de sécurité ont été introduites dans la version 2.1. x ne sont pas rétrocompatibles. Pour effectuer une mise à jour à partir de la version 1.7. x ou version antérieure, vous devez d'abord effectuer la mise à jour vers la dernière version 1. version x de la CLI AWS de chiffrement. Pour plus de détails, consultez Migrer votreAWS Encryption SDK.
Les nouvelles fonctionnalités de sécurité ont été initialement publiées dans les versions 1.7 de la CLI de AWS chiffrement. x et 2.0. x. Cependant, AWS Encryption CLI version 1.8. x remplace la version 1.7. x et CLI de AWS chiffrement 2.1. x remplace 2.0. x. Pour plus de détails, consultez l'avis de sécurité correspondant dans le aws-encryption-sdk-cliréférentiel sur GitHub.

Pour installer la dernière version de la CLI de AWS chiffrement

```
pip install aws-encryption-sdk-cli
```

Pour effectuer une mise à niveau vers la dernière version de la CLI de AWS chiffrement

```
pip install --upgrade aws-encryption-sdk-cli
```

Pour trouver les numéros de version de votre CLI de AWS chiffrement et AWS Encryption SDK

```
aws-encryption-cli --version
```

Le résultat répertorie les numéros de version des deux bibliothèques.

```
aws-encryption-sdk-cli/2.1.0 aws-encryption-sdk/2.0.0
```

Pour effectuer une mise à niveau vers la dernière version de la CLI de AWS chiffrement

```
pip install --upgrade aws-encryption-sdk-cli
```

L'installation de la CLI de AWS chiffrement installe également la dernière version de AWS SDK for Python (Boto3), si elle n'est pas déjà installée. Si Boto3 est installé, le programme d'installation vérifie la version de Boto3 et la met à jour si nécessaire.

Pour trouver la version de Boto3 que vous avez installée

```
pip show boto3
```

Pour effectuer la mise à jour vers la dernière version de Boto3

```
pip install --upgrade boto3
```

Pour installer la version de la CLI de AWS chiffrement actuellement en cours de développement, consultez le aws-encryption-sdk-cliréférentiel sur GitHub.

Pour de plus amples informations sur l'utilisation de `pip` pour installer et mettre à niveau les packages Python, veuillez consulter la documentation pip.

Comment utiliser l'interface de ligne de commande AWS de chiffrement

Cette rubrique explique comment utiliser les paramètres de l'interface de ligne de commande de AWS chiffrement. Pour obtenir des exemples, consultez Exemples deAWSCLI de chiffrement. Pour obtenir la documentation complète, consultez Lisez les documents. La syntaxe présentée dans ces exemples est celle de la version 2.1 de AWS Encryption CLI. x et versions ultérieures.

> **Note**
>
> Les versions de l'interface de ligne de commande de AWS chiffrement antérieures à la version 4.0.0 sont en phase e. nd-of-support
> Vous pouvez effectuer la mise à jour en toute sécurité à partir de la version 2.1. x et versions ultérieures vers la dernière version de l'interface de ligne de commande de AWS chiffrement sans aucune modification du code ou des données. Toutefois, de nouvelles fonctionnalités de sécurité ont été introduites dans la version 2.1. x ne sont pas rétrocompatibles. Pour effectuer une mise à jour depuis la version 1.7. x ou version antérieure, vous devez d'abord effectuer la mise à jour vers la dernière version 1. version x de l'interface de ligne de commande de AWS chiffrement. Pour plus de détails, consultez Migrer votreAWS Encryption SDK.
> De nouvelles fonctionnalités de sécurité ont été initialement publiées dans les versions 1.7 d'AWSEncryption CLI. x et 2.0. x. Toutefois, AWS Encryption CLI version 1.8. x remplace la version 1.7. x et AWS Encryption CLI 2.1. x remplace la version 2.0. x. Pour plus de détails, consultez l'avis de sécurité correspondant dans le aws-encryption-sdk-cliréférentiel surGitHub.

Pour un exemple montrant comment utiliser la fonctionnalité de sécurité qui limite les clés de données cryptées, voirLimiter les clés de données chiffrées.

Pour obtenir un exemple montrant comment utiliser les clés AWS KMS multirégions, reportez-vous Utilisation de plusieurs régions AWS KMS keys à la section.

Rubriques

- Procédure pour chiffrer et déchiffrer des données
- Comment spécifier les clés d'encapsulation
- Procédure pour fournir une entrée
- Procédure pour spécifier l'emplacement de sortie
- Procédure pour utiliser un contexte de chiffrement
- Comment définir une politique d'engagement
- Procédure pour stocker les paramètres dans un fichier de configuration

Procédure pour chiffrer et déchiffrer des données

La CLI de AWS chiffrement utilise les fonctionnalités du AWS Encryption SDK pour faciliter le chiffrement et le déchiffrement des données en toute sécurité.

> **Note**
>
> Le `--master-keys` paramètre est obsolète dans la version 1.8. x de l'interface de ligne de commande de AWS chiffrement et supprimé dans la version 2.1. x. Utilisez plutôt le paramètre `--wrapping-keys`. À partir de la version 2.1. x, le `--wrapping-keys` paramètre est obligatoire lors du chiffrement et du déchiffrement. Pour plus de détails, consultez Référence des paramètres et de la syntaxe de l'interface de ligne de commande du kit AWS Encryption SDK.

- Lorsque vous chiffrez des données dans la CLI de AWS chiffrement, vous spécifiez vos données en texte brut et une clé d'encapsulation (ou clé principale), telle qu'un AWS KMS key in AWS Key Management Service (). AWS KMS Si vous utilisez un fournisseur de clé principale personnalisé, vous devez également spécifier le fournisseur. Vous spécifiez également les emplacements de sortie pour le message chiffré et pour les métadonnées relatives à l'opération de chiffrement. Le contexte de chiffrement est facultatif, mais il est recommandé.

Dans la version 1.8. x, le `--commitment-policy` paramètre est obligatoire lorsque vous l'`--wrapping-keys`utilisez ; sinon, il n'est pas valide. À partir de la version 2.1. x, le `--commitment-policy` paramètre est facultatif, mais recommandé.

```
aws-encryption-cli --encrypt --input myPlaintextData \
                   --wrapping-keys key=1234abcd-12ab-34cd-56ef-1234567890ab \
                   --output myEncryptedMessage \
                   --metadata-output ~/metadata \
                   --encryption-context purpose=test \
                   --commitment-policy require-encrypt-require-decrypt
```

L'AWSinterface de ligne de commande de chiffrement chiffre vos données à l'aide d'une clé de données unique. Il chiffre ensuite la clé de données sous les clés d'encapsulation que vous avez spécifiées. Elle renvoie un message chiffré et des métadonnées sur l'opération. Le message chiffré contient vos données chiffrées (texte chiffré) et une copie chiffrée de la clé de données. Vous n'avez pas à vous soucier du stockage, de la gestion ou de la perte de la clé de données.

- Lorsque vous déchiffrez des données, vous transmettez votre message, le contexte de chiffrement facultatif, et l'emplacement de la sortie en texte brut et des métadonnées. Vous devez également spécifier les clés d'encapsulation que la CLI de AWS chiffrement peut utiliser pour déchiffrer le message, ou indiquer à l'interface de ligne de commande de AWS chiffrement qu'elle peut utiliser n'importe quelle clé d'encapsulation qui a chiffré le message.

  À partir de la version 1.8. x, le `--wrapping-keys` paramètre est facultatif lors du déchiffrement, mais recommandé. À partir de la version 2.1. x, le `--wrapping-keys` paramètre est obligatoire lors du chiffrement et du déchiffrement.

  Lors du déchiffrement, vous pouvez utiliser l'attribut key du `--wrapping-keys` paramètre pour spécifier les clés d'encapsulation qui déchiffrent vos données. La spécification d'une clé AWS KMS d'encapsulation lors du déchiffrement est facultative, mais il s'agit d'une bonne pratique qui vous empêche d'utiliser une clé que vous n'aviez pas l'intention d'utiliser. Si vous utilisez un fournisseur de clé principale personnalisé, vous devez spécifier le fournisseur et la clé d'encapsulation.

  Si vous n'utilisez pas l'attribut key, vous devez définir l'attribut discovery du `--wrapping-keys` paramètre sur`true`, ce qui permet à la CLI de AWS chiffrement de déchiffrer à l'aide de n'importe quelle clé d'encapsulation ayant chiffré le message.

Il est recommandé d'utiliser ce `--max-encrypted-data-keys` paramètre pour éviter de déchiffrer un message mal formé comportant un nombre excessif de clés de données cryptées. Spécifiez le nombre attendu de clés de données cryptées (une pour chaque clé d'encapsulage utilisée pour le chiffrement) ou un maximum raisonnable (par exemple 5). Pour plus de détails, consultez Limiter les clés de données chiffrées.

Le `--buffer` paramètre renvoie du texte en clair uniquement après le traitement de toutes les entrées, y compris la vérification de la signature numérique s'il en existe une.

Le `--decrypt-unsigned` paramètre déchiffre le texte chiffré et garantit que les messages ne sont pas signés avant le déchiffrement. Utilisez ce paramètre si vous l'`--algorithm`avez utilisé et sélectionné une suite d'algorithmes sans signature numérique pour crypter les données. Si le texte chiffré est signé, le déchiffrement échoue.

Vous pouvez utiliser `--decrypt` ou `--decrypt-unsigned` pour le déchiffrement, mais pas les deux.

```
aws-encryption-cli --decrypt --input myEncryptedMessage \
                   --wrapping-keys key=1234abcd-12ab-34cd-56ef-1234567890ab \
                   --output myPlaintextData \
                   --metadata-output ~/metadata \
                   --max-encrypted-data-keys 1 \
                   --buffer \
                   --encryption-context purpose=test \
                   --commitment-policy require-encrypt-require-decrypt
```

La CLI de AWS chiffrement utilise la clé d'encapsulation pour déchiffrer la clé de données du message crypté. Elle utilise ensuite la clé de données pour déchiffrer vos données. Elle renvoie vos données en texte brut et vos métadonnées sur l'opération.

Comment spécifier les clés d'encapsulation

Lorsque vous chiffrez des données dans l'interface de ligne de commande de AWS chiffrement, vous devez spécifier au moins une clé d'encapsulation (ou clé principale). Vous pouvez utiliser AWS KMS keys in AWS Key Management Service (AWS KMS), des clés d'encapsulation provenant d'un fournisseur de clé principale personnalisé, ou les deux. Le fournisseur de clés principales personnalisé peut être n'importe quel fournisseur de clés principales Python compatible.

Pour spécifier les clés d'encapsulation dans les versions 1.8. x et versions ultérieures, utilisez le `--wrapping-keys` paramètre (`-w`). La valeur de ce paramètre est un ensemble d'attributs au `attribute=value` format. Les attributs que vous utilisez dépendent du fournisseur de clés principales et de la commande.

- AWS KMS. Dans les commandes de chiffrement, vous devez spécifier un `--wrapping-keys` paramètre avec un attribut clé. À partir de la version 2.1. x, le `--wrapping-keys` paramètre est également requis dans les commandes de déchiffrement. Lors du déchiffrement, le `--wrapping-keys` paramètre doit comporter un attribut clé ou un attribut de découverte dont la valeur est `true` (mais pas les deux). Les autres attributs sont facultatifs.
- Fournisseur de clés principales personnalisé. Vous devez spécifier un `--wrapping-keys` paramètre dans chaque commande. La valeur du paramètre doit avoir des attributs key et provider.

Vous pouvez inclure plusieurs `--wrapping-keys` paramètres et plusieurs attributs clés dans la même commande.

Encapsulation des attributs des paramètres clés

La valeur du paramètre `--wrapping-keys` comprend les attributs suivants et leurs valeurs. Un `--wrapping-keys` paramètre (ou `--master-keys` paramètre) est requis dans toutes les commandes de chiffrement. À partir de la version 2.1. x, le `--wrapping-keys` paramètre est également requis lors du déchiffrement.

Si le nom d'un attribut ou la valeur d'un attribut inclut des espaces ou des caractères spéciaux, entourez le nom et la valeur de guillemets. Par exemple, `--wrapping-keys key=12345 "provider=my cool provider"`.

Clé : Spécifiez une clé d'encapsulation

Utilisez l'attribut key pour identifier une clé d'encapsulation. Lors du chiffrement, la valeur peut être n'importe quel identifiant de clé reconnu par le fournisseur de clé principale.

```
--wrapping-keys key=1234abcd-12ab-34cd-56ef-1234567890ab
```

Dans une commande de chiffrement, vous devez inclure au moins un attribut clé et une valeur. Pour chiffrer votre clé de données sous plusieurs clés d'encapsulation, utilisez plusieurs attributs de clé.

```
aws-encryption-cli --encrypt --wrapping-keys
 key=1234abcd-12ab-34cd-56ef-1234567890ab key=1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d
```

Dans les commandes de chiffrement qui utilisentAWS KMS keys, la valeur de la clé peut être l'ID de la clé, son ARN de clé, un nom d'alias ou un ARN d'alias. Par exemple, cette commande de chiffrement utilise un ARN d'alias dans la valeur de l'attribut key. Pour plus d'informations sur les identificateurs clés d'unAWS KMS key, consultez la section Identifiants clés du Guide du AWS Key Management Servicedéveloppeur.

```
aws-encryption-cli --encrypt --wrapping-keys key=arn:aws:kms:us-
west-2:111122223333:alias/ExampleAlias
```

Dans les commandes de déchiffrement qui utilisent un fournisseur de clés principales personnalisé, les attributs key et provider sont requis.

```
\\ Custom master key provider
aws-encryption-cli --decrypt --wrapping-keys provider='myProvider' key='100101'
```

Dans les commandes de déchiffrement qui utilisentAWS KMS, vous pouvez utiliser l'attribut key pour spécifier l'attribut AWS KMS keys à utiliser pour le déchiffrement, ou l'attribut discovery avec une valeur de`true`, qui permet à l'interface de ligne de commande de AWS chiffrement d'utiliser tout ce AWS KMS key qui a été utilisé pour chiffrer le message. Si vous spécifiez unAWS KMS key, il doit s'agir de l'une des clés d'encapsulation utilisées pour crypter le message.

La spécification de la clé d'encapsulation est une AWS Encryption SDKbonne pratique. Cela garantit que vous utilisez ce que AWS KMS key vous avez l'intention d'utiliser.

Dans une commande de déchiffrement, la valeur de l'attribut clé doit être un ARN clé.

```
\\ AWS KMS key
aws-encryption-cli --decrypt --wrapping-keys key=arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

Découverte : utilisez-en n'importe lequel AWS KMS key lors du décryptage

Si vous n'avez pas besoin de AWS KMS keys limiter l'utilisation lors du déchiffrement, vous pouvez utiliser l'attribut discovery avec une valeur de. `true` La valeur de `true` permet à l'interface de ligne de commande de AWS chiffrement de déchiffrer le message à l'aide de tout

AWS KMS key ce qui a chiffré le message. Si vous ne spécifiez aucun attribut de découverte, la détection est `false` (par défaut). L'attribut discovery n'est valide que dans les commandes de déchiffrement et uniquement lorsque le message a été chiffré avecAWS KMS keys.

L'attribut discovery avec une valeur de `true` est une alternative à l'utilisation de l'attribut clé pour spécifierAWS KMS keys. Lors du déchiffrement d'un message chiffré avecAWS KMS keys, chaque `--wrapping-keys` paramètre doit comporter un attribut clé ou un attribut de découverte dont la valeur est égale à`true`, mais pas les deux.

Lorsque la découverte est vraie, il est recommandé d'utiliser les attributs discovery-partition et discovery-account pour limiter ceux AWS KMS keys utilisés à ceux que vous avez spécifiés. Comptes AWS Dans l'exemple suivant, les attributs de découverte permettent à la CLI de AWS chiffrement d'utiliser n'importe lequel AWS KMS key des attributs spécifiésComptes AWS.

```
aws-encryption-cli --decrypt --wrapping-keys \
    discovery=true \
    discovery-partition=aws \
    discovery-account=111122223333 \
    discovery-account=444455556666
```

Fournisseur : Spécifiez le fournisseur de clé principale

L'attribut provider identifie le fournisseur de clés principales. La valeur par défaut est `aws-kms`, qui représente AWS KMS. Si vous utilisez un autre fournisseur de clés principales, l'attribut provider est requis.

```
--wrapping-keys key=12345 provider=my_custom_provider
```

Pour plus d'informations sur l'utilisation de fournisseurs de clés (nonAWS KMS) principales personnalisés, consultez la rubrique Configuration avancée dans le fichier README du référentiel AWSEncryption CLI.

Région : Spécifiez une Région AWS

Utilisez l'attribut region pour spécifier le nom Région AWS d'unAWS KMS key. L'attribut est valide uniquement dans les commandes de chiffrement et uniquement lorsque le fournisseur de clés principales est AWS KMS.

```
--encrypt --wrapping-keys key=alias/primary-key region=us-east-2
```

AWSLes commandes CLI de chiffrement utilisent la Région AWS valeur spécifiée dans la valeur de l'attribut clé si celle-ci inclut une région, telle qu'un ARN. Si la valeur de clé indique unRégion AWS, l'attribut de région est ignoré.

L'attribut region est prioritaire sur les autres spécifications relatives à la région. Si vous n'utilisez pas d'attribut de région, les commandes de la CLI de AWS chiffrement utilisent celui Région AWS spécifié dans votre profil AWS CLI nommé, le cas échéant, ou dans votre profil par défaut.

Profil : Spécifier un profil nommé

Utilisez l'attribut de profil pour spécifier un AWS CLIprofil nommé de l'. Les profils nommés peuvent inclure des informations d'identification et unRégion AWS. Cet attribut est valide uniquement lorsque le fournisseur de clés principales est AWS KMS.

```
--wrapping-keys key=alias/primary-key profile=admin-1
```

Vous pouvez utiliser l'attribut profile pour spécifier d'autres informations d'identification dans les commandes de chiffrement et de déchiffrement. Dans une commande de chiffrement, l'interface de ligne de commande de AWS chiffrement utilise le Région AWS dans le profil nommé uniquement lorsque la valeur de la clé n'inclut pas de région et qu'il n'existe aucun attribut de région. Dans une commande de déchiffrement, le profil Région AWS figurant dans le nom est ignoré.

Comment spécifier plusieurs clés d'encapsulation

Vous pouvez spécifier plusieurs clés d'encapsulation (ou clés principales) dans chaque commande.

Si vous spécifiez plusieurs clés d'encapsulation, la première clé d'encapsulation génère et chiffre la clé de données utilisée pour chiffrer vos données. Les autres clés d'encapsulation chiffrent la même clé de données. Le message crypté qui en résulte contient les données cryptées (« texte chiffré ») et un ensemble de clés de données cryptées, une cryptée par clé d'encapsulation. N'importe quel encapsulage peut déchiffrer une clé de données cryptée, puis déchiffrer les données.

Il existe deux manières de spécifier plusieurs clés d'encapsulation :

- Incluez plusieurs attributs clés dans la valeur du `--wrapping-keys` paramètre.

```
$key_oregon=arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
$key_ohio=arn:aws:kms:us-east-2:111122223333:key/0987ab65-43cd-21ef-09ab-87654321cdef
```

```
--wrapping-keys key=$key_oregon key=$key_ohio
```

- Inclure plusieurs paramètres `--wrapping-keys` dans la même commande. Utilisez cette syntaxe lorsque les valeurs d'attribut que vous spécifiez ne s'appliquent pas à toutes les touches d'encapsulation de la commande.

```
--wrapping-keys region=us-east-2 key=alias/test_key \
--wrapping-keys region=us-west-1 key=alias/test_key
```

L'attribut discovery dont la valeur est de `true` permet à l'interface de ligne de commande de AWS chiffrement d'utiliser AWS KMS key celui qui a chiffré le message. Si vous utilisez plusieurs `--wrapping-keys` paramètres dans la même commande, l'utilisation `discovery=true` de n'importe quel `--wrapping-keys` paramètre remplace effectivement les limites de l'attribut clé dans `--wrapping-keys` les autres paramètres.

Par exemple, dans la commande suivante, l'attribut clé du premier `--wrapping-keys` paramètre limite l'interface de ligne de commande de AWS chiffrement à la valeur spécifiéeAWS KMS key. Toutefois, l'attribut discovery du second `--wrapping-keys` paramètre permet à l'interface de ligne de commande de AWS chiffrement d'utiliser n'importe quel AWS KMS key compte spécifié pour déchiffrer le message.

```
aws-encryption-cli --decrypt \
    --wrapping-keys key=arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab \
    --wrapping-keys discovery=true \
                    discovery-partition=aws \
                    discovery-account=111122223333 \
                    discovery-account=444455556666
```

Procédure pour fournir une entrée

L'opération de chiffrement dans l'interface de ligne de commande de AWS chiffrement prend des données en texte brut comme entrée et renvoie un message crypté. L'opération de déchiffrement utilise un message chiffré comme entrée et renvoie des données en texte brut.

Le `--input` paramètre (`-i`), qui indique à la CLI de AWS chiffrement où trouver l'entrée, est obligatoire dans toutes les commandes de la CLI de AWS chiffrement.

Vous pouvez fournir une entrée des façons suivantes :

- Utiliser un fichier.

```
--input myData.txt
```

- Utiliser un modèle de nom de fichier.

```
--input testdir/*.xml
```

- Utiliser un répertoire ou un modèle de nom de répertoire. Lorsque l'entrée est un répertoire, le paramètre `--recursive` (`-r`, `-R`) est obligatoire.

```
--input testdir --recursive
```

- Dirigez l'entrée vers la commande (stdin). Utilisez une valeur de `-` pour le paramètre `--input`. (Le paramètre `--input` est toujours obligatoire.)

```
echo 'Hello World' | aws-encryption-cli --encrypt --input -
```

Procédure pour spécifier l'emplacement de sortie

Le `--output` paramètre indique à l'interface de ligne de commande de AWS chiffrement où écrire les résultats de l'opération de chiffrement ou de déchiffrement. Elle est requise dans chaque commande de l'interface AWS de ligne de commande de chiffrement. L'interface AWS de ligne de commande de chiffrement crée un nouveau fichier de sortie pour chaque fichier d'entrée de l'opération.

Si un fichier de sortie existe déjà, par défaut, l'interface de ligne de commande de AWS chiffrement affiche un avertissement, puis remplace le fichier. Pour empêcher le remplacement, utilisez le paramètre `--interactive`, qui vous invite à confirmer avant le remplacement, ou le paramètre `--no-overwrite`, qui ignore l'entrée si la sortie doit entraîner un remplacement. Pour supprimer l'avertissement de remplacement, utilisez `--quiet`. Pour capturer les erreurs et les avertissements provenant de l'interface de ligne de commande de AWS chiffrement, utilisez l'opérateur de `2>&1` redirection pour les écrire dans le flux de sortie.

> **Note**
>
> Les commandes qui remplacent les fichiers de sortie commencent par supprimer le fichier de sortie. Si la commande échoue, il est possible que le fichier de sortie soit déjà supprimé.

Vous pouvez spécifier l'emplacement de sortie de plusieurs façons.

- Spécifiez un nom de fichier. Si vous spécifiez un chemin d'accès au fichier, tous les répertoires du chemin doivent exister avant l'exécution de la commande.

  ```
  --output myEncryptedData.txt
  ```

- Spécifiez un répertoire. Le répertoire de sortie doit exister avant l'exécution de la commande.

  Si l'entrée contient des sous-répertoires, la commande reproduit les sous-répertoires sous le répertoire spécifié.

  ```
  --output Test
  ```

  Lorsque l'emplacement de sortie est un répertoire (sans nom de fichier), l'interface de ligne de commande de AWS chiffrement crée des noms de fichiers de sortie sur la base des noms des fichiers d'entrée et d'un suffixe. Les opérations de chiffrement ajoutent `.encrypted` au nom du fichier d'entrée et les opérations de déchiffrement ajoutent `.decrypted`. Pour modifier le suffixe, utilisez le paramètre `--suffix`.

  Par exemple, si vous chiffrez `file.txt`, la commande de chiffrement crée `file.txt.encrypted`. Si vous déchiffrez `file.txt.encrypted`, la commande de déchiffrement crée `file.txt.encrypted.decrypted`.

- Écrivez sur la ligne de commande (stdout). Entrez une valeur de `-` pour le paramètre `--output`. Vous pouvez utiliser `--output -` pour diriger la sortie vers une autre commande ou un autre programme.

  ```
  --output -
  ```

Procédure pour utiliser un contexte de chiffrement

L'interface de ligne de commande de AWS chiffrement vous permet de fournir un contexte de chiffrement dans les commandes de chiffrement et de déchiffrement. Ce contexte n'est pas obligatoire, mais c'est une bonne pratique de chiffrement que nous recommandons.

Un contexte de chiffrement est un type de données authentifiées supplémentaires non secrètes et arbitraires. Dans l'AWSinterface de ligne de commande de chiffrement, le contexte de chiffrement consiste en un ensemble de `name=value` paires. Vous pouvez utiliser n'importe quel contenu dans les paires, y compris les informations sur les fichiers, les données qui vous aident à identifier l'opération de chiffrement dans les journaux ou les données nécessaires pour vos droits ou vos stratégies.

Dans une commande de chiffrement

Le contexte de chiffrement que vous spécifiez dans une commande de chiffrement, ainsi que toutes paires supplémentaires ajoutées par le CMM, est lié de manière chiffrée aux données chiffrées. Il est également inclus (en texte brut) dans le message chiffré renvoyé par la commande. Si vous utilisez unAWS KMS key, le contexte de chiffrement peut également apparaître en texte clair dans les enregistrements et les journaux d'audit, tels queAWS CloudTrail.

L'exemple suivant illustre un contexte de chiffrement avec trois paires `name=value`.

```
--encryption-context purpose=test dept=IT class=confidential
```

Dans une commande de déchiffrement

Dans une commande de déchiffrement, le contexte de chiffrement vous permet de confirmer que vous déchiffrez le message chiffré correct.

Vous n'êtes pas tenu de fournir un contexte de chiffrement dans une commande de déchiffrement, même si un contexte de chiffrement a été utilisé pour le chiffrement. Toutefois, si vous le faites, l'interface de ligne de commande de AWS chiffrement vérifie que chaque élément du contexte de chiffrement de la commande de déchiffrement correspond à un élément du contexte de chiffrement du message crypté. Si certains éléments ne correspondent pas, la commande de déchiffrement échoue.

Par exemple, la commande suivante déchiffre le message chiffré uniquement si son contexte de chiffrement inclut `dept=IT`.

```
aws-encryption-cli --decrypt --encryption-context dept=IT ...
```

Un contexte de chiffrement est un élément important de votre stratégie de sécurité. Cependant, lorsque vous choisissez un contexte de chiffrement, n'oubliez pas que ses valeurs ne sont pas secrètes. N'incluez aucune donnée confidentielle dans le contexte de chiffrement.

Pour spécifier un contexte de chiffrement

- Dans une commande de chiffrement, utilisez le paramètre `--encryption-context` avec une ou plusieurs paires `name=value`. Séparez chaque paire par un espace.

```
--encryption-context name=value [name=value] ...
```

- Dans une commande de déchiffrement, la valeur du paramètre `--encryption-context` peut inclure des paires `name=value`, des éléments `name` (sans valeur) ou une combinaison des deux.

```
--encryption-context name[=value] [name] [name=value] ...
```

Si `name` ou `value` dans une paire `name=value` contient des espaces ou des caractères spéciaux, placez la totalité de la paire entre guillemets.

```
--encryption-context "department=software engineering" "Région AWS=us-west-2"
```

Par exemple, cette commande de chiffrement inclut un contexte de chiffrement avec deux paires, `purpose=test` et `dept=23`.

```
aws-encryption-cli --encrypt --encryption-context purpose=test dept=23 ...
```

Ces commandes de déchiffrement réussiraient. Le contexte de chiffrement de chaque commande est un sous-ensemble du contexte de chiffrement d'origine.

```
\\ Any one or both of the encryption context pairs
aws-encryption-cli --decrypt --encryption-context dept=23 ...

\\ Any one or both of the encryption context names
aws-encryption-cli --decrypt --encryption-context purpose ...

\\ Any combination of names and pairs
aws-encryption-cli --decrypt --encryption-context dept purpose=test ...
```

Cependant, ces commandes de déchiffrement échoueraient. Le contexte de chiffrement du message chiffré ne contient pas les éléments spécifiés.

```
aws-encryption-cli --decrypt --encryption-context dept=Finance ...
aws-encryption-cli --decrypt --encryption-context scope ...
```

Comment définir une politique d'engagement

Pour définir la politique d'engagement pour la commande, utilisez le `--commitment-policy`paramètre. Ce paramètre est introduit dans la version 1.8. x. Il est valide dans les commandes de chiffrement et de déchiffrement. La politique d'engagement que vous définissez n'est valide que pour la commande dans laquelle elle apparaît. Si vous ne définissez pas de politique d'engagement pour une commande, l'interface de ligne de commande de AWS chiffrement utilise la valeur par défaut.

Par exemple, la valeur de paramètre suivante définit la politique d'engagement sur`require-encrypt-allow-decrypt`, qui chiffre toujours avec un engagement de clé, mais qui déchiffre un texte chiffré avec ou sans validation de clé.

```
--commitment-policy require-encrypt-allow-decrypt
```

Procédure pour stocker les paramètres dans un fichier de configuration

Vous pouvez gagner du temps et éviter les erreurs de frappe en enregistrant les paramètres et les valeurs de la CLI de AWS chiffrement fréquemment utilisés dans des fichiers de configuration.

Un fichier de configuration est un fichier texte qui contient les paramètres et les valeurs d'une commande de AWS chiffrement CLI. Lorsque vous faites référence à un fichier de configuration dans une commande de l'interface de ligne de commande de AWS chiffrement, la référence est remplacée par les paramètres et les valeurs du fichier de configuration. L'effet est le même que si vous aviez saisi le contenu d'un fichier dans la ligne de commande. Un fichier de configuration peut avoir n'importe quel nom et peut être situé dans n'importe quel répertoire auquel l'utilisateur actuel peut accéder.

L'exemple de fichier de configuration suivant `key.conf` en spécifie deux AWS KMS keys dans des régions différentes.

```
--wrapping-keys key=arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
--wrapping-keys key=arn:aws:kms:us-
east-2:111122223333:key/0987ab65-43cd-21ef-09ab-87654321cdef
```

Pour utiliser le fichier de configuration dans une commande, ajoutez au nom du fichier le signe arobase, `@`, en préfixe. Dans une PowerShell console, utilisez un caractère de coche arrière pour échapper au signe arobase (`` `@ ``).

Cet exemple de commande utilise le fichier `key.conf` dans une commande de chiffrement.

Bash

```
$ aws-encryption-cli -e @key.conf -i hello.txt -o testdir
```

PowerShell

```
PS C:\> aws-encryption-cli -e `@key.conf -i .\Hello.txt -o .\TestDir
```

Règles des fichiers de configuration

Les règles relatives à l'utilisation des fichiers de configuration sont les suivantes :

- Vous pouvez inclure plusieurs paramètres dans chaque fichier de configuration et les répertorier dans n'importe quel ordre. Répertoriez chaque paramètre avec ses valeurs (le cas échéant) sur une ligne distincte.
- Utilisez # pour ajouter un commentaire à toute ou partie d'une ligne.
- Vous pouvez inclure des références à d'autres fichiers de configuration. N'utilisez pas de marche arrière pour échapper au @ panneau, même à PowerShell l'intérieur.
- Si vous utilisez des guillemets dans un fichier de configuration, le texte entre guillemets ne peut pas s'étendre sur plusieurs lignes.

Par exemple, voici le contenu d'un exemple de fichier `encrypt.conf`.

```
# Archive Files
--encrypt
--output /archive/logs
--recursive
--interactive
--encryption-context class=unclassified dept=IT
--suffix  # No suffix
--metadata-output ~/metadata
@caching.conf  # Use limited caching
```

Vous pouvez également inclure plusieurs fichiers de configuration dans une commande. Cet exemple de commande utilise à la fois les fichiers de configuration `encrypt.conf` et `master-keys.conf`.

Bash

```
$  aws-encryption-cli -i /usr/logs @encrypt.conf @master-keys.conf
```

PowerShell

```
PS C:\> aws-encryption-cli -i $home\Test\*.log `@encrypt.conf `@master-keys.conf
```

Suivant : Essayez les exemples d'interface de ligne de commande de AWS chiffrement

Exemples deAWSCLI de chiffrement

Utilisez les exemples suivants pour tester leAWSL'interface de chiffrement sur la plateforme de votre choix. Pour obtenir de l'aide concernant les clés principales et d'autres paramètres, consultez Comment utiliser l'interface de ligne de commande AWS de chiffrement. Pour accéder à une référence rapide, consultez Référence des paramètres et de la syntaxe de l'interface de ligne de commande du kit AWS Encryption SDK.

> **Note**
>
> Les exemples suivants utilisent la syntaxe deAWSVersion 2.1 de ligne de commande de chiffrement.h/24, j/7.
> Les nouvelles fonctionnalités de sécurité ont été initialement publiées dansAWSVersion de ligne de commande de chiffrement 1.7.h/24, j/7et 2.0.h/24, j/7. Toutefois,AWSVersion de ligne de commande de chiffrement version 1.8.h/24, j/7remplace la version 1.7.h/24, j/7etAWSChiffrement CLI 2.1.h/24, j/7remplace 2,0.h/24, j/7. Pour en savoir plus, consultez leconseil de sécuritédans leaws-encryption-sdk-clirepository surGitHub.

Pour obtenir un exemple montrant comment utiliser la fonctionnalité de sécurité qui limite les clés de données chiffrées, voirLimiter les clés de données chiffrées.

Pour un exemple montrant comment utiliserAWS KMSClés multi-régions, voirUtilisation de plusieurs régions AWS KMS keys.

Rubriques

- Chiffrement d'un fichier

- Déchiffrement d'un fichier
- Chiffrement de tous les fichiers d'un répertoire
- Déchiffrement de tous les fichiers d'un répertoire
- Chiffrement et déchiffrement sur la ligne de commande
- Utilisation de plusieurs clés principales
- Chiffrement et déchiffrement dans les scripts
- Utilisation de la mise en cache des clés de données

Chiffrement d'un fichier

Cet exemple utilise leAWSCryptage CLI pour chiffrer le contenu de l'`hello.txt`, qui contient une chaîne « Hello World ».

Lorsque vous exécutez une commande de chiffrement sur un fichier, leAWSL'interface de chiffrement récupère le contenu du fichier, génère unclé de données, chiffre le contenu du fichier sous la clé de données, puis écrit lemessage chiffrévers un nouveau fichier.

La première commande enregistre l'ARN clé d'unAWS KMS keydans le`$keyArn`Variable. Lors du chiffrement avec unAWS KMS key, vous pouvez l'identifier à l'aide d'un ID de clé, d'un ARN de clé, d'un nom d'alias ou d'un ARN d'alias. Pour plus de détails sur les identificateurs de clé d'unAWS KMS key, voirIdentificateurs clésdans leAWS Key Management ServiceManuel du développeur.

La seconde commande chiffre le contenu du fichier. La commande utilise le paramètre `--encrypt` pour spécifier l'opération et le paramètre `--input` pour indiquer le fichier à chiffrer. Le paramètre `--wrapping-keys` et son attribut key obligatoire indiquent à la commande d'utiliser la AWS KMS key représentée par l'ARN de la clé.

La commande utilise le paramètre `--metadata-output` pour spécifier un fichier texte pour les métadonnées à propos de l'opération de chiffrement. En tant que bonne pratique, la commande utilise le paramètre `--encryption-context` pour spécifier un contexte de chiffrement.

Cette commande utilise également la commande`--commitment-policy`paramètrepour définir explicitement la politique d'engagement. Dans la version 1.8.h/24, j/7, ce paramètre est obligatoire lorsque vous utilisez l'`--wrapping-keys` Paramètre . À partir de la version 2.1.h/24, j/7, le`--commitment-policy`Le paramètre est facultatif, mais recommandé.

La valeur du paramètre `--output`, un point (.), indique à la commande d'écrire le fichier de sortie dans le répertoire actuel.

Bash

```
\\ To run this example, replace the fictitious key ARN with a valid value.
$ keyArn=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

$ aws-encryption-cli --encrypt \
                     --input hello.txt \
                     --wrapping-keys key=$keyArn \
                     --metadata-output ~/metadata \
                     --encryption-context purpose=test \
                     --commitment-policy require-encrypt-require-decrypt \
                     --output .
```

PowerShell

```
# To run this example, replace the fictitious key ARN with a valid value.
PS C:\> $keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

PS C:\> aws-encryption-cli --encrypt `
                           --input Hello.txt `
                           --wrapping-keys key=$keyArn `
                           --metadata-output $home\Metadata.txt `
                           --commitment-policy require-encrypt-require-decrypt `
                           --encryption-context purpose=test `
                           --output .
```

Lorsque la commande de chiffrement aboutit, elle ne renvoie pas de sortie. Pour déterminer si la commande a abouti, vérifiez la valeur booléenne dans la variable `$?`. Lorsque la commande aboutit, la valeur de`$?`est`0`(Bash) ou`True`(PowerShell). Lorsque la commande échoue, la valeur de`$?`est différent de zéro (Bash) ou`False`(PowerShell).

Bash

```
$ echo $?
0
```

PowerShell

```
PS C:\> $?
```

```
True
```

Vous pouvez également utiliser une commande de liste de répertoires pour vérifier que la commande de chiffrement a créé un nouveau fichier `hello.txt.encrypted`. Étant donné que la commande de chiffrement n'a pas spécifié de nom de fichier pour la sortie, l'AWSL'interface de chiffrement a écrit la sortie dans un fichier nommé de la même façon que le fichier d'entrée, auquel un `.encrypted`suffixe. Pour utiliser un autre suffixe ou supprimer le suffixe, utilisez le paramètre `--suffix`.

Le fichier `hello.txt.encrypted` contient un message chiffré qui comprend le texte chiffré du fichier `hello.txt`, une copie chiffrée de la clé de données et des métadonnées supplémentaires, y compris le contexte de chiffrement.

Bash

```
$  ls
hello.txt  hello.txt.encrypted
```

PowerShell

```
PS C:\> dir

    Directory: C:\TestCLI

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        9/15/2017   5:57 PM             11 Hello.txt
-a----        9/17/2017   1:06 PM            585 Hello.txt.encrypted
```

Déchiffrement d'un fichier

Cet exemple utilise leAWSEncryption CLI pour déchiffrer le contenu du`Hello.txt.encrypted`fichier qui a été chiffré dans l'exemple précédent.

La commande de déchiffrement utilise le paramètre `--decrypt` pour indiquer l'opération et le paramètre `--input` pour identifier le fichier à déchiffrer. La valeur du paramètre `--output` est un point qui représente le répertoire actuel.

Le`--wrapping-keys`Paramètre avec uncléspécifie la clé d'encapsulation utilisée pour déchiffrer le message chiffré. Dans les commandes de déchiffrement avecAWS KMS keys, la valeur de l'attribut clé doit être une valeurARN de clé. Le`--wrapping-keys`est obligatoire dans une commande de déchiffrement. Si vous utilisezAWS KMS keys, vous pouvez utiliser lecléAttribut à spécifierAWS KMS keyspour le déchiffrement ou ledécouverteavec une valeur de`true`(mais pas les deux). Si vous utilisez un fournisseur de clés principales personnalisé, l'cléetfournisseurLes attributs sont obligatoires.

Le`--commitment-policy`paramètreest facultatif à partir de la version 2.1.h/24, j/7, mais il est recommandé. Son utilisation rend explicitement votre intention claire, même si vous spécifiez la valeur par défaut,`require-encrypt-require-decrypt`.

Le paramètre `--encryption-context` est facultatif dans la commande de déchiffrement, même lorsqu'un contexte de chiffrement est fourni dans la commande de chiffrement. Dans ce cas, la commande de déchiffrement utilise le même contexte de chiffrement que celui fourni dans la commande de chiffrement. Avant de déchiffrer, leAWSL'interface de ligne de commande de chiffrement vérifie que le contexte de chiffrement du message chiffré comprend une`purpose=test`paire. Si ce n'est pas le cas, la commande de déchiffrement échoue.

Le paramètre `--metadata-output` spécifie un fichier pour les métadonnées relatives à l'opération de déchiffrement. La valeur du paramètre `--output`, un point (.), écrit le fichier de sortie dans le répertoire actuel.

À titre de bonne pratique, utilisez le`--max-encrypted-data-keys`pour éviter de déchiffrer un message mal formé avec un nombre excessif de clés de données chiffrées. Spécifiez le nombre attendu de clés de données chiffrées (une pour chaque clé d'encapsulation utilisée dans le chiffrement) ou un maximum raisonnable (par exemple 5). Pour plus d'informations, consultez Limiter les clés de données chiffrées.

Le`--buffer`renvoie le texte brut uniquement après le traitement de toutes les entrées, y compris la vérification de la signature numérique si elle est présente.

Bash

```
\\ To run this example, replace the fictitious key ARN with a valid value.
$ keyArn=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

$ aws-encryption-cli --decrypt \
                     --input hello.txt.encrypted \
                     --wrapping-keys key=$keyArn \
```

```
                    --commitment-policy require-encrypt-require-decrypt \
                    --encryption-context purpose=test \
                    --metadata-output ~/metadata \
                    --max-encrypted-data-keys 1 \
                    --buffer \
                    --output .
```

PowerShell

```
\\ To run this example, replace the fictitious key ARN with a valid value.
PS C:\> $keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

PS C:\> aws-encryption-cli --decrypt `
                           --input Hello.txt.encrypted `
                           --wrapping-keys key=$keyArn `
                           --commitment-policy require-encrypt-require-decrypt `
                           --encryption-context purpose=test `
                           --metadata-output $home\Metadata.txt `
                           --max-encrypted-data-keys 1 `
                           --buffer `
                           --output .
```

Lorsqu'une commande de déchiffrement aboutit, elle ne renvoie pas de sortie. Pour déterminer si la commande a abouti, récupérez la valeur de la variable `$?`. Vous pouvez également utiliser une commande de liste de répertoires pour vérifier que la commande a créé un nouveau fichier avec un suffixe `.decrypted`. Pour voir le contenu en texte brut, utilisez une commande pour récupérer le contenu d'un fichier, par exemple `cat` ou Get-Content.

Bash

```
$  ls
hello.txt  hello.txt.encrypted  hello.txt.encrypted.decrypted

$  cat hello.txt.encrypted.decrypted
Hello World
```

PowerShell

```
PS C:\> dir
```

```
    Directory: C:\TestCLI

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        9/17/2017   1:01 PM             11 Hello.txt
-a----        9/17/2017   1:06 PM            585 Hello.txt.encrypted
-a----        9/17/2017   1:08 PM             11 Hello.txt.encrypted.decrypted


PS C:\> Get-Content Hello.txt.encrypted.decrypted
Hello World
```

Chiffrement de tous les fichiers d'un répertoire

Cet exemple utilise leAWSEncryption CLI pour chiffrer le contenu de tous les fichiers d'un répertoire.

Lorsqu'une commande affecte plusieurs fichiers, l'AWSL'interface de ligne de commande de chiffrement traite chaque fichier individuellement. Elle récupère le contenu du fichier, récupère une clé de données unique pour le fichier à partir de la clé principale, chiffre le contenu du fichier sous la clé de données et écrit les résultats dans un nouveau fichier dans le répertoire de sortie. Par conséquent, vous pouvez déchiffrer les fichiers de sortie de manière indépendante.

Cette liste du répertoire `TestDir` affiche les fichiers en texte brut que nous souhaitons chiffrer.

Bash

```
$  ls testdir
cool-new-thing.py  hello.txt  employees.csv
```

PowerShell

```
PS C:\> dir C:\TestDir

    Directory: C:\TestDir

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        9/12/2017   3:11 PM           2139 cool-new-thing.py
-a----        9/15/2017   5:57 PM             11 Hello.txt
-a----        9/17/2017   1:44 PM             46 Employees.csv
```

La première commande enregistre l'optionAmazon Resource Name (ARN)d'unAWS KMS keydans le`$keyArn`Variable.

La deuxième commande chiffre le contenu des fichiers dans le répertoire `TestDir` et écrit les fichiers du contenu chiffré dans le répertoire `TestEnc`. Si le répertoire `TestEnc` n'existe pas, la commande échoue. Étant donné que l'emplacement d'entrée est un répertoire, le paramètre `--recursive` est obligatoire.

Le`--wrapping-keys`paramètre, et ce qui est requisclé, spécifiez la clé d'encapsulation à utiliser. La commande de chiffrement inclut un contexte de chiffrement, `dept=IT`. Lorsque vous spécifiez un contexte de chiffrement dans une commande qui chiffre plusieurs fichiers, le même contexte de chiffrement est utilisée pour tous les fichiers.

La commande dispose également d'un`--metadata-output`pour indiquer le paramètreAWSEncryption CLI où écrire les métadonnées relatives aux opérations de chiffrement. LeAWSL'interface de ligne de chiffrement écrit un enregistrement de métadonnées pour chaque fichier qui a été chiffré.

Le`--commitment-policy parameter`est facultative à partir de la version 2.1.h/24, j/7, mais il est recommandé. Si la commande ou le script échoue parce qu'il ne peut pas déchiffrer un texte chiffré, le paramètre de stratégie d'engagement explicite peut vous aider à détecter rapidement le problème.

Lorsque la commande aboutit, leAWSEncryption CLI écrit les fichiers chiffrés dans le`TestEnc`, mais elle ne renvoie aucune sortie.

La dernière commande répertorie les fichiers dans le répertoire `TestEnc`. Il existe un fichier de sortie du contenu chiffré pour chaque fichier d'entrée de contenu en texte brut. Étant donné que la commande n'a pas spécifié d'autre suffixe, la commande de chiffrement a ajouté `.encrypted` à chacun des noms des fichiers d'entrée.

Bash

```
# To run this example, replace the fictitious key ARN with a valid master key
 identifier.
$  keyArn=arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

$ aws-encryption-cli --encrypt \
                     --input testdir --recursive\
                     --wrapping-keys key=$keyArn \
                     --encryption-context dept=IT \
```

```
                    --commitment-policy require-encrypt-require-decrypt \
                    --metadata-output ~/metadata \
                    --output testenc

$ ls testenc
cool-new-thing.py.encrypted  employees.csv.encrypted  hello.txt.encrypted
```

PowerShell

```
# To run this example, replace the fictitious key ARN with a valid master key
 identifier.
PS C:\> $keyArn = arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

PS C:\> aws-encryption-cli --encrypt `
                           --input .\TestDir --recursive `
                           --wrapping-keys key=$keyArn `
                           --encryption-context dept=IT `
                           --commitment-policy require-encrypt-require-decrypt `
                           --metadata-output .\Metadata\Metadata.txt `
                           --output .\TestEnc

PS C:\> dir .\TestEnc

    Directory: C:\TestEnc

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        9/17/2017   2:32 PM           2713 cool-new-thing.py.encrypted
-a----        9/17/2017   2:32 PM            620 Hello.txt.encrypted
-a----        9/17/2017   2:32 PM            585 Employees.csv.encrypted
```

Déchiffrement de tous les fichiers d'un répertoire

Cet exemple déchiffre tous les fichiers d'un répertoire. Il commence par les fichiers du répertoire `TestEnc` qui ont été chiffrés dans l'exemple précédent.

Bash

```
$  ls testenc
cool-new-thing.py.encrypted  hello.txt.encrypted  employees.csv.encrypted
```

PowerShell

```
PS C:\> dir C:\TestEnc

    Directory: C:\TestEnc

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        9/17/2017   2:32 PM           2713 cool-new-thing.py.encrypted
-a----        9/17/2017   2:32 PM            620 Hello.txt.encrypted
-a----        9/17/2017   2:32 PM            585 Employees.csv.encrypted
```

Cette commande de déchiffrement déchiffre tous les fichiers duTestEncet écrit les fichiers en texte brut dans le répertoireTestDecrépertoire. Le`--wrapping-keys`Paramètre avec uncléAttribut etARN de cléindique la valeurAWSCLI de chiffrement quiAWS KMS keysà utiliser pour déchiffrer les fichiers. La commande utilise le`--interactive`pour indiquer le paramètreAWSEncryption CLI pour vous adresser une invite avant de remplacer un fichier nommé.

Cette commande utilise également le contexte de chiffrement qui a été fourni lorsque les fichiers ont été chiffrés. Lors du déchiffrement de plusieurs fichiers, l'AWSL'interface de ligne de chiffrement vérifie le contexte de chiffrement de chaque fichier. Si la vérification du contexte de chiffrement d'un fichier échoue, leAWSL'interface de chiffrement rejette le fichier, écrit un avertissement, enregistre l'échec dans les métadonnées, puis continue à vérifier les fichiers restants. Si l'icôneAWSL'interface de chiffrement ne parvient pas à déchiffrer un fichier pour toute autre raison, l'ensemble de la commande de déchiffrement échoue immédiatement.

Dans cet exemple, les messages chiffrés de tous les fichiers d'entrée contiennent l'élément du contexte de chiffrement `dept=IT`. Toutefois, si vous procédiez au déchiffrement de messages ayant des contextes de chiffrement différents, vous devriez toujours être en mesure de vérifier une partie du contexte de chiffrement. Par exemple, si certains messages avaient un contexte de chiffrement `dept=finance` et d'autres messages `dept=IT`, vous pourriez vérifier que le contexte de chiffrement contient toujours un nom `dept` sans spécifier la valeur. Pour plus de précisions, vous pourriez déchiffrer les fichiers dans des commandes distinctes.

La commande de déchiffrement ne renvoie pas de sortie, mais vous pouvez utiliser une commande de liste de répertoires pour vérifier qu'elle a crée de nouveaux fichiers avec le suffixe `.decrypted`. Pour voir le contenu en texte brut, utilisez une commande pour récupérer le contenu d'un fichier.

Bash

```
# To run this example, replace the fictitious key ARN with a valid master key
 identifier.
$ keyArn=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

$ aws-encryption-cli --decrypt \
                     --input testenc --recursive \
                     --wrapping-keys key=$keyArn \
                     --encryption-context dept=IT \
                     --commitment-policy require-encrypt-require-decrypt \
                     --metadata-output ~/metadata \
                     --max-encrypted-data-keys 1 \
                     --buffer \
                     --output testdec --interactive

$ ls testdec
cool-new-thing.py.encrypted.decrypted  hello.txt.encrypted.decrypted
 employees.csv.encrypted.decrypted
```

PowerShell

```
# To run this example, replace the fictitious key ARN with a valid master key
 identifier.
PS C:\> $keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

PS C:\> aws-encryption-cli --decrypt `
                           --input C:\TestEnc --recursive `
                           --wrapping-keys key=$keyArn `
                           --encryption-context dept=IT `
                           --commitment-policy require-encrypt-require-decrypt `
                           --metadata-output $home\Metadata.txt `
                           --max-encrypted-data-keys 1 `
                           --buffer `
                           --output C:\TestDec --interactive

PS C:\> dir .\TestDec


    Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
```

```
-a----        10/8/2017   4:57 PM           2139 cool-new-
thing.py.encrypted.decrypted
-a----        10/8/2017   4:57 PM             46 Employees.csv.encrypted.decrypted
-a----        10/8/2017   4:57 PM             11 Hello.txt.encrypted.decrypted
```

Chiffrement et déchiffrement sur la ligne de commande

Ces exemples vous montrent comment diriger l'entrée vers des commandes (stdin) et écrire la sortie dans la ligne de commande (stdout). Ils expliquent comment représenter stdin et stdout dans une commande et comment utiliser les outils de codage en Base64 intégrés pour empêcher que le shell interprète mal les caractères non-ASCII.

Cet exemple dirige une chaîne en texte brut vers une commande de chiffrement et enregistre le message chiffré dans une variable. Ensuite, il dirige le message chiffré de la variable vers une commande de déchiffrement, qui écrit sa sortie dans le pipeline (stdout).

L'exemple se compose de trois commandes :

- La première commande enregistre l'optionARN de cléd'unAWS KMS keydans le`$keyArn`Variable.

  Bash

  ```
  $  keyArn=arn:aws:kms:us-
  west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
  ```

  PowerShell

  ```
  PS C:\> $keyArn = 'arn:aws:kms:us-
  west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
  ```

- La deuxième commande dirige la chaîne `Hello World` vers la commande de chiffrement et enregistre le résultat dans la variable `$encrypted`.

  Le`--input`et`--output`Les paramètres sont obligatoires dans tous lesAWSCommandes de commande de chiffrement. Pour indiquer que l'entrée est dirigée vers la commande (stdin), utilisez un trait d'union (`-`) pour la valeur du paramètre `--input`. Pour envoyer la sortie vers la ligne de commande (stdout), utilisez un tiret pour la valeur du paramètre `--output`.

Le paramètre `--encode` code en base64 la sortie avant de la renvoyer. Cela empêche que le shell interprète mal les caractères non-ASCII dans le message chiffré.

Étant donné que cette commande est une simple preuve de concept, nous avons omis le contexte de chiffrement et supprimé les métadonnées (`-S`).

Bash

```
$ encrypted=$(echo 'Hello World' | aws-encryption-cli --encrypt -S \
                                                      --input - --output - --
encode \
                                                      --wrapping-keys key=
$keyArn )
```

PowerShell

```
PS C:\> $encrypted = 'Hello World' | aws-encryption-cli --encrypt -S `
                                                        --input - --output - --
encode `
                                                        --wrapping-keys key=
$keyArn
```

- La troisième commande dirige le message chiffré de la variable `$encrypted` vers la commande de déchiffrement.

  Cette commande de déchiffrement utilise `--input -` pour indiquer que l'entrée provient du pipeline (stdin) et `--output -` pour envoyer la sortie vers le pipeline (stdout). (Le paramètre d'entrée prend l'emplacement de l'entrée, et non les octets d'entrée réels. Vous ne pouvez donc pas utiliser la variable `$encrypted` en tant que valeur du paramètre `--input`.)

  Cet exemple utilise ledécouverteAttribut du`--wrapping-keys`pour autoriser le paramètreAWSCLI de chiffrement pour utiliser n'importe quelAWS KMS keypour déchiffrer les données. Il ne spécifie pas depolitique d'engagement, il utilise donc la valeur par défaut de la version 2.1.h/24, j/7et plus tard,`require-encrypt-require-decrypt`.

  Étant donné que la sortie a été chiffrée, puis codée, la commande de déchiffrement utilise le paramètre `--decode` pour décoder l'entrée codée en Base64 avant de la déchiffrer. Vous pouvez

également utiliser le paramètre `--decode` pour décoder l'entrée codée en Base64 avant de la chiffrer.

Cette fois encore, la commande omet le contexte de chiffrement et supprime les métadonnées (-S).

Bash

```
$  echo $encrypted | aws-encryption-cli --decrypt --wrapping-keys discovery=true
 --input - --output - --decode --buffer -S
Hello World
```

PowerShell

```
PS C:\> $encrypted | aws-encryption-cli --decrypt --wrapping-keys discovery=$true
 --input - --output - --decode --buffer -S
Hello World
```

Vous pouvez également effectuer les opérations de chiffrement et de déchiffrement dans une même commande sans la variable intermédiaire.

Comme dans l'exemple précédent, les paramètres `--input` et `--output` ont une valeur `-` et la commande utilise le paramètre `--encode` pour coder la sortie et le paramètre `--decode` pour décoder l'entrée.

Bash

```
$  keyArn=arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

$  echo 'Hello World' |
          aws-encryption-cli --encrypt --wrapping-keys key=$keyArn --input - --
output - --encode -S |
          aws-encryption-cli --decrypt --wrapping-keys discovery=true --input - --
output - --decode -S
Hello World
```

PowerShell

```
PS C:\> $keyArn = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
```

```
PS C:\> 'Hello World' |
               aws-encryption-cli --encrypt --wrapping-keys key=$keyArn --input - --
output - --encode -S |
               aws-encryption-cli --decrypt --wrapping-keys discovery=$true --input
 - --output - --decode -S
Hello World
```

Utilisation de plusieurs clés principales

Cet exemple montre comment utiliser plusieurs clés principales lors du chiffrement et du déchiffrement de données dans leAWSCLI de chiffrement.

Lorsque vous utilisez plusieurs clés principales pour chiffrer des données, n'importe quelle clé principale peut être utilisée pour déchiffrer les données. Cette stratégie garantit que vous pouvez déchiffrer les données même si l'une des clés principales n'est pas disponible. Si vous stockez les données chiffrées dans plusieursRégions AWS, cette stratégie vous permet d'utiliser une clé principale dans la même région pour déchiffrer les données.

Lorsque vous procédez à un chiffrement avec plusieurs clés principales, la première clé principale joue un rôle spécifique. Elle génère la clé de données qui est utilisée pour chiffrer les données. Les autres clés principales chiffrent la clé de données en texte brut. Le message chiffré inclut les données chiffrées et un ensemble de clés de données chiffrées, une pour chaque clé principale. Même si la première clé principale a généré la clé de données, toutes les clés principales peuvent déchiffrer les clés de données, qui peuvent être utilisées pour déchiffrer les données.

Chiffrement avec trois clés principales

Cet exemple de commande utilise trois clés d'encapsulation pour chiffrer le`Finance.log`, un dossier sur chacun des troisRégions AWS.

Elle écrit ensuite le message chiffré dans le répertoire `Archive`. La commande utilise le paramètre `--suffix` sans valeur pour supprimer le suffixe. Par conséquent, les noms des fichiers d'entrée et de sortie seront les mêmes.

La commande utilise le paramètre `--wrapping-keys` avec trois attributs key. Vous pouvez également utiliser plusieurs paramètres `--wrapping-keys` dans la même commande.

Pour chiffrer le fichier journal, leAWSLa CLI de chiffrement demande la première clé d'encapsulation de la liste,`$key1`, pour générer la clé de données qu'elle utilise pour chiffrer les données. Ensuite,

elle utilise chacune des autres clés d'encapsulation pour chiffrer une copie en texte brut de la même clé de données. Le message chiffré dans le fichier de sortie inclut les trois des clés de données chiffrées.

Bash

```
$ key1=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
$ key2=arn:aws:kms:us-east-2:111122223333:key/0987ab65-43cd-21ef-09ab-87654321cdef
$ key3=arn:aws:kms:ap-
southeast-1:111122223333:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d

$ aws-encryption-cli --encrypt --input /logs/finance.log \
                               --output /archive --suffix \
                               --encryption-context class=log \
                               --metadata-output ~/metadata \
                               --wrapping-keys key=$key1 key=$key2 key=$key3
```

PowerShell

```
PS C:\> $key1 = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
PS C:\> $key2 = 'arn:aws:kms:us-
east-2:111122223333:key/0987ab65-43cd-21ef-09ab-87654321cdef'
PS C:\> $key3 = 'arn:aws:kms:ap-
southeast-1:111122223333:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d'

PS C:\> aws-encryption-cli --encrypt --input D:\Logs\Finance.log `
                           --output D:\Archive --suffix `
                           --encryption-context class=log `
                           --metadata-output $home\Metadata.txt `
                           --wrapping-keys key=$key1 key=$key2 key=$key3
```

Cette commande déchiffre la copie chiffrée du fichier `Finance.log` et l'écrit dans un fichier `Finance.log.clear` du répertoire `Finance`. Pour déchiffrer des données chiffrées sous troisAWS KMS keys, vous pouvez spécifier les trois mêmesAWS KMS keysou n'importe quel sous-ensemble d'entre eux. Cet exemple spécifie uniquement l'une des options suivantes :AWS KMS keys.

Pour dire à laAWSCLI de chiffrement quiAWS KMS keyspour déchiffrer vos données, utilisez lecléAttribut du`--wrapping-keys` Paramètre . Lors du déchiffrement avecAWS KMS keys, la valeur duclédoit être un attributARN de clé.

Vous devez être autorisé à appeler leAPI de déchiffrementsur leAWS KMS keysque vous spécifiez. Pour de plus amples informations, veuillez consulterAuthentification et contrôle d'accès pourAWS KMS.

À titre de bonne pratique, cet exemple utilise le`--max-encrypted-data-keys`pour éviter de déchiffrer un message mal formé avec un nombre excessif de clés de données chiffrées. Même si cet exemple utilise une seule clé d'encapsulation pour le déchiffrement, le message chiffré comporte trois (3) clés de données chiffrées, une pour chacune des trois clés d'encapsulation utilisées lors du chiffrement. Spécifiez le nombre attendu de clés de données chiffrées ou une valeur maximale raisonnable, par exemple 5. Si vous spécifiez une valeur maximale inférieure à 3, la commande échoue. Pour plus d'informations, consultez Limiter les clés de données chiffrées.

Bash

```
$ aws-encryption-cli --decrypt --input /archive/finance.log \
                     --wrapping-keys key=$key1 \
                     --output /finance --suffix '.clear' \
                     --metadata-output ~/metadata \
                     --max-encrypted-data-keys 3 \
                     --buffer \
                     --encryption-context class=log
```

PowerShell

```
PS C:\> aws-encryption-cli --decrypt `
                           --input D:\Archive\Finance.log `
                           --wrapping-keys key=$key1 `
                           --output D:\Finance --suffix '.clear' `
                           --metadata-output .\Metadata\Metadata.txt `
                           --max-encrypted-data-keys 3 `
                           --buffer `
                           --encryption-context class=log
```

Chiffrement et déchiffrement dans les scripts

Cet exemple vous montre comment utiliser l'AWSCLI de chiffrement dans les scripts. Vous pouvez écrire des scripts qui chiffrent et déchiffrent simplement les données, ou des scripts qui effectuent des chiffrements ou des déchiffrements dans le cadre d'un processus de gestion des données.

Dans cet exemple, le script récupère une collection de fichiers journaux, les compresse, chiffre, puis copie les fichiers chiffrés dans un compartiment Amazon S3. Ce script traite chaque fichier séparément. Vous pouvez ainsi les déchiffrer et les développer de manière indépendante.

Lorsque vous compressez et chiffrez les fichiers, assurez-vous d'effectuer la compression avant le chiffrement. Les données correctement chiffrées ne sont pas compressibles.

> **Warning**
>
> Soyez prudent lorsque vous compressez des données qui incluent à la fois des secrets et des données pouvant être contrôlées par un utilisateur malveillant. La taille finale des données compressées peut révéler par inadvertance des informations sensibles sur son contenu.

Bash

```
# Continue running even if an operation fails.
set +e

dir=$1
encryptionContext=$2
s3bucket=$3
s3folder=$4
masterKeyProvider="aws-kms"
metadataOutput="/tmp/metadata-$(date +%s)"

compress(){
    gzip -qf $1
}

encrypt(){
    # -e encrypt
    # -i input
    # -o output
    # --metadata-output unique file for metadata
    # -m masterKey read from environment variable
    # -c encryption context read from the second argument.
    # -v be verbose
    aws-encryption-cli -e -i ${1} -o $(dirname ${1}) --metadata-output
 ${metadataOutput} -m key="${masterKey}" provider="${masterKeyProvider}" -c
 "${encryptionContext}" -v
}
```

```
s3put (){
    # copy file argument 1 to s3 location passed into the script.
    aws s3 cp ${1} ${s3bucket}/${s3folder}
}

# Validate all required arguments are present.
if [ "${dir}" ] && [ "${encryptionContext}" ] && [ "${s3bucket}" ] &&
 [ "${s3folder}" ] && [ "${masterKey}" ]; then

# Is $dir a valid directory?
test -d "${dir}"
if [ $? -ne 0 ]; then
    echo "Input is not a directory; exiting"
    exit 1
fi

# Iterate over all the files in the directory, except *gz and *encrypted (in case of
 a re-run).
for f in $(find ${dir} -type f \( -name "*" ! -name \*.gz ! -name \*encrypted \) );
 do
    echo "Working on $f"
    compress ${f}
    encrypt ${f}.gz
    rm -f ${f}.gz
    s3put ${f}.gz.encrypted
done;
else
    echo "Arguments: <Directory> <encryption context> <s3://bucketname> <s3 folder>"
    echo " and ENV var \$masterKey must be set"
    exit 255
fi
```

PowerShell

```
#Requires -Modules AWSPowerShell, Microsoft.PowerShell.Archive
Param
(
    [Parameter(Mandatory)]
    [ValidateScript({Test-Path $_})]
    [String[]]
    $FilePath,
```

```
    [Parameter()]
    [Switch]
    $Recurse,

    [Parameter(Mandatory=$true)]
    [String]
    $wrappingKeyID,

    [Parameter()]
    [String]
    $masterKeyProvider = 'aws-kms',

    [Parameter(Mandatory)]
    [ValidateScript({Test-Path $_})]
    [String]
    $ZipDirectory,

    [Parameter(Mandatory)]
    [ValidateScript({Test-Path $_})]
    [String]
    $EncryptDirectory,

    [Parameter()]
    [String]
    $EncryptionContext,

    [Parameter(Mandatory)]
    [ValidateScript({Test-Path $_})]
    [String]
    $MetadataDirectory,

    [Parameter(Mandatory)]
    [ValidateScript({Test-S3Bucket -BucketName $_})]
    [String]
    $S3Bucket,

    [Parameter()]
    [String]
    $S3BucketFolder
)

BEGIN {}
PROCESS {
```

```
    if ($files = dir $FilePath -Recurse:$Recurse)
    {

        # Step 1: Compress
        foreach ($file in $files)
        {
            $fileName = $file.Name
            try
            {
                Microsoft.PowerShell.Archive\Compress-Archive -Path $file.FullName -
DestinationPath $ZipDirectory\$filename.zip
            }
            catch
            {
                Write-Error "Zip failed on $file.FullName"
            }

            # Step 2: Encrypt
            if (-not (Test-Path "$ZipDirectory\$filename.zip"))
            {
                Write-Error "Cannot find zipped file: $ZipDirectory\$filename.zip"
            }
            else
            {
                # 2>&1 captures command output
                $err = (aws-encryption-cli -e -i "$ZipDirectory\$filename.zip" `
                                           -o $EncryptDirectory `
                                           -m key=$wrappingKeyID provider=
$masterKeyProvider `
                                           -c $EncryptionContext `
                                           --metadata-output $MetadataDirectory `
                                           -v) 2>&1

                # Check error status
                if ($? -eq $false)
                {
                    # Write the error
                    $err
                }
                elseif (Test-Path "$EncryptDirectory\$fileName.zip.encrypted")
                {
                    # Step 3: Write to S3 bucket
                    if ($S3BucketFolder)
                    {
```

```
                        Write-S3Object -BucketName $S3Bucket -File
 "$EncryptDirectory\$fileName.zip.encrypted" -Key "$S3BucketFolder/
$fileName.zip.encrypted"

                    }
                    else
                    {
                        Write-S3Object -BucketName $S3Bucket -File
 "$EncryptDirectory\$fileName.zip.encrypted"
                    }
                }
            }
        }
    }
}
```

Utilisation de la mise en cache des clés de données

Cet exemple utilise la mise en cache des clés de données dans une commande qui chiffre un grand nombre de fichiers.

Par défaut, leAWSCLI de chiffrement (et autres versions de laAWS Encryption SDK) génère une clé de données unique pour chaque fichier qu'il chiffre. Bien que l'utilisation d'une clé de données unique pour chaque opération est une bonne pratique de chiffrement, la réutilisation limitée des clés de données est acceptable dans certaines situations. Si vous envisagez de mettre en cache des clés de données, contactez un ingénieur sécurité afin de comprendre les exigences de sécurité de votre application et déterminer les seuils de sécurité qui vous conviennent le mieux.

Dans cet exemple, la mise en cache des clés de données accélère l'opération de chiffrement en réduisant la fréquence des demandes auprès du fournisseur de clés principales.

La commande de cet exemple chiffre un grand répertoire avec plusieurs sous-répertoires qui contiennent un total d'environ 800 petits fichiers journaux. La première commande enregistre l'ARN de la AWS KMS key dans une variable `keyARN`. La deuxième commande chiffre tous les fichiers du répertoire d'entrée (de façon récursive) et les écrit dans un répertoire d'archivage. La commande utilise le paramètre `--suffix` pour spécifier le suffixe `.archive`.

Le paramètre `--caching` permet la mise en cache des clés de données. L'attribut capacity, qui limite le nombre de clés de données dans le cache, est défini sur 1, car le traitement de fichiers en

série n'utilise jamais plus d'une clé de données à la fois. L'attribut max_age, qui détermine la durée pendant laquelle la clé de données mise en cache peut être utilisée, est défini sur 10 secondes.

L'attribut max_messages_encrypted facultatif est défini sur 10 messages. Par conséquent, une même clé de données n'est jamais utilisée pour chiffrer plus de 10 fichiers. Le fait de limiter le nombre de fichiers chiffrés par chaque clé de données réduit le nombre de fichiers qui seraient touchés dans le cas peu probable où une clé de données serait compromise.

Pour exécuter cette commande sur les fichiers journaux générés par votre système d'exploitation, vous pouvez avoir besoin d'autorisations d'administrateur (`sudo` dans Linux et Exécuter en tant qu'administrateur dans Windows).

Bash

```
$  keyArn=arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

$  aws-encryption-cli --encrypt \
                      --input /var/log/httpd --recursive \
                      --output ~/archive --suffix .archive \
                      --wrapping-keys key=$keyArn \
                      --encryption-context class=log \
                      --suppress-metadata \
                      --caching capacity=1 max_age=10 max_messages_encrypted=10
```

PowerShell

```
PS C:\> $keyARN = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

PS C:\> aws-encryption-cli --encrypt `
                           --input C:\Windows\Logs --recursive `
                           --output $home\Archive --suffix '.archive' `
                           --wrapping-keys key=$keyARN `
                           --encryption-context class=log `
                           --suppress-metadata `
                           --caching capacity=1 max_age=10
 max_messages_encrypted=10
```

Pour tester l'effet de la mise en cache des clés de données, cet exemple utilise l'Commande mesureApplet de commande dansPowerShell. Lorsque vous exécutez cet exemple sans mise

en cache des clés de données, l'exécution dure environ 25 secondes. Ce processus génère une nouvelle clé de données pour chaque fichier du répertoire.

```
PS C:\> Measure-Command {aws-encryption-cli --encrypt `
                                            --input C:\Windows\Logs --recursive `
                                            --output $home\Archive  --suffix '.archive'
 `
                                            --wrapping-keys key=$keyARN `
                                            --encryption-context class=log `
                                            --suppress-metadata }


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 25
Milliseconds      : 453
Ticks             : 254531202
TotalDays         : 0.000294596298611111
TotalHours        : 0.00707031116666667
TotalMinutes      : 0.42421867
TotalSeconds      : 25.4531202
TotalMilliseconds : 25453.1202
```

La mise en cache des clés de données accélère le processus, même lorsque vous limitez chaque clé de données à un maximum de 10 fichiers. L'exécution de la commande dure désormais moins de 12 secondes et réduit le nombre d'appels au fournisseur de clés principales à 1/10 de la valeur d'origine.

```
PS C:\> Measure-Command {aws-encryption-cli --encrypt `
                                            --input C:\Windows\Logs --recursive `
                                            --output $home\Archive  --suffix '.archive'
 `
                                            --wrapping-keys key=$keyARN `
                                            --encryption-context class=log `
                                            --suppress-metadata `
                                            --caching capacity=1 max_age=10
 max_messages_encrypted=10}


Days              : 0
Hours             : 0
```

```
Minutes           : 0
Seconds           : 11
Milliseconds      : 813
Ticks             : 118132640
TotalDays         : 0.000136727592592593
TotalHours        : 0.00328146222222222
TotalMinutes      : 0.196887733333333
TotalSeconds      : 11.813264
TotalMilliseconds : 11813.264
```

Si vous supprimez la restriction `max_messages_encrypted`, tous les fichiers sont chiffrés sous la même clé de données. Ce changement augmente le risque de réutilisation des clés de données sans réellement accélérer le processus. Cependant, cela réduit le nombre d'appels au fournisseur de clés principales à 1.

```
PS C:\> Measure-Command {aws-encryption-cli --encrypt `
                                            --input C:\Windows\Logs --recursive `
                                            --output $home\Archive  --suffix '.archive'
 `
                                            --wrapping-keys key=$keyARN `
                                            --encryption-context class=log `
                                            --suppress-metadata `
                                            --caching capacity=1 max_age=10}


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 10
Milliseconds      : 252
Ticks             : 102523367
TotalDays         : 0.000118661304398148
TotalHours        : 0.00284787130555556
TotalMinutes      : 0.170872278333333
TotalSeconds      : 10.2523367
TotalMilliseconds : 10252.3367
```

Référence des paramètres et de la syntaxe de l'interface de ligne de commande du kit AWS Encryption SDK

Cette rubrique fournit des diagrammes de syntaxe et de brèves descriptions des paramètres pour vous aider à utiliser l'interface de ligne de commande (CLI) du kit AWS Encryption SDK. Pour obtenir de l'aide sur l'encapsulation des clés et d'autres paramètres, consultezComment utiliser l'interface de ligne de commande AWS de chiffrement. Pour obtenir des exemples, consultez Exemples deAWSCLI de chiffrement. Pour obtenir la documentation complète, consultez Lisez les documents.

Rubriques

- AWSSyntaxe CLI de chiffrement
- AWSParamètres de ligne de commande CLI
- Paramètres avancés

AWSSyntaxe CLI de chiffrement

Ces diagrammes de syntaxe de la CLI deAWS chiffrement présentent la syntaxe de chaque tâche que vous effectuez avec la CLI deAWS chiffrement. Ils représentent la syntaxe recommandée dansAWS Encryption CLI version 2.1. x et versions ultérieures.

De nouvelles fonctionnalités de sécurité ont été initialement publiées dans les versions 1.7 de l'AWSEncryption CLI. x et 2.0. x. Toutefois,AWS Encryption CLI version 1.8. x remplace la version 1.7. x etAWS Encryption CLI 2.1. x remplace 2.0. x. Pour plus de détails, consultez l'avis de sécurité correspondant dans le aws-encryption-sdk-cliréférentiel sur GitHub.

> **Note**
>
> Sauf indication contraire dans la description du paramètre, chaque paramètre ou attribut ne peut être utilisé qu'une seule fois dans chaque commande.
> Si vous utilisez un attribut non pris en charge par un paramètre, la CLI deAWS chiffrement ignore cet attribut non pris en charge sans avertissement ni erreur.

Obtenir de l'aide

Pour obtenir la syntaxe complète de la CLI deAWS chiffrement avec les descriptions des paramètres, utilisez`--help` ou`-h`.

```
aws-encryption-cli (--help | -h)
```

Obtenir la version

Pour obtenir le numéro de version de votre installationAWS Encryption CLI, utilisez`--version`. Assurez-vous d'inclure la version lorsque vous posez des questions, signalez des problèmes ou partagez des conseils sur l'utilisation de la CLI deAWS chiffrement.

```
aws-encryption-cli --version
```

Chiffrer des données

Le diagramme de syntaxe suivant montre les paramètres utilisés par une commande encrypt.

```
aws-encryption-cli --encrypt
                   --input <input> [--recursive] [--decode]
                   --output <output> [--interactive] [--no-overwrite] [--suffix
 [<suffix>]] [--encode]
                   --wrapping-keys  [--wrapping-keys] ...
                       key=<keyID> [key=<keyID>] ...
                       [provider=<provider-name>] [region=<aws-region>]
 [profile=<aws-profile>]
                   --metadata-output <location> [--overwrite-metadata] | --suppress-
metadata]
                   [--commitment-policy <commitment-policy>]
                   [--encryption-context <encryption_context> [<encryption_context>
 ...]]
                   [--max-encrypted-data-keys <integer>]
                   [--algorithm <algorithm_suite>]
                   [--caching <attributes>]
                   [--frame-length <length>]
                   [-v | -vv | -vvv | -vvvv]
                   [--quiet]
```

Déchiffrer des données

Le diagramme de syntaxe suivant montre les paramètres utilisés par une commande decrypt.

Dans la version 1.8. x, le`--wrapping-keys` paramètre est facultatif lors du déchiffrement, mais recommandé. À partir de version 2.1. x, le`--wrapping-keys` paramètre est requis lors du chiffrement et du déchiffrement. EnAWS KMS keys effet, vous pouvez utiliser l'attribut key pour

spécifier des clés d'encapsulation (meilleure pratique) ou définir l'attribut de découverte sur`true`, ce qui ne limite pas le nombre de clés d'encapsulation que la CLI deAWS chiffrement peut utiliser.

```
aws-encryption-cli --decrypt (or [--decrypt-unsigned])
                   --input <input> [--recursive] [--decode]
                   --output <output> [--interactive] [--no-overwrite]  [--suffix
 [<suffix>]] [--encode]
                   --wrapping-keys  [--wrapping-keys] ...
                       [key=<keyID>] [key=<keyID>] ...
                       [discovery={true|false}] [discovery-partition=<aws-partition-
name> discovery-account=<aws-account-ID> [discovery-account=<aws-account-ID>] ...]
                       [provider=<provider-name>] [region=<aws-region>]
 [profile=<aws-profile>]
                   --metadata-output <location> [--overwrite-metadata] | --suppress-
metadata]
                   [--commitment-policy <commitment-policy>]
                   [--encryption-context <encryption_context> [<encryption_context>
 ...]]
                   [--buffer]
                   [--max-encrypted-data-keys <integer>]
                   [--caching <attributes>]
                   [--max-length <length>]
                   [-v | -vv | -vvv | -vvvv]
                   [--quiet]
```

Utiliser des fichiers de configuration

Vous pouvez faire référence aux fichiers de configuration qui contiennent les paramètres et leurs valeurs. Cela équivaut à saisir les paramètres et les valeurs dans la commande. Pour voir un exemple, consultez Procédure pour stocker les paramètres dans un fichier de configuration.

```
aws-encryption-cli @<configuration_file>

# In a PowerShell console, use a backtick to escape the @.
aws-encryption-cli `@<configuration_file>
```

AWSParamètres de ligne de commande CLI

Cette liste fournit une description de base des paramètres de la commandeAWS Encryption CLI. Pour une description complète, consultez la aws-encryption-sdk-clidocumentation.

--encrypt (-e)

Chiffre les données d'entrée. Chaque commande doit avoir un`--decrypt-unsigned` paramètre`--encrypt--decrypt`, ou, ou.

--decrypt (-d)

Déchiffre les données d'entrée. Chaque commande doit avoir un`--decrypt-unsigned` paramètre`--encrypt--decrypt`, ou.

--decrypt-unsigned [Introduit dans les versions 1.9. x et 2.2. x]

Le`--decrypt-unsigned` paramètre déchiffre le texte chiffré et garantit que les messages ne sont pas signés avant le déchiffrement. Utilisez ce paramètre si vous l'`--algorithm`avez utilisé et sélectionné une suite d'algorithmes sans signature numérique pour crypter les données. Si le texte chiffré est signé, le déchiffrement échoue.

Vous pouvez utiliser`--decrypt` ou`--decrypt-unsigned` pour le déchiffrement, mais pas les deux.

--wrapping-keys (-w) [Introduit dans la version 1.8. x]

Spécifie les clés d'encapsulation (ou clés principales) utilisées dans les opérations de chiffrement et de déchiffrement. Vous pouvez utiliser plusieurs`--wrapping-keys` paramètres dans chaque commande.

À partir de version 2.1. x, le`--wrapping-keys` paramètre est requis dans les commandes de chiffrement et de déchiffrement. Dans la version 1.8. x, les commandes de chiffrement nécessitent un`--wrapping-keys` ou un`--master-keys` paramètre. Dans la version 1.8. x commandes de déchiffrement, un`--wrapping-keys` paramètre est facultatif mais recommandé.

Lorsque vous utilisez un fournisseur de clé principale personnalisé, les commandes de chiffrement et de déchiffrement nécessitent des attributs de clé et de fournisseur. Lors de leur utilisationAWS KMS keys, les commandes de chiffrement nécessitent un attribut clé. Les commandes de déchiffrement nécessitent un attribut clé ou un attribut de découverte avec une valeur de`true` (mais pas les deux). Il est recommandé d'utiliser l'attribut clé lors du AWS Encryption SDKdéchiffrement. C'est particulièrement important si vous déchiffrez des lots de messages inconnus, tels que ceux qui se trouvent dans un compartiment Amazon S3 ou une file d'attente Amazon SQS.

Pour un exemple montrant comment utiliser des clésAWS KMS multirégions comme clés d'encapsulation, consultezUtilisation de plusieurs régions AWS KMS keys.

Attributs : la valeur du paramètre `--wrapping-keys` comprend les attributs suivants. Le format est le suivant `attribute_name=value`.

key

Identifie la clé d'encapsulation utilisée dans l'opération. Le format est une clé=paire d'ID. Vous pouvez spécifier plusieurs attributs key dans chaque valeur du paramètre `--wrapping-keys`.

- Commandes de chiffrement : toutes les commandes de chiffrement nécessitent l'attribut clé. Lorsque vous utilisez une commandeAWS KMS key in an encrypt, la valeur de l'attribut clé peut être un ID de clé, un ARN de clé, un nom d'alias ou un ARN d'alias. Pour obtenir une description des identificateursAWS KMS clés, voir Identifiants clés dans le Guide duAWS Key Management Service développeur.
- Commandes de déchiffrement : lors du déchiffrement avecAWS KMS keys, le`--wrapping-keys` paramètre nécessite un attribut clé avec une valeur ARN clé ou un attribut de découverte avec une valeur de`true` (mais pas les deux). L'utilisation de l'attribut clé est une AWS Encryption SDKbonne pratique. Lors du déchiffrement avec un fournisseur de clé principale personnalisé, l'attribut clé est obligatoire.

> **Note**
>
> Pour spécifier une cléAWS KMS d'encapsulation dans une commande de déchiffrement, la valeur de l'attribut clé doit être un ARN clé. Si vous utilisez un ID de clé, un nom d'alias ou un ARN d'alias, la CLI deAWS chiffrement ne reconnaît pas la clé d'encapsulation.

Vous pouvez spécifier plusieurs attributs key dans chaque valeur du paramètre `--wrapping-keys`. Toutefois, tous les attributs de fournisseur, de région et de profil d'un`--wrapping-keys` paramètre s'appliquent à toutes les clés d'encapsulation de cette valeur de paramètre. Pour spécifier des clés d'encapsulation avec différentes valeurs d'attribut, utilisez plusieurs`--wrapping-keys` paramètres dans la commande.

découverte

Autorise la CLI deAWS chiffrement à utiliser n'importeAWS KMS key laquelle pour déchiffrer le message. La valeur de découverte peut être`true` ou`false`. La valeur par défaut est `false`. L'attribut de découverte n'est valide que dans les commandes de déchiffrement et uniquement lorsque le fournisseur de clé principale l'estAWS KMS.

Lors du déchiffrement avecAWS KMS keys, le`--wrapping-keys` paramètre nécessite un attribut clé ou un attribut de découverte avec une valeur de`true` (mais pas les deux). Si vous utilisez l'attribut key, vous pouvez utiliser un attribut de découverte avec une valeur de`false` pour rejeter explicitement la découverte.

- `False`(par défaut) — Lorsque l'attribut de découverte n'est pas spécifié ou que sa valeur l'est`false`, la CLI deAWS chiffrement déchiffre le message en utilisant uniquement l'attributAWS KMS keys spécifié par la clé du`--wrapping-keys` paramètre. Si vous ne spécifiez aucun attribut clé lors de la découverte`false`, la commande de déchiffrement échoue. Cette valeur est conforme aux meilleures pratiques de l'interface de ligne de commande deAWS chiffrement.
- `True`— Lorsque la valeur de l'attribut de découverte est`true` égale à, la CLIAWS KMS keys deAWS chiffrement obtient les métadonnées du message crypté et les utiliseAWS KMS keys pour déchiffrer le message. L'attribut discovery dont la valeur est`true` se comporte comme les versions de l'AWSEncryption CLI antérieures à la version 1.8. x qui ne vous permettait pas de spécifier une clé d'encapsulation lors du déchiffrement. Toutefois, votre intention d'en utiliserAWS KMS key est explicite. Si vous spécifiez un attribut clé lors de la découverte`true`, la commande de déchiffrement échoue.

  La`true` valeur peut entraîner l'utilisation de la CLI deAWS chiffrementAWS KMS keys dans différentes régionsComptes AWS et, ou une tentative d'utilisationAWS KMS keys que l'utilisateur n'est pas autorisé à utiliser.

Lorsque c'est le cas**`true`**, il est recommandé d'utiliser les attributs discovery-partition et discovery-account pour limiter les attributsAWS KMS keys utilisés à ceuxComptes AWS que vous spécifiez.

compte de découverte

Limite les valeursAWS KMS keys utilisées pour le déchiffrement à celles spécifiéesCompte AWS. La seule valeur valide pour cet attribut est un Compte AWSidentifiant.

Cet attribut est facultatif et valide uniquement dans les commandes de déchiffrement dansAWS KMS keys lesquelles l'attribut discovery est défini`true` et l'attribut discovery-partition est spécifié.

Chaque attribut de compte de découverte ne prend qu'un seulCompte AWS identifiant, mais vous pouvez spécifier plusieurs attributs de compte de découverte dans le même`--wrapping-keys` paramètre. Tous les comptes spécifiés dans un`--wrapping-keys` paramètre donné doivent se trouver dans laAWS partition spécifiée.

découverte

Spécifie laAWS partition des comptes dans l'attribut discovery-account. Sa valeur doit être uneAWS partition, telle que`aws`, `aws-cn`, ou`aws-gov-cloud`. Pour plus d'informations, consultez Amazon Resource Names dans le Références générales AWS.

Cet attribut est obligatoire lorsque vous utilisez l'attribut discovery-account. Vous ne pouvez spécifier qu'un seul attribut discovery-partition dans chaque`--wrapping keys` paramètre. Pour spécifierComptes AWS dans plusieurs partitions, utilisez un`--wrapping-keys` paramètre supplémentaire.

provider

Identifie le fournisseur de clés principales. Le format est un fournisseur=paire d'ID. La valeur par défaut, aws-kms, représente AWS KMS. Cet attribut est obligatoire uniquement lorsque le fournisseur de clés principales n'est pas AWS KMS.

Région

IdentifieRégion AWS leAWS KMS key. Cet attribut n'est valide que pourAWS KMS keys. Il est utilisé uniquement lorsque l'identifiant de l'attribut key ne spécifie pas de région ; sinon, il est ignoré. Lorsqu'il est utilisé, il remplace la région par défaut dans le profil nommé de l'interface de ligne de commande AWS.

profile

Identifie un AWS CLIprofil nommé pour l'. Cet attribut n'est valide que pourAWS KMS keys. La région du profil est utilisée uniquement lorsque l'identifiant de clé ne spécifie pas de région et qu'il n'y a pas d'attribut region dans la commande.

--input (-i)

Spécifie l'emplacement des données à chiffrer ou déchiffrer. Ce paramètre est obligatoire. La valeur peut être un chemin d'accès à un fichier ou à un répertoire, ou un modèle de nom de fichier. Si vous dirigez l'entrée vers la commande (stdin), utilisez `-`.

Si l'entrée n'existe pas, la commande se termine correctement sans erreur ni avertissement.

--recursive (-r, -R)

Effectue l'opération sur les fichiers du répertoire d'entrée et ses sous-répertoires. Ce paramètre est obligatoire lorsque la valeur de `--input` est un répertoire.

--decode

Décode l'entrée codée en Base64.

Si vous déchiffrez un message qui a été chiffré puis codé, vous devez décoder le message avant de le déchiffrer. C'est ce que ce paramètre fait pour vous.

Par exemple, si vous avez utilisé le paramètre `--encode` dans une commande de chiffrement, utilisez le paramètre `--decode` dans la commande de déchiffrement correspondante. Vous pouvez également utiliser ce paramètre pour décoder l'entrée codée en Base64 avant de la chiffrer.

--output (-o)

Spécifie une destination pour la sortie. Ce paramètre est obligatoire. La valeur peut être un nom de fichier, un répertoire existant ou `-`, qui écrit la sortie dans la ligne de commande (stdout).

Si le répertoire de sortie spécifié n'existe pas, la commande échoue. Si l'entrée contient des sous-répertoires, la CLI deAWS chiffrement reproduit les sous-répertoires dans le répertoire de sortie que vous spécifiez.

Par défaut, la CLI deAWS chiffrement remplace les fichiers portant le même nom. Pour modifier ce comportement, utilisez les paramètres `--interactive` ou `--no-overwrite`. Pour supprimer l'avertissement de remplacement, utilisez le paramètre `--quiet`.

> **Note**
>
> Si une commande susceptible de remplacer un fichier de sortie échoue, le fichier de sortie est supprimé.

--interactif

Envoie une invite avant de remplacer le fichier.

--no-overwrite

Ne remplace pas les fichiers. Au lieu de cela, si le fichier de sortie existe, la CLI deAWS chiffrement ignore l'entrée correspondante.

--suffix

Spécifie un suffixe de nom de fichier personnalisé pour les fichiers créés par la CLI deAWS chiffrement. Pour indiquer que vous ne souhaitez aucun suffixe, utilisez le paramètre sans valeur (`--suffix`).

Par défaut, lorsque le paramètre `--output` ne spécifie pas de nom de fichier, le nom du fichier de sortie est le même que celui du fichier d'entrée, auquel est ajouté le suffixe. Le suffixe pour les commandes de chiffrement est `.encrypted`. Le suffixe pour les commandes de déchiffrement est `.decrypted`.

--encode

Applique le codage Base64 (binaire en texte) à la sortie. Le codage empêche le programme hôte du shell de mal interpréter les caractères non ASCII dans le texte de sortie.

Utilisez ce paramètre lorsque vous écrivez une sortie cryptée dans stdout (`--output -`), en particulier dans une PowerShell console, même lorsque vous redirigez la sortie vers une autre commande ou que vous l'enregistrez dans une variable.

--metadata-output

Spécifie un emplacement pour les métadonnées relatives aux opérations de chiffrement. Saisissez un chemin et un nom de fichier. Si le répertoire n'existe pas, la commande échoue. Pour écrire les métadonnées sur la ligne de commande (stdout), utilisez `-`.

Vous ne pouvez pas écrire la sortie de commande (`--output`) et la sortie des métadonnées (`--metadata-output`) dans stdout dans la même commande. De plus, lorsque la valeur de `--input` ou `--output` est un répertoire (sans nom de fichier), vous ne pouvez pas écrire la sortie des métadonnées dans le même répertoire ou dans un sous-répertoire de ce répertoire.

Si vous spécifiez un fichier existant, la CLI deAWS chiffrement ajoute par défaut de nouveaux enregistrements de métadonnées à tout contenu du fichier. Cette fonctionnalité vous permet de créer un fichier unique qui contient les métadonnées pour l'ensemble de vos opérations de chiffrement. Pour remplacer le contenu dans un fichier existant, utilisez le paramètre `--overwrite-metadata`.

La CLI deAWS chiffrement renvoie un enregistrement de métadonnées au format JSON pour chaque opération de chiffrement ou de déchiffrement exécutée par la commande. Chaque enregistrement de métadonnées inclut les chemins d'accès complets aux fichiers d'entrée et de sortie, le contexte de chiffrement, la suite d'algorithmes, et d'autres informations utiles que vous pouvez utiliser pour vérifier l'opération et vous assurer qu'elle respecte vos normes de sécurité.

--overwrite-metadata

Remplace le contenu dans le fichier de sortie des métadonnées. Par défaut, le paramètre `--metadata-output` ajoute les métadonnées à un contenu existant dans le fichier.

--suppress-metadata (-S)

Supprime les métadonnées relatives à l'opération de chiffrement ou de déchiffrement.

--politique-d'engagement

Spécifie la politique d'engagement pour les commandes de chiffrement et de déchiffrement. La politique d'engagement détermine si votre message est crypté et déchiffré à l'aide de la fonctionnalité clé de sécurité d'engagement.

Le`--commitment-policy` paramètre est introduit dans la version 1.8. x. Il est valide dans les commandes de chiffrement et de déchiffrement.

Dans la version 1.8. x, la CLI deAWS chiffrement utilise la politique d'`forbid-encrypt-allow-decrypt`engagement pour toutes les opérations de chiffrement et de déchiffrement. Lorsque vous utilisez le`--wrapping-keys` paramètre dans une commande de chiffrement ou de déchiffrement, un`--commitment-policy` paramètre avec la`forbid-encrypt-allow-decrypt` valeur est requis. Si vous n'utilisez pas le`--wrapping-keys` paramètre, celui-ci n'est pas valide.`--commitment-policy` La définition d'une politique d'engagement empêche explicitement la modification automatique de votre politique d'engagement`require-encrypt-require-decrypt` lors de la mise à niveau vers la version 2.1. x

À partir de version 2.1. x, toutes les valeurs de la politique d'engagement sont prises en charge. Le`--commitment-policy` paramètre est facultatif et la valeur par défaut est`require-encrypt-require-decrypt`.

Ce paramètre a les valeurs suivantes:

- `forbid-encrypt-allow-decrypt`— Impossible de chiffrer avec la saisie de la clé. Il peut déchiffrer des textes chiffrés avec ou sans saisie de clé.

  Dans la version 1.8. x, c'est la seule valeur valide. La CLI deAWS chiffrement utilise la politique d'`forbid-encrypt-allow-decrypt`engagement pour toutes les opérations de chiffrement et de déchiffrement.
- `require-encrypt-allow-decrypt`— Chiffre uniquement avec un engagement clé. Déchiffre avec et sans engagement de clé. Cette valeur est introduite dans la version 2.1. x.

- `require-encrypt-require-decrypt`(par défaut) — Chiffre et déchiffre uniquement avec un engagement de clé. Cette valeur est introduite dans la version 2.1. x. Il s'agit de la valeur par défaut dans les versions 2.1. x et versions ultérieures. Avec cette valeur, l'interface de ligne de commande deAWS chiffrement ne déchiffrera aucun texte chiffré avec des versions antérieures duAWS Encryption SDK.

Pour plus d'informations sur la définition de votre politique d'engagement, consultezMigrer votreAWS Encryption SDK.

--encryption-context (-c)

Spécifie un contexte de chiffrement pour l'opération. Ce paramètre n'est pas obligatoire, mais il est recommandé.

- Dans une commande `--encrypt`, entrez une ou plusieurs paires `name=value`. Utilisez des espaces pour séparer les paires.
- Dans une`--decrypt` commande, entrez des`name=value` paires,`name` des éléments sans valeur, ou les deux.

Si `name` ou `value` dans une paire `name=value` contient des espaces ou des caractères spéciaux, placez la totalité de la paire entre guillemets. Par exemple, `--encryption-context "department=software development"`.

--buffer (-b) [Introduit dans les versions 1.9. x et 2.2. x]

Renvoie le texte en clair uniquement après le traitement de toutes les entrées, y compris la vérification de la signature numérique si elle est présente.

--max-encrypted-data-keys [Introduit dans les versions 1.9. x et 2.2. x]

Spécifie le nombre maximal de clés de données cryptées dans un message crypté. Ce paramètre est facultatif.

Les valeurs valides vont de 1 à 65 535. Si vous omettez ce paramètre, la CLI deAWS chiffrement n'impose pas de maximum. Un message crypté peut contenir jusqu'à 65 535 (2^16 - 1) clés de données cryptées.

Vous pouvez utiliser ce paramètre dans les commandes de chiffrement afin d'éviter un message mal formé. Vous pouvez l'utiliser dans des commandes de déchiffrement pour détecter les messages malveillants et éviter de déchiffrer des messages contenant de nombreuses clés de données cryptées que vous ne pouvez pas déchiffrer. Pour plus de détails et un exemple, reportez-vous à la section Limiter les clés de données chiffrées.

--help (-h)

Imprime l'utilisation et la syntaxe dans la ligne de commande.

--Version

Obtient la version de la CLI deAWS chiffrement.

-v | -vv | -vvv | -vvvv

Affiche des informations détaillées, des avertissements et des messages de débogage. Le niveau de détail dans la sortie augmente avec le nombre de `v` dans le paramètre. Le paramètre le plus détaillé (`-vvvv`) renvoie des données de débogage à partir de la CLI deAWS chiffrement et de tous les composants qu'elle utilise.

--quiet (-q)

Supprime les messages d'avertissement, comme le message qui s'affiche lorsque vous remplacez un fichier de sortie.

--master-keys (-m) [Obsolète]

> **Note**
>
> Le paramètre --master-keys est obsolète depuis la version 1.8. x et supprimé dans la version 2.1. x. Utilisez plutôt le paramètre --wrapping-keys.

Spécifie les clés principales utilisées dans les opérations de chiffrement et de déchiffrement. Vous pouvez utiliser plusieurs paramètres de clés principales dans chaque commande.

Le paramètre `--master-keys` est obligatoire dans les commandes de chiffrement. Elle est requise dans les commandes de déchiffrement uniquement lorsque vous utilisez un fournisseur de clé (nonAWS KMS) principale personnalisé.

Attributs : la valeur du paramètre `--master-keys` comprend les attributs suivants. Le format est le suivant `attribute_name=value`.

key

Identifie la clé d'encapsulation utilisée dans l'opération. Le format est une clé=paire d'ID. L'attribut key est obligatoire dans toutes les commandes de chiffrement.

Lorsque vous utilisez une commandeAWS KMS key in an encrypt, la valeur de l'attribut clé peut être un ID de clé, un ARN de clé, un nom d'alias ou un ARN d'alias. Pour plus de

détails surAWS KMS les identificateurs clés, voir Identifiants clés dans le Guide duAWS Key Management Service développeur.

L'attribut clé est requis dans les commandes de déchiffrement lorsque le fournisseur de clé principale ne l'est pasAWS KMS. L'attribut clé n'est pas autorisé dans les commandes qui déchiffrent des données chiffrées dans le cadre d'unAWS KMS key.

Vous pouvez spécifier plusieurs attributs key dans chaque valeur du paramètre `--master-keys`. Toutefois, n'importe quel attribut provider, region et profile s'applique à toutes les clés principales de la valeur du paramètre. Pour spécifier des clés principales avec différentes valeurs d'attribut, utilisez plusieurs paramètres `--master-keys` dans la commande.

provider

Identifie le fournisseur de clés principales. Le format est un fournisseur=paire d'ID. La valeur par défaut, aws-kms, représente AWS KMS. Cet attribut est obligatoire uniquement lorsque le fournisseur de clés principales n'est pas AWS KMS.

Région

IdentifieRégion AWS leAWS KMS key. Cet attribut n'est valide que pourAWS KMS keys. Il est utilisé uniquement lorsque l'identifiant de l'attribut key ne spécifie pas de région ; sinon, il est ignoré. Lorsqu'il est utilisé, il remplace la région par défaut dans le profil nommé de l'interface de ligne de commande AWS.

profile

Identifie un AWS CLIprofil nommé pour l'. Cet attribut n'est valide que pourAWS KMS keys. La région du profil est utilisée uniquement lorsque l'identifiant de clé ne spécifie pas de région et qu'il n'y a pas d'attribut region dans la commande.

Paramètres avancés

--algorithm

Spécifie une autre suite d'algorithmes. Ce paramètre est facultatif et uniquement valide dans les commandes de chiffrement.

Si vous omettez ce paramètre, la CLI deAWS chiffrement utilise l'une des suites d'algorithmes par défaut pour les versionsAWS Encryption SDK introduites dans la version 1.8. x. Les deux algorithmes par défaut utilisent AES-GCM avec un HKDF, une signature ECDSA et une clé

de cryptage 256 bits. L'un utilise un engagement clé ; l'autre ne le fait pas. Le choix de la suite d'algorithmes par défaut est déterminé par la politique d'engagement de la commande.

Les suites d'algorithmes par défaut sont recommandées pour la plupart des opérations de chiffrement. Pour obtenir une liste des valeurs valides, consultez les valeurs du paramètre `algorithm` dans Lisez les documents.

--frame-length

Crée la sortie avec la longueur de cadre spécifiée. Ce paramètre est facultatif et uniquement valide dans les commandes de chiffrement.

Entrez une valeur en octets. Les valeurs valides sont 0 et 1 — 2^31 - 1. La valeur 0 indique des données sans cadre. La valeur par défaut est de 4 096 (octets).

> **Note**
>
> Dans la mesure du possible, utilisez des données encadrées. IIAWS Encryption SDK prend en charge les données sans cadre uniquement pour une utilisation traditionnelle. Certaines implémentations linguistiques duAWS Encryption SDK peuvent toujours générer du texte chiffré sans cadre. Toutes les implémentations linguistiques prises en charge peuvent déchiffrer du texte chiffré encadré et non encadré.

--max-length

Indique la taille de trame maximale (ou la longueur maximale du contenu pour les messages non cadrés), en octets, à lire à partir de messages chiffrés. Ce paramètre est facultatif et uniquement valide dans les commandes de déchiffrement. Il est conçu pour vous protéger contre le déchiffrement de gros volumes de texte chiffré malveillant.

Entrez une valeur en octets. Si vous omettez ce paramètre, la taille d'imageAWS Encryption SDK ne limite pas la taille d'image lors du déchiffrement.

--caching

Active la fonctionnalité de mise en cache des clés de données, qui réutilise des clés de données plutôt que de générer une nouvelle clé de données pour chaque fichier d'entrée. Ce paramètre prend en charge un scénario plus avancé. Veillez à bien lire la documentation Mise en cache des clés de données avant d'utiliser cette fonctionnalité.

Le paramètre `--caching` possède les attributs suivants.

capacity (obligatoire)

Détermine le nombre maximum d'entrées dans le cache.

La valeur minimale est de 1. Il n'y a pas de valeur maximale.

max_age (obligatoire)

Déterminez la durée pendant laquelle les entrées du cache sont utilisées, en secondes, à compter de leur ajout au cache.

Saisissez une valeur supérieure à 0. Il n'y a pas de valeur maximale.

max_messages_encrypted (facultatif)

Détermine le nombre maximal de messages qu'une entrée mise en cache peut chiffrer.

Les valeurs valides vont de 1 à 2^32. La valeur par défaut est 2^32 (messages).

max_bytes_encrypted (facultatif)

Détermine le nombre maximal d'octets qu'une entrée mise en cache peut chiffrer.

Les valeurs valides sont 0 et 1 — 2^63 - 1. La valeur par défaut est 2^63 - 1 (messages). Une valeur de 0 vous permet d'utiliser la mise en cache des clés de données uniquement lorsque vous chiffrez des chaînes de message vides.

Versions duAWSCLI de chiffrement

Nous vous recommandons d'utiliser la dernière version de l'interface de l'interface de laAWSCLI de chiffrement.

> **Note**
>
> Versions duAWSL'interface de l'interface de l'interface de l'interface de l'interfaceend-of-supportphase.
> Vous pouvez mettre à jour en toute sécurité depuis la version 2.1.Xet plus tard vers la dernière version duAWSCryptage CLI sans aucune modification du code ou des données. Cependantnouvelles fonctions de sécuritéintroduit dans la version 2.1.Xne sont pas rétrocompatibles. Pour mettre à jour depuis la version 1.7.Xou version antérieure, vous devez d'abord la mettre à jour vers la version la plus récente.Xversion duAWSCLI de chiffrement. Pour plus d'informations, consultez.Migrer votreAWS Encryption SDK

Les nouvelles fonctionnalités de sécurité ont été initialement publiées enAWSCryptage CLI version 1.7.Xet 2.0X. CependantAWSVersion 1.8 de la CLI de chiffrementXremplace la version 1.7.XetAWSCryptage CLI.Xremplace la 2.0X. Pour plus d'informations, veuillez consulter laAvis de sécuritédans leaws-encryption-sdk-cliréférentiel sur GitHub.

Pour plus d'informations sur les versions importantes deAWS Encryption SDKVeuillez consulterVersions du AWS Encryption SDK.

Quelle version dois-je utiliser ?

Si vous débutez avec leAWSEncryption CLI, utilisez la version la plus récente.

Pour déchiffrer des données cryptées par une version duAWS Encryption SDKversion inférieure à la version 1.7.X, migrer d'abord vers la dernière version de l'interface de l'interfaceAWSCLI de chiffrement. Marquetoutes les modifications recommandéesavant la mise à jour vers la version 2.1.Xou plus tard. Pour plus d'informations, consultez.Migrer votreAWS Encryption SDK

En savoir plus

- Pour des informations détaillées sur les modifications et des instructions relatives à la migration vers ces nouvelles versions, voirMigrer votreAWS Encryption SDK.
- Pour les descriptions de la nouvelleAWSParamètres et attributs de la CLI de chiffrement, voirRéférence des paramètres et de la syntaxe de l'interface de ligne de commande du kit AWS Encryption SDK.

Les listes suivantes décrivent la modification de laAWSCLI de chiffrement dans les versions 1.8.Xet 2.1.X.

Version 1.8XmodificationsAWSCLI de chiffrement

- Obsolète`--master-keys`paramètre. Utilisez plutôt le paramètre `--wrapping-keys`.
- Ajout`--wrapping-keys`(`-w`paramètre. Il prend en charge tous les attributs du`--master-keys`paramètre. Il ajoute également les attributs facultatifs suivants, qui ne sont valides que lors du déchiffrement avecAWS KMS keys.
  - découverte
  - partition de découverte
  - compte-découverte

Pour les fournisseurs de clés principales personnalisées,`--encrypt`et `--decrypt`les commandes nécessitent soit un`--wrapping-keys`paramètre`--master-keys`paramètre (mais pas les deux). En outre`--encrypt`commande avecAWS KMS keysnécessite soit un`--wrapping-keys`paramètre`--master-keys`paramètre (mais pas les deux).

Dans un`--decrypt`commande avecAWS KMS keys, le`--wrapping-keys`Le paramètre est facultatif, mais recommandé, car il est requis dans la version 2.1.X. Si vous l'utilisez, vous devez spécifier lacléAttributdécouverteattribut avec une valeur de`true`(mais pas les deux).

- Ajout`--commitment-policy`paramètre. La seule valeur valide est `forbid-encrypt-allow-decrypt`. Dans la`forbid-encrypt-allow-decrypt`la politique d'engagement est utilisée dans toutes les commandes de chiffrement et de déchiffrement.

  Dans la version 1.8X, lorsque vous utilisez`--wrapping-keys`paramètre`--commitment-policy`paramètre avec le`forbid-encrypt-allow-decrypt`valeur est obligatoire. La définition de la valeur empêche explicitement votrePolitique d'engagementdu passage automatique à`require-encrypt-require-decrypt`lors de la mise à niveau vers la version 2.1.X.

Version 2.1XmodificationsAWSCLI de chiffrement

- Supprime`--master-keys`paramètre. Utilisez plutôt le paramètre `--wrapping-keys`.
- Dans la`--wrapping-keys`Le paramètre est requis dans toutes les commandes de chiffrement et de déchiffrement. Vous devez spécifier oucléAttributdécouverteattribut avec une valeur de`true`(mais pas les deux).
- Dans la`--commitment-policy`Le paramètre prend en charge les valeurs suivantes. Pour plus d'informations, consultez.Définition de votre politique d'engagement
  - `forbid-encrypt-allow-decrypt`
  - `require-encrypt-allow-decrypt`
  - `require-encrypt-require decrypt`(Valeur)
- Dans la`--commitment-policy`Le paramètre est facultatif dans la version 2.1.X. La valeur par défaut est `require-encrypt-require-decrypt`.

Version 1.9Xet 2.2.XmodificationsAWSCLI de chiffrement

- Ajout`--decrypt-unsigned`paramètre. Pour plus d'informations, consultez.La version 2.2. x

- Ajout`--buffer`paramètre. Pour plus d'informations, consultez.La version 2.2. x
- Ajout`--max-encrypted-data-keys`paramètre. Pour plus d'informations, consultez.Limiter les clés de données chiffrées

Version 3.0XmodificationsAWSCLI de chiffrement

- Ajoute le support pourAWS KMSclés multi-régions. Pour plus d'informations, consultez Utilisation de plusieurs régions AWS KMS keys.

Mise en cache des clés de données

La mise en cache des clés de données stocke les clés de données et les matériaux de chiffrement connexes dans un cache. Lorsque vous chiffrez ou déchiffrez des données, la AWS Encryption SDK clé de données correspondante est recherchée dans le cache. Si une correspondance est trouvée, il utilise la clé de données mise en cache au lieu d'en générer une nouvelle. La mise en cache des clés de données peut améliorer les performances, réduire les coûts et vous aider à respecter les limites de service lorsque votre application grandit.

Votre application peut tirer parti de la mise en cache des clés de données si :

- elle peut réutiliser les clés de données ;
- elle génère de nombreuses clés de données ;
- vos opérations de chiffrement sont trop lentes, trop coûteuses, trop limitées ou utilisent trop de ressources.

La mise en cache peut réduire votre utilisation de services cryptographiques, tels que AWS Key Management Service ()AWS KMS. Si vous atteignez votre AWS KMS requests-per-secondlimite, la mise en cache peut vous aider. Votre application peut utiliser des clés mises en cache pour répondre à certaines de vos demandes de clés de données au lieu d'appeler AWS KMS. (Vous pouvez également créer un dossier dans le AWS Support Center pour augmenter la limite de votre compte.)

Il vous AWS Encryption SDK aide à créer et à gérer votre cache de clés de données. Il fournit un cache local et un gestionnaire de matériel cryptographique de mise en cache (CMM de mise en cache) qui interagit avec le cache et applique les seuils de sécurité que vous définissez. Associés, ces composants vous permettent de bénéficier de l'efficacité que génère la réutilisation des clés de données, tout en préservant la sécurité de votre système.

La mise en cache des clés de données est une fonctionnalité facultative AWS Encryption SDK que vous devez utiliser avec prudence. Par défaut, une nouvelle clé de données est générée pour chaque opération de chiffrement. AWS Encryption SDK Cette technique prend en charge les bonnes pratiques de chiffrement, ce qui dissuade de réutiliser excessivement les clés de données. En général, utilisez la mise en cache des clés de données uniquement lorsqu'elle est nécessaire pour atteindre vos objectifs de performance. Ensuite, utilisez les seuils de sécurité de la mise en cache des clés de données afin vous assurer que vous utilisez la quantité minimale de mise en cache requise pour atteindre vos objectifs de coûts et de performance.

Le CMM de mise en cache n'est pas pris en charge par le AWS Encryption SDK pour .NET. Version 3. x of the prend Kit SDK de chiffrement AWS pour Java uniquement en charge le CMM de mise en cache avec l'ancienne interface des fournisseurs de clés principales, et non avec l'interface keyring. Cependant, version 4. x du AWS Encryption SDK pour .NET et la version 3. x du Kit SDK de chiffrement AWS pour Java support le trousseau de clés AWS KMS hiérarchique, une solution alternative de mise en cache de matériaux cryptographiques. Le contenu chiffré avec le trousseau de clés AWS KMS hiérarchique ne peut être déchiffré qu'avec le trousseau de clés hiérarchique. AWS KMS

Pour une discussion détaillée de ces compromis en matière de sécurité, voir AWS Encryption SDK: Comment déterminer si la mise en cache des clés de données convient à votre application dans le blog sur la AWS sécurité.

Rubriques

Utilisation de la mise en cache des clés de données

Cette rubrique vous montre comment utiliser la mise en cache des clés de données dans votre application. Elle vous guide à travers le processus étape par étape. Elle associe ensuite les étapes dans un exemple simple qui utilise la mise en cache des clés de données dans une opération visant à chiffrer une chaîne.

Les exemples de cette section montrent comment utiliser la version 2.0. x et versions ultérieures duAWS Encryption SDK. Pour les exemples utilisant des versions antérieures, recherchez votre version dans la liste des versions du GitHub référentiel de votre langage de programmation.

Pour obtenir des exemples complets et testés d'utilisation de la mise en cache de clés de données dans le AWS Encryption SDK, veuillez consulter :

- C/C++ : caching_cmm.cpp
- Java : SimpleDataKeyCachingExample.java
- JavaScript Navigateur : caching_cmm.ts
- JavaScript Node.js : caching_cmm.ts

- Python : data_key_caching_basic.py

AWS Encryption SDKfor .NET ne prend pas en charge la mise en cache des clés de données.

Rubriques

- Utilisation de la mise en cache des clés de données : S tep-by-step
- Exemple de mise en cache des clés de données : chiffrement d'une chaîne

Utilisation de la mise en cache des clés de données : S tep-by-step

Ces step-by-step instructions vous montrent comment créer les composants dont vous avez besoin pour implémenter la mise en cache des clés de données.

- Création d'un cache de clé de données Dans ces exemples, nous utilisons le cache local AWS Encryption SDK fourni par le. Nous limitons le cache à 10 clés de données.

C

```
// Cache capacity (maximum number of entries) is required
size_t cache_capacity = 10;
struct aws_allocator *allocator = aws_default_allocator();

struct aws_cryptosdk_materials_cache *cache =
 aws_cryptosdk_materials_cache_local_new(allocator, cache_capacity);
```

Java

L'exemple suivant utilise la version 2. x duKit SDK de chiffrement AWS pour Java. Version 3. x du rend Kit SDK de chiffrement AWS pour Java obsolète le CMM de mise en cache des clés de données. Avec la version 3. x, vous pouvez également utiliser le trousseau de clés AWS KMS hiérarchique, une solution alternative de mise en cache des matériaux cryptographiques.

```
// Cache capacity (maximum number of entries) is required
int MAX_CACHE_SIZE = 10;

CryptoMaterialsCache cache = new LocalCryptoMaterialsCache(MAX_CACHE_SIZE);
```

JavaScript Browser

```
const capacity = 10

const cache = getLocalCryptographicMaterialsCache(capacity)
```

JavaScript Node.js

```
const capacity = 10

const cache = getLocalCryptographicMaterialsCache(capacity)
```

Python

```
# Cache capacity (maximum number of entries) is required
MAX_CACHE_SIZE = 10

cache = aws_encryption_sdk.LocalCryptoMaterialsCache(MAX_CACHE_SIZE)
```

- Créez un fournisseur de clés principales (Java et Python) ou un trousseau de clés (C et JavaScript). Ces exemples utilisent un fournisseur de clés principales AWS Key Management Service (AWS KMS) ou un trousseau de AWS KMSclés compatible.

  C

  ```
  // Create an AWS KMS keyring
  //   The input is the Amazon Resource Name (ARN)
  //   of an AWS KMS key

  struct aws_cryptosdk_keyring *kms_keyring =
   Aws::Cryptosdk::KmsKeyring::Builder().Build(kms_key_arn);
  ```

  Java

  L'exemple suivant utilise la version 2. x duKit SDK de chiffrement AWS pour Java. Version 3. x du rend Kit SDK de chiffrement AWS pour Java obsolète le CMM de mise en cache des clés de

données. Avec la version 3. x, vous pouvez également utiliser le trousseau de clés AWS KMS hiérarchique, une solution alternative de mise en cache des matériaux cryptographiques.

```
// Create an AWS KMS master key provider
//   The input is the Amazon Resource Name (ARN)
//   of an AWS KMS key

MasterKeyProvider<KmsMasterKey> keyProvider =
 KmsMasterKeyProvider.builder().buildStrict(kmsKeyArn);
```

JavaScript Browser

Dans le navigateur, vous devez injecter vos informations d'identification en toute sécurité. Cet exemple définit les informations d'identification dans un webpack (kms.webpack.config) qui résout les informations d'identification lors de l'exécution. Il crée une instance de fournisseur client AWS KMS à partir d'un client AWS KMS et les informations d'identification. Ensuite, lorsqu'il crée le trousseau de clés, il transmet le fournisseur client au constructeur avec le AWS KMS key (. `generatorKeyId`)

```
const { accessKeyId, secretAccessKey, sessionToken } = credentials

const clientProvider = getClient(KMS, {
    credentials: {
      accessKeyId,
      secretAccessKey,
      sessionToken
    }
  })

/*  Create an AWS KMS keyring
 *  You must configure the AWS KMS keyring with at least one AWS KMS key
 *  The input is the Amazon Resource Name (ARN)
 */ of an AWS KMS key

const keyring = new KmsKeyringBrowser({
    clientProvider,
    generatorKeyId,
    keyIds,
  })
```

JavaScript Node.js

```
/* Create an AWS KMS keyring
 *   The input is the Amazon Resource Name (ARN)
*/   of an AWS KMS key

const keyring = new KmsKeyringNode({ generatorKeyId })
```

Python

```
# Create an AWS KMS master key provider
#  The input is the Amazon Resource Name (ARN)
#  of an AWS KMS key

key_provider =
 aws_encryption_sdk.StrictAwsKmsMasterKeyProvider(key_ids=[kms_key_arn])
```

- Créez un gestionnaire de matériel cryptographique de mise en cache (CMM de mise en cache).

Associez votre CMM de mise en cache à votre cache et à votre fournisseur de clé principale ou à votre trousseau de clés. Définissez ensuite les seuils de sécurité du cache sur le CMM de mise en cache.

C

Dans leKit SDK de chiffrement AWS pour C, vous pouvez créer un CMM de mise en cache à partir d'un CMM sous-jacent, tel que le CMM par défaut, ou à partir d'un trousseau de clés. Cet exemple crée le CMM de mise en cache à partir d'un porte-clés.

Après avoir créé le CMM de mise en cache, vous pouvez libérer vos références au trousseau de clés et au cache. Pour plus de détails, consultez the section called “Comptage des références”.

```
// Create the caching CMM
//   Set the partition ID to NULL.
```

```
//   Set the required maximum age value to 60 seconds.
struct aws_cryptosdk_cmm *caching_cmm =
 aws_cryptosdk_caching_cmm_new_from_keyring(allocator, cache, kms_keyring, NULL,
 60, AWS_TIMESTAMP_SECS);

// Add an optional message threshold
//   The cached data key will not be used for more than 10 messages.
aws_status = aws_cryptosdk_caching_cmm_set_limit_messages(caching_cmm, 10);

// Release your references to the cache and the keyring.
aws_cryptosdk_materials_cache_release(cache);
aws_cryptosdk_keyring_release(kms_keyring);
```

Java

L'exemple suivant utilise la version 2. x duKit SDK de chiffrement AWS pour Java. Version 3. x of the Kit SDK de chiffrement AWS pour Java ne prend pas en charge la mise en cache des clés de données, mais prend en charge le keyring AWS KMS hiérarchique, une solution alternative de mise en cache des matériaux cryptographiques.

```
/*
 * Security thresholds
 *   Max entry age is required.
 *   Max messages (and max bytes) per entry are optional
 */
int MAX_ENTRY_AGE_SECONDS = 60;
int MAX_ENTRY_MSGS = 10;

//Create a caching CMM
CryptoMaterialsManager cachingCmm =
    CachingCryptoMaterialsManager.newBuilder().withMasterKeyProvider(keyProvider)
                                 .withCache(cache)
                                 .withMaxAge(MAX_ENTRY_AGE_SECONDS,
 TimeUnit.SECONDS)
                                 .withMessageUseLimit(MAX_ENTRY_MSGS)
                                 .build();
```

JavaScript Browser

```
/*
 * Security thresholds
 *   Max age (in milliseconds) is required.
```

```
 *   Max messages (and max bytes) per entry are optional.
 */
const maxAge = 1000 * 60
const maxMessagesEncrypted = 10

/* Create a caching CMM from a keyring  */
const cachingCmm = new WebCryptoCachingMaterialsManager({
  backingMaterials: keyring,
  cache,
  maxAge,
  maxMessagesEncrypted
})
```

JavaScript Node.js

```
/*
 * Security thresholds
 *   Max age (in milliseconds) is required.
 *   Max messages (and max bytes) per entry are optional.
 */
const maxAge = 1000 * 60
const maxMessagesEncrypted = 10

/* Create a caching CMM from a keyring  */
const cachingCmm = new NodeCachingMaterialsManager({
  backingMaterials: keyring,
  cache,
  maxAge,
  maxMessagesEncrypted
})
```

Python

```
# Security thresholds
#   Max entry age is required.
#   Max messages (and max bytes) per entry are optional
#
MAX_ENTRY_AGE_SECONDS = 60.0
MAX_ENTRY_MESSAGES = 10

# Create a caching CMM
caching_cmm = CachingCryptoMaterialsManager(
    master_key_provider=key_provider,
```

```
    cache=cache,
    max_age=MAX_ENTRY_AGE_SECONDS,
    max_messages_encrypted=MAX_ENTRY_MESSAGES
)
```

C'est tout ce que vous avez besoin de faire. Ensuite, laissez le kit AWS Encryption SDK gérer le cache pour vous, ou ajoutez votre propre logique de gestion de cache.

Lorsque vous souhaitez utiliser la mise en cache des clés de données dans un appel pour chiffrer ou déchiffrer des données, spécifiez votre CMM de mise en cache plutôt qu'un fournisseur de clé principale ou un autre CMM.

> **Note**
>
> Si vous chiffrez des flux de données, ou des données de taille inconnue, assurez-vous de spécifier la taille des données dans la demande. Le kit AWS Encryption SDK n'utilise pas la mise en cache des clés de données lors du chiffrement des données de taille inconnue.

C

Dans le kit Kit SDK de chiffrement AWS pour C, vous créez une session avec le CMM de mise en cache, puis vous traitez la session.

Par défaut, lorsque la taille du message est inconnue et sans limite, le kit AWS Encryption SDK ne met pas en cache les clés de données. Pour autoriser la mise en cache lorsque vous ne connaissez pas la taille exacte des données, utilisez la méthode `aws_cryptosdk_session_set_message_bound` pour définir une taille maximale pour le message. Définissez une limite supérieure à la taille estimée du message. Si la taille réelle du message dépasse la limite, l'opération échoue.

```
/* Create a session with the caching CMM. Set the session mode to encrypt. */
struct aws_cryptosdk_session *session =
 aws_cryptosdk_session_new_from_cmm_2(allocator, AWS_CRYPTOSDK_ENCRYPT,
 caching_cmm);

/* Set a message bound of 1000 bytes */
aws_status = aws_cryptosdk_session_set_message_bound(session, 1000);

/* Encrypt the message using the session with the caching CMM */
```

```
aws_status = aws_cryptosdk_session_process(
             session, output_buffer, output_capacity, &output_produced,
 input_buffer, input_len, &input_consumed);

/* Release your references to the caching CMM and the session. */
aws_cryptosdk_cmm_release(caching_cmm);
aws_cryptosdk_session_destroy(session);
```

Java

L'exemple suivant utilise la version 2. x duKit SDK de chiffrement AWS pour Java. Version 3. x du rend Kit SDK de chiffrement AWS pour Java obsolète le CMM de mise en cache des clés de données. Avec la version 3. x, vous pouvez également utiliser le trousseau de clés AWS KMS hiérarchique, une solution alternative de mise en cache des matériaux cryptographiques.

```
// When the call to encryptData specifies a caching CMM,
// the encryption operation uses the data key cache
final AwsCrypto encryptionSdk = AwsCrypto.standard();
return encryptionSdk.encryptData(cachingCmm, plaintext_source).getResult();
```

JavaScript Browser

```
const { result } = await encrypt(cachingCmm, plaintext)
```

JavaScript Node.js

Lorsque vous utilisez le CMM de mise en cache dans Kit SDK de chiffrement AWS pour JavaScript for Node.js, la `encrypt` méthode nécessite la longueur du texte en clair. Si vous ne le fournissez pas, la clé de données n'est pas mise en cache. Si vous fournissez une longueur, mais que les données en texte brut que vous fournissez dépassent cette longueur, l'opération de chiffrement échoue. Si vous ne connaissez pas la longueur exacte du texte brut, par exemple lorsque vous diffusez des données, indiquez la plus grande valeur attendue.

```
const { result } = await encrypt(cachingCmm, plaintext, { plaintextLength:
 plaintext.length })
```

Python

```
# Set up an encryption client
client = aws_encryption_sdk.EncryptionSDKClient()
```

```
# When the call to encrypt specifies a caching CMM,
# the encryption operation uses the data key cache
#
encrypted_message, header = client.encrypt(
    source=plaintext_source,
    materials_manager=caching_cmm
)
```

Exemple de mise en cache des clés de données : chiffrement d'une chaîne

Cet exemple de code simple utilise la mise en cache des clés de données lors du chiffrement d'une chaîne. Il combine le code de la step-by-step procédure dans un code de test que vous pouvez exécuter.

L'exemple crée un cache local et un fournisseur de clés principales ou un trousseau de clés pour unAWS KMS key. Il utilise ensuite le cache local et le fournisseur de clés principales ou le trousseau de clés pour créer un CMM de mise en cache avec des seuils de sécurité appropriés. En Java et Python, la demande de chiffrement spécifie le CMM de mise en cache, les données en texte brut à chiffrer et un contexte de chiffrement. En C, le CMM de mise en cache est spécifié dans la session et la session est fournie à la demande de chiffrement.

Pour exécuter ces exemples, vous devez fournir le nom de ressource Amazon (ARN) d'un AWS KMS key. Assurez-vous que vous êtes autorisé à utiliser la AWS KMS key pour générer une clé de données.

Pour des exemples concrets plus détaillés de création et d'utilisation d'un cache de clé de données, voirExemple de code de mise en cache des clés de données.

C

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License"). You may not use
 * this file except in compliance with the License. A copy of the License is
 * located at
 *
 *     http://aws.amazon.com/apache2.0/
 *
 * or in the "license" file accompanying this file. This file is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
```

```
 * implied. See the License for the specific language governing permissions and
 * limitations under the License.
 */

#include <aws/cryptosdk/cache.h>
#include <aws/cryptosdk/cpp/kms_keyring.h>
#include <aws/cryptosdk/session.h>

void encrypt_with_caching(
    uint8_t *ciphertext,     // output will go here (assumes ciphertext_capacity
 bytes already allocated)
    size_t *ciphertext_len,  // length of output will go here
    size_t ciphertext_capacity,
    const char *kms_key_arn,
    int max_entry_age,
    int cache_capacity) {
    const uint64_t MAX_ENTRY_MSGS = 100;

    struct aws_allocator *allocator = aws_default_allocator();

    // Load error strings for debugging
    aws_cryptosdk_load_error_strings();

    // Create a keyring
    struct aws_cryptosdk_keyring *kms_keyring =
 Aws::Cryptosdk::KmsKeyring::Builder().Build(kms_key_arn);

    // Create a cache
    struct aws_cryptosdk_materials_cache *cache =
 aws_cryptosdk_materials_cache_local_new(allocator, cache_capacity);

    // Create a caching CMM
    struct aws_cryptosdk_cmm *caching_cmm =
 aws_cryptosdk_caching_cmm_new_from_keyring(
        allocator, cache, kms_keyring, NULL, max_entry_age, AWS_TIMESTAMP_SECS);
    if (!caching_cmm) abort();

    if (aws_cryptosdk_caching_cmm_set_limit_messages(caching_cmm, MAX_ENTRY_MSGS))
 abort();

    // Create a session
    struct aws_cryptosdk_session *session =
        aws_cryptosdk_session_new_from_cmm_2(allocator, AWS_CRYPTOSDK_ENCRYPT,
 caching_cmm);
```

```
    if (!session) abort();

    // Encryption context
    struct aws_hash_table *enc_ctx =
 aws_cryptosdk_session_get_enc_ctx_ptr_mut(session);
    if (!enc_ctx) abort();
    AWS_STATIC_STRING_FROM_LITERAL(enc_ctx_key, "purpose");
    AWS_STATIC_STRING_FROM_LITERAL(enc_ctx_value, "test");
    if (aws_hash_table_put(enc_ctx, enc_ctx_key, (void *)enc_ctx_value, NULL))
 abort();

    // Plaintext data to be encrypted
    const char *my_data = "My plaintext data";
    size_t my_data_len  = strlen(my_data);
    if (aws_cryptosdk_session_set_message_size(session, my_data_len)) abort();

    // When the session uses a caching CMM, the encryption operation uses the data
 key cache
    // specified in the caching CMM.
    size_t bytes_read;
    if (aws_cryptosdk_session_process(
            session,
            ciphertext,
            ciphertext_capacity,
            ciphertext_len,
            (const uint8_t *)my_data,
            my_data_len,
            &bytes_read))
        abort();
    if (!aws_cryptosdk_session_is_done(session) || bytes_read != my_data_len)
 abort();

    aws_cryptosdk_session_destroy(session);
    aws_cryptosdk_cmm_release(caching_cmm);
    aws_cryptosdk_materials_cache_release(cache);
    aws_cryptosdk_keyring_release(kms_keyring);
}
```

Java

L'exemple suivant utilise la version 2. x duKit SDK de chiffrement AWS pour Java. Version 3. x du rend Kit SDK de chiffrement AWS pour Java obsolète le CMM de mise en cache des clés de

données. Avec la version 3. x, vous pouvez également utiliser le trousseau de clés AWS KMS hiérarchique, une solution alternative de mise en cache des matériaux cryptographiques.

```
// Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

package com.amazonaws.crypto.examples;

import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CryptoMaterialsManager;
import com.amazonaws.encryptionsdk.MasterKeyProvider;
import com.amazonaws.encryptionsdk.caching.CachingCryptoMaterialsManager;
import com.amazonaws.encryptionsdk.caching.CryptoMaterialsCache;
import com.amazonaws.encryptionsdk.caching.LocalCryptoMaterialsCache;
import com.amazonaws.encryptionsdk.kmssdkv2.KmsMasterKey;
import com.amazonaws.encryptionsdk.kmssdkv2.KmsMasterKeyProvider;
import java.nio.charset.StandardCharsets;
import java.util.Collections;
import java.util.Map;
import java.util.concurrent.TimeUnit;

/**
 * <p>
 * Encrypts a string using an &KMS; key and data key caching
 *
 * <p>
 * Arguments:
 * <ol>
 * <li>KMS Key ARN: To find the Amazon Resource Name of your &KMS; key,
 *     see 'Find the key ID and ARN' at https://docs.aws.amazon.com/kms/latest/
developerguide/find-cmk-id-arn.html
 * <li>Max entry age: Maximum time (in seconds) that a cached entry can be used
 * <li>Cache capacity: Maximum number of entries in the cache
 * </ol>
 */
public class SimpleDataKeyCachingExample {

    /*
     * Security thresholds
     *   Max entry age is required.
     *   Max messages (and max bytes) per data key are optional
     */
    private static final int MAX_ENTRY_MSGS = 100;
```

```
    public static byte[] encryptWithCaching(String kmsKeyArn, int maxEntryAge, int
 cacheCapacity) {
        // Plaintext data to be encrypted
        byte[] myData = "My plaintext data".getBytes(StandardCharsets.UTF_8);

        // Encryption context
        // Most encrypted data should have an associated encryption context
        // to protect integrity. This sample uses placeholder values.
        // For more information see:
        // blogs.aws.amazon.com/security/post/Tx2LZ6WBJJANTNW/How-to-Protect-the-
Integrity-of-Your-Encrypted-Data-by-Using-AWS-Key-Management
        final Map<String, String> encryptionContext =
 Collections.singletonMap("purpose", "test");

        // Create a master key provider
        MasterKeyProvider<KmsMasterKey> keyProvider =
 KmsMasterKeyProvider.builder()
            .buildStrict(kmsKeyArn);

        // Create a cache
        CryptoMaterialsCache cache = new LocalCryptoMaterialsCache(cacheCapacity);

        // Create a caching CMM
        CryptoMaterialsManager cachingCmm =

 CachingCryptoMaterialsManager.newBuilder().withMasterKeyProvider(keyProvider)
                .withCache(cache)
                .withMaxAge(maxEntryAge, TimeUnit.SECONDS)
                .withMessageUseLimit(MAX_ENTRY_MSGS)
                .build();

        // When the call to encryptData specifies a caching CMM,
        // the encryption operation uses the data key cache
        final AwsCrypto encryptionSdk = AwsCrypto.standard();
        return encryptionSdk.encryptData(cachingCmm, myData,
 encryptionContext).getResult();
    }
}
```

JavaScript Browser

```
// Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0

/* This is a simple example of using a caching CMM with a KMS keyring
 * to encrypt and decrypt using the AWS Encryption SDK for Javascript in a browser.
 */

import {
  KmsKeyringBrowser,
  KMS,
  getClient,
  buildClient,
  CommitmentPolicy,
  WebCryptoCachingMaterialsManager,
  getLocalCryptographicMaterialsCache,
} from '@aws-crypto/client-browser'
import { toBase64 } from '@aws-sdk/util-base64-browser'

/* This builds the client with the REQUIRE_ENCRYPT_REQUIRE_DECRYPT commitment
 policy,
 * which enforces that this client only encrypts using committing algorithm suites
 * and enforces that this client
 * will only decrypt encrypted messages
 * that were created with a committing algorithm suite.
 * This is the default commitment policy
 * if you build the client with `buildClient()`.
 */
const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

/* This is injected by webpack.
 * The webpack.DefinePlugin or @aws-sdk/karma-credential-loader will replace the
 values when bundling.
 * The credential values are pulled from @aws-sdk/credential-provider-node
 * Use any method you like to get credentials into the browser.
 * See kms.webpack.config
 */
declare const credentials: {
  accessKeyId: string
  secretAccessKey: string
  sessionToken: string
}

/* This is done to facilitate testing. */
```

```
export async function testCachingCMMExample() {
  /* This example uses an &KMS; keyring. The generator key in a &KMS; keyring
 generates and encrypts the data key.
   * The caller needs kms:GenerateDataKey permission on the &KMS; key in
 generatorKeyId.
   */
  const generatorKeyId =
    'arn:aws:kms:us-west-2:658956600833:alias/EncryptDecrypt'

  /* Adding additional KMS keys that can decrypt.
   * The caller must have kms:Encrypt permission for every &KMS; key in keyIds.
   * You might list several keys in different AWS Regions.
   * This allows you to decrypt the data in any of the represented Regions.
   * In this example, the generator key
   * and the additional key are actually the same &KMS; key.
   * In `generatorId`, this &KMS; key is identified by its alias ARN.
   * In `keyIds`, this &KMS; key is identified by its key ARN.
   * In practice, you would specify different &KMS; keys,
   * or omit the `keyIds` parameter.
   * This is *only* to demonstrate how the &KMS; key ARNs are configured.
   */
  const keyIds = [
    'arn:aws:kms:us-west-2:658956600833:key/b3537ef1-d8dc-4780-9f5a-55776cbb2f7f',
  ]

  /* Need a client provider that will inject correct credentials.
   * The credentials here are injected by webpack from your environment bundle is
 created
   * The credential values are pulled using @aws-sdk/credential-provider-node.
   * See kms.webpack.config
   * You should inject your credential into the browser in a secure manner
   * that works with your application.
   */
  const { accessKeyId, secretAccessKey, sessionToken } = credentials

  /* getClient takes a KMS client constructor
   * and optional configuration values.
   * The credentials can be injected here,
   * because browsers do not have a standard credential discovery process the way
 Node.js does.
   */
  const clientProvider = getClient(KMS, {
    credentials: {
      accessKeyId,
```

```
      secretAccessKey,
      sessionToken,
    },
  })

  /* You must configure the KMS keyring with your &KMS; keys */
  const keyring = new KmsKeyringBrowser({
    clientProvider,
    generatorKeyId,
    keyIds,
  })

  /* Create a cache to hold the data keys (and related cryptographic material).
   * This example uses the local cache provided by the Encryption SDK.
   * The `capacity` value represents the maximum number of entries
   * that the cache can hold.
   * To make room for an additional entry,
   * the cache evicts the oldest cached entry.
   * Both encrypt and decrypt requests count independently towards this threshold.
   * Entries that exceed any cache threshold are actively removed from the cache.
   * By default, the SDK checks one item in the cache every 60 seconds (60,000
milliseconds).
   * To change this frequency, pass in a `proactiveFrequency` value
   * as the second parameter. This value is in milliseconds.
   */
  const capacity = 100
  const cache = getLocalCryptographicMaterialsCache(capacity)

  /* The partition name lets multiple caching CMMs share the same local
cryptographic cache.
   * By default, the entries for each CMM are cached separately. However, if you
want these CMMs to share the cache,
   * use the same partition name for both caching CMMs.
   * If you don't supply a partition name, the Encryption SDK generates a random
name for each caching CMM.
   * As a result, sharing elements in the cache MUST be an intentional operation.
   */
  const partition = 'local partition name'

  /* maxAge is the time in milliseconds that an entry will be cached.
   * Elements are actively removed from the cache.
   */
  const maxAge = 1000 * 60
```

```
  /* The maximum number of bytes that will be encrypted under a single data key.
   * This value is optional,
   * but you should configure the lowest practical value.
   */
  const maxBytesEncrypted = 100

  /* The maximum number of messages that will be encrypted under a single data key.
   * This value is optional,
   * but you should configure the lowest practical value.
   */
  const maxMessagesEncrypted = 10

  const cachingCMM = new WebCryptoCachingMaterialsManager({
    backingMaterials: keyring,
    cache,
    partition,
    maxAge,
    maxBytesEncrypted,
    maxMessagesEncrypted,
  })

  /* Encryption context is a *very* powerful tool for controlling
   * and managing access.
   * When you pass an encryption context to the encrypt function,
   * the encryption context is cryptographically bound to the ciphertext.
   * If you don't pass in the same encryption context when decrypting,
   * the decrypt function fails.
   * The encryption context is ***not*** secret!
   * Encrypted data is opaque.
   * You can use an encryption context to assert things about the encrypted data.
   * The encryption context helps you to determine
   * whether the ciphertext you retrieved is the ciphertext you expect to decrypt.
   * For example, if you are are only expecting data from 'us-west-2',
   * the appearance of a different AWS Region in the encryption context can indicate
 malicious interference.
   * See: https://docs.aws.amazon.com/encryption-sdk/latest/developer-guide/
concepts.html#encryption-context
   *
   * Also, cached data keys are reused ***only*** when the encryption contexts
 passed into the functions are an exact case-sensitive match.
   * See: https://docs.aws.amazon.com/encryption-sdk/latest/developer-guide/data-
caching-details.html#caching-encryption-context
   */
  const encryptionContext = {
```

```
    stage: 'demo',
    purpose: 'simple demonstration app',
    origin: 'us-west-2',
  }

  /* Find data to encrypt. */
  const plainText = new Uint8Array([1, 2, 3, 4, 5])

  /* Encrypt the data.
   * The caching CMM only reuses data keys
   * when it know the length (or an estimate) of the plaintext.
   * However, in the browser,
   * you must provide all of the plaintext to the encrypt function.
   * Therefore, the encrypt function in the browser knows the length of the
plaintext
   * and does not accept a plaintextLength option.
   */
  const { result } = await encrypt(cachingCMM, plainText, { encryptionContext })

  /* Log the plain text
   * only for testing and to show that it works.
   */
  console.log('plainText:', plainText)
  document.write('</br>plainText:' + plainText + '</br>')

  /* Log the base64-encoded result
   * so that you can try decrypting it with another AWS Encryption SDK
implementation.
   */
  const resultBase64 = toBase64(result)
  console.log(resultBase64)
  document.write(resultBase64)

  /* Decrypt the data.
   * NOTE: This decrypt request will not use the data key
   * that was cached during the encrypt operation.
   * Data keys for encrypt and decrypt operations are cached separately.
   */
  const { plaintext, messageHeader } = await decrypt(cachingCMM, result)

  /* Grab the encryption context so you can verify it. */
  const { encryptionContext: decryptedContext } = messageHeader

  /* Verify the encryption context.
```

```
   * If you use an algorithm suite with signing,
   * the Encryption SDK adds a name-value pair to the encryption context that
 contains the public key.
   * Because the encryption context might contain additional key-value pairs,
   * do not include a test that requires that all key-value pairs match.
   * Instead, verify that the key-value pairs that you supplied to the `encrypt`
 function are included in the encryption context that the `decrypt` function
 returns.
   */
  Object.entries(encryptionContext).forEach(([key, value]) => {
    if (decryptedContext[key] !== value)
      throw new Error('Encryption Context does not match expected values')
  })

  /* Log the clear message
   * only for testing and to show that it works.
   */
  document.write('</br>Decrypted:' + plaintext)
  console.log(plaintext)

  /* Return the values to make testing easy. */
  return { plainText, plaintext }
}
```

JavaScript Node.js

```
// Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  KmsKeyringNode,
  buildClient,
  CommitmentPolicy,
  NodeCachingMaterialsManager,
  getLocalCryptographicMaterialsCache,
} from '@aws-crypto/client-node'

/* This builds the client with the REQUIRE_ENCRYPT_REQUIRE_DECRYPT commitment
 policy,
 * which enforces that this client only encrypts using committing algorithm suites
 * and enforces that this client
 * will only decrypt encrypted messages
 * that were created with a committing algorithm suite.
```

```
 * This is the default commitment policy
 * if you build the client with `buildClient()`.
 */
const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT
)

export async function cachingCMMNodeSimpleTest() {
  /* An &KMS; key is required to generate the data key.
   * You need kms:GenerateDataKey permission on the &KMS; key in generatorKeyId.
   */
  const generatorKeyId =
    'arn:aws:kms:us-west-2:658956600833:alias/EncryptDecrypt'

  /* Adding alternate &KMS; keys that can decrypt.
   * Access to kms:Encrypt is required for every &KMS; key in keyIds.
   * You might list several keys in different AWS Regions.
   * This allows you to decrypt the data in any of the represented Regions.
   * In this example, the generator key
   * and the additional key are actually the same &KMS; key.
   * In `generatorId`, this &KMS; key is identified by its alias ARN.
   * In `keyIds`, this &KMS; key is identified by its key ARN.
   * In practice, you would specify different &KMS; keys,
   * or omit the `keyIds` parameter.
   * This is *only* to demonstrate how the &KMS; key ARNs are configured.
   */
  const keyIds = [
    'arn:aws:kms:us-west-2:658956600833:key/b3537ef1-d8dc-4780-9f5a-55776cbb2f7f',
  ]

  /* The &KMS; keyring must be configured with the desired &KMS; keys
   * This example passes the keyring to the caching CMM
   * instead of using it directly.
   */
  const keyring = new KmsKeyringNode({ generatorKeyId, keyIds })

  /* Create a cache to hold the data keys (and related cryptographic material).
   * This example uses the local cache provided by the Encryption SDK.
   * The `capacity` value represents the maximum number of entries
   * that the cache can hold.
   * To make room for an additional entry,
   * the cache evicts the oldest cached entry.
   * Both encrypt and decrypt requests count independently towards this threshold.
   * Entries that exceed any cache threshold are actively removed from the cache.
```

```
  * By default, the SDK checks one item in the cache every 60 seconds (60,000
milliseconds).
  * To change this frequency, pass in a `proactiveFrequency` value
  * as the second parameter. This value is in milliseconds.
  */
 const capacity = 100
 const cache = getLocalCryptographicMaterialsCache(capacity)

 /* The partition name lets multiple caching CMMs share the same local
cryptographic cache.
  * By default, the entries for each CMM are cached separately. However, if you
want these CMMs to share the cache,
  * use the same partition name for both caching CMMs.
  * If you don't supply a partition name, the Encryption SDK generates a random
name for each caching CMM.
  * As a result, sharing elements in the cache MUST be an intentional operation.
  */
 const partition = 'local partition name'

 /* maxAge is the time in milliseconds that an entry will be cached.
  * Elements are actively removed from the cache.
  */
 const maxAge = 1000 * 60

 /* The maximum amount of bytes that will be encrypted under a single data key.
  * This value is optional,
  * but you should configure the lowest value possible.
  */
 const maxBytesEncrypted = 100

 /* The maximum number of messages that will be encrypted under a single data key.
  * This value is optional,
  * but you should configure the lowest value possible.
  */
 const maxMessagesEncrypted = 10

 const cachingCMM = new NodeCachingMaterialsManager({
   backingMaterials: keyring,
   cache,
   partition,
   maxAge,
   maxBytesEncrypted,
   maxMessagesEncrypted,
 })
```

```
  /* Encryption context is a *very* powerful tool for controlling
   * and managing access.
   * When you pass an encryption context to the encrypt function,
   * the encryption context is cryptographically bound to the ciphertext.
   * If you don't pass in the same encryption context when decrypting,
   * the decrypt function fails.
   * The encryption context is ***not*** secret!
   * Encrypted data is opaque.
   * You can use an encryption context to assert things about the encrypted data.
   * The encryption context helps you to determine
   * whether the ciphertext you retrieved is the ciphertext you expect to decrypt.
   * For example, if you are are only expecting data from 'us-west-2',
   * the appearance of a different AWS Region in the encryption context can indicate
 malicious interference.
   * See: https://docs.aws.amazon.com/encryption-sdk/latest/developer-guide/
concepts.html#encryption-context
   *
   * Also, cached data keys are reused ***only*** when the encryption contexts
 passed into the functions are an exact case-sensitive match.
   * See: https://docs.aws.amazon.com/encryption-sdk/latest/developer-guide/data-
caching-details.html#caching-encryption-context
   */
  const encryptionContext = {
    stage: 'demo',
    purpose: 'simple demonstration app',
    origin: 'us-west-2',
  }

  /* Find data to encrypt.  A simple string. */
  const cleartext = 'asdf'

  /* Encrypt the data.
   * The caching CMM only reuses data keys
   * when it know the length (or an estimate) of the plaintext.
   * If you do not know the length,
   * because the data is a stream
   * provide an estimate of the largest expected value.
   *
   * If your estimate is smaller than the actual plaintext length
   * the AWS Encryption SDK will throw an exception.
   *
   * If the plaintext is not a stream,
   * the AWS Encryption SDK uses the actual plaintext length
```

```
   * instead of any length you provide.
   */
  const { result } = await encrypt(cachingCMM, cleartext, {
    encryptionContext,
    plaintextLength: 4,
  })

  /* Decrypt the data.
   * NOTE: This decrypt request will not use the data key
   * that was cached during the encrypt operation.
   * Data keys for encrypt and decrypt operations are cached separately.
   */
  const { plaintext, messageHeader } = await decrypt(cachingCMM, result)

  /* Grab the encryption context so you can verify it. */
  const { encryptionContext: decryptedContext } = messageHeader

  /* Verify the encryption context.
   * If you use an algorithm suite with signing,
   * the Encryption SDK adds a name-value pair to the encryption context that
 contains the public key.
   * Because the encryption context might contain additional key-value pairs,
   * do not include a test that requires that all key-value pairs match.
   * Instead, verify that the key-value pairs that you supplied to the `encrypt`
 function are included in the encryption context that the `decrypt` function
 returns.
   */
  Object.entries(encryptionContext).forEach(([key, value]) => {
    if (decryptedContext[key] !== value)
      throw new Error('Encryption Context does not match expected values')
  })

  /* Return the values so the code can be tested. */
  return { plaintext, result, cleartext, messageHeader }
}
```

Python

```
# Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You
# may not use this file except in compliance with the License. A copy of
# the License is located at
```

```
#
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is
# distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF
# ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.
"""Example of encryption with data key caching."""
import aws_encryption_sdk
from aws_encryption_sdk import CommitmentPolicy


def encrypt_with_caching(kms_key_arn, max_age_in_cache, cache_capacity):
    """Encrypts a string using an &KMS; key and data key caching.

    :param str kms_key_arn: Amazon Resource Name (ARN) of the &KMS; key
    :param float max_age_in_cache: Maximum time in seconds that a cached entry can
 be used
    :param int cache_capacity: Maximum number of entries to retain in cache at once
    """
    # Data to be encrypted
    my_data = "My plaintext data"

    # Security thresholds
    #   Max messages (or max bytes per) data key are optional
    MAX_ENTRY_MESSAGES = 100

    # Create an encryption context
    encryption_context = {"purpose": "test"}

    # Set up an encryption client with an explicit commitment policy. Note that if
 you do not explicitly choose a
    # commitment policy, REQUIRE_ENCRYPT_REQUIRE_DECRYPT is used by default.
    client =
 aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_R

    # Create a master key provider for the &KMS; key
    key_provider =
 aws_encryption_sdk.StrictAwsKmsMasterKeyProvider(key_ids=[kms_key_arn])

    # Create a local cache
    cache = aws_encryption_sdk.LocalCryptoMaterialsCache(cache_capacity)

    # Create a caching CMM
```

```
    caching_cmm = aws_encryption_sdk.CachingCryptoMaterialsManager(
        master_key_provider=key_provider,
        cache=cache,
        max_age=max_age_in_cache,
        max_messages_encrypted=MAX_ENTRY_MESSAGES,
    )

    # When the call to encrypt data specifies a caching CMM,
    # the encryption operation uses the data key cache specified
    # in the caching CMM
    encrypted_message, _header = client.encrypt(
        source=my_data, materials_manager=caching_cmm,
encryption_context=encryption_context
    )

    return encrypted_message
```

Définition des seuils de sécurité du cache

Lorsque vous implémentez la mise en cache des clés de données, vous devez configurer les seuils de sécurité appliqués par le kitmise en cache de CMMapplique.

Les seuils de sécurité vous aident à limiter la durée d'utilisation de chaque clé de données mise en cache et la quantité de données protégée par chaque clé de données. Le CMM de mise en cache renvoie des clés de données mises en cache uniquement lorsque l'entrée de cache respecte tous les seuils de sécurité. Si l'entrée de cache dépasse un seuil, elle n'est pas utilisée pour l'opération actuelle et elle est expulsée du cache dès que possible. La première utilisation de chaque clé de données (avant la mise en cache) n'est pas comptabilisée dans ces seuils.

En règle générale, utilisez le volume de cache minimum nécessaire pour atteindre vos objectifs de coûts et de performance.

Le kit AWS Encryption SDK met uniquement en cache les clés de données qui sont chiffrées à l'aide d'une fonction de dérivation de clés. Il établit également des limites supérieures pour certaines valeurs de seuil. Ces restrictions garantissent que les clés de données ne sont pas réutilisées au-delà de leurs limites de chiffrement. Cependant, étant donné que vos clés de données en texte brut sont mises en cache (dans la mémoire, par défaut), essayez de réduire le temps d'enregistrement des clés. Essayez également de limiter les données qui pourraient être exposées si une clé est compromise.

Pour obtenir des exemples de définition de seuils de sécurité du cache, voirAWS Encryption SDK : Comment déterminer si la mise en cache des clés de données convient à votre applicationdans leAWSBlog sur la sécurité.

> **Note**
>
> Le CMM de mise en cache applique tous les seuils suivants. Si vous ne spécifiez pas de valeur facultative, le CMM de mise en cache utilise la valeur par défaut.
> Pour désactiver temporairement la mise en cache des clés de données, les implémentations Java et Python du kitAWS Encryption SDKFournissez uncache de matériaux de chiffrement nul(cache nul). Le cache nul renvoie un message d'échec pour chaque demande `GET` et ne répond pas aux demandes `PUT`. Nous vous recommandons d'utiliser le cache nul au lieu de la définir la capacité de cache ou les seuils de sécurité à 0. Pour plus d'informations, consultez le cache nul dans Java and Python.

Âge maximum (obligatoire)

Détermine la durée pendant laquelle une entrée mise en cache peut être utilisée, à partir du moment où elle a été ajoutée. Cette valeur est obligatoire. Saisissez une valeur supérieure à 0. Le kit AWS Encryption SDK ne limite pas la valeur d'âge maximum.

Toutes les implémentations de langage du kitAWS Encryption SDKdéfinir l'âge maximum en secondes, à l'exception duKit SDK de chiffrement AWS pour JavaScript, qui utilise des millisecondes.

Utilisez l'intervalle le plus court qui permet cependant à votre application de tirer parti du cache. Vous pouvez utiliser le seuil d'âge maximal comme une stratégie de rotation des clés. Utilisez-le pour limiter la réutilisation des clés de données, minimiser l'exposition des matériaux de chiffrement et expulser les clés de données dont les stratégies ont pu être modifiées pendant qu'elles étaient mises en cache.

Nombre maximal de messages chiffrés (facultatif)

Spécifie le nombre maximal de messages qu'une clé de données mise en cache peut chiffrer. Cette valeur est facultative. Saisissez une valeur comprise entre 1 et 2^32 messages. La valeur par défaut est 2^32 messages.

Définissez le nombre de messages protégés par chaque clé mise en cache de sorte qu'il soit suffisamment grand pour obtenir des valeurs par la réutilisation, mais suffisamment petit pour limiter le nombre de messages pouvant être exposés si une clé est compromise.

Nombre maximal d'octets chiffrés (facultatif)

Spécifie le nombre maximal d'octets qu'une clé de données mise en cache peut chiffrer. Cette valeur est facultative. Saisissez une valeur comprise entre 0 et 2^63 - 1. La valeur par défaut est 2^63 - 1. Une valeur de 0 vous permet d'utiliser la mise en cache des clés de données uniquement lorsque vous chiffrez des chaînes de message vides.

Les octets de la demande en cours sont inclus lors de l'évaluation de ce seuil. Si le nombre d'octets traités plus le nombre d'octets en cours dépassent le seuil, la clé de données mise en cache est expulsée du cache, même si elle aurait pu être utilisée pour une demande plus petite.

Détails de la mise en cache des clés de données

La plupart des applications peuvent utiliser l'implémentation par défaut de la mise en cache des clés de données sans écrire de code personnalisé. Cette section décrit l'implémentation par défaut, ainsi que certaines informations sur les options.

Rubriques

Fonctionnement de la mise en cache des clés de données

Lorsque vous utilisez la mise en cache des clés de données dans une demande afin de chiffrer ou déchiffrer des données, le kit AWS Encryption SDK cherche d'abord dans le cache une clé de données correspondant à la demande. Si le kit trouve une correspondance valide, il utilise la clé de données mise en cache pour chiffrer les données. Sinon, il génère une nouvelle clé de données, comme il le ferait en l'absence de cache.

La mise en cache des clés de données n'est pas utilisée pour les données de taille inconnue, par exemple les données diffusées. Cela permet au CMM de mise en cache d'appliquer correctement leseuil maximal d'octets. Pour éviter ce comportement, ajoutez la taille du message à la demande de chiffrement.

En plus d'un cache, la mise en cache des clés de données utilise unGestionnaire de matériaux de chiffrement de mise en cache(mise en cache CMM). Le CMM de mise en cache est un outil spécialiséGestionnaire de matériaux de chiffrement (CMM)qui interagit avec uncacheet un sous-jacentCMM. (Lorsque vous spécifiez unFournisseur de clés principalesou porte-clés, leAWS Encryption SDKcrée une MMT par défaut pour vous.) Le CMM de mise en cache les clés de données renvoyées par sa CMM sous-jacente. Le CMM de mise en cache applique également les seuils de sécurité du cache que vous définissez.

Pour éviter qu'une clé de données incorrecte soit sélectionnée à partir du cache, chaque CMM de mise en cache compatible nécessite que les propriétés suivantes des matériaux cryptographiques de mise en cache correspondent à la demande de matériaux.

- Suite d'algorithmes
- Contexte de chiffrement (même lorsqu'il est vide)
- Nom de partition (une chaîne qui identifie le CMM de mise en cache)
- (Déchiffrement uniquement) Clés de données de chiffrement

> **Note**
>
> Le kit AWS Encryption SDK met en cache les clés de données uniquement lorsque la suite d'algorithmes utilise une fonction de dérivation de clés.

Les flux de travail suivants illustrent le traitement d'une demande de chiffrement de données avec et sans mise en cache des clés de données. Ils expliquent l'utilisation des composants de mise en cache que vous créez, y compris le cache et le CMM de mise en cache, pendant le processus.

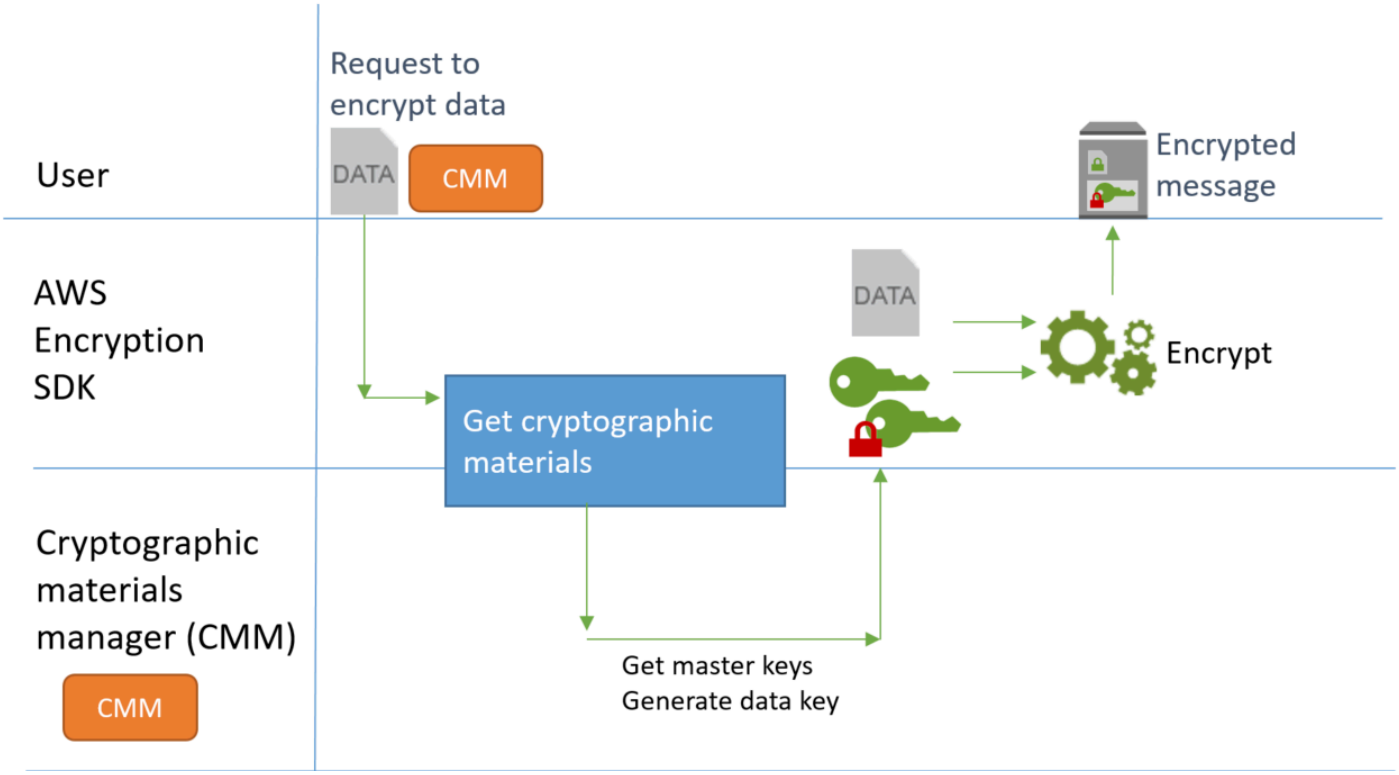
Chiffrement de données sans mise en cache

Pour obtenir les matériaux de chiffrement sans mise en cache :

1. Une application demande au kit AWS Encryption SDK de chiffrer des données.

La requête spécifie un fournisseur de clés principales ou un porte-clés. LeAWS Encryption SDKcrée un CMM par défaut qui interagit avec votre fournisseur de clés principales ou votre porte-clés.

2. LeAWS Encryption SDKLe demande au CMM les matériaux de chiffrement (obtenir les matériaux de chiffrement).
3. Le CMM demande sonPorte-clés(C) etJavaScript) ouFournisseur de clés principales(Java et Python) pour les matériaux de chiffrement. Pour cela, il peut être nécessaire d'appeler un service de chiffrement, comme AWS Key Management Service (AWS KMS). Le CMM retourne les matériaux de chiffrement au kitAWS Encryption SDK.
4. Le kit AWS Encryption SDK utilise la clé de données en texte brut pour chiffrer les données. Il stocke les données chiffrées et des clés de données chiffrées dans un message chiffré, qu'il renvoie à l'utilisateur.



Chiffrement de données avec mise en cache

Pour obtenir les matériaux de chiffrement sans mise en cache de clé de données :

1. Une application demande au kit AWS Encryption SDK de chiffrer des données.

La requête spécifie unMise en cache du gestionnaire de matériaux de chiffrement de mise en cache (mise en cache CMM)qui est associé à un gestionnaire de matériaux de chiffrement (CMM) sous-jacent. Lorsque vous spécifiez un fournisseur de clés principales ou un porte-clés, le kitAWS Encryption SDKcrée une MMT par défaut pour vous.

2. Le kit SDK demande les matériaux de chiffrement au CMM de mise en cache spécifié.
3. Le CMM de mise en cache demande les matériaux de chiffrement au cache.

   a. Si le cache trouve une correspondance, il met à jour l'âge et utilise les valeurs de l'entrée de cache correspondante, et renvoie les matériaux de chiffrement mis en cache au CMM de mise en cache.

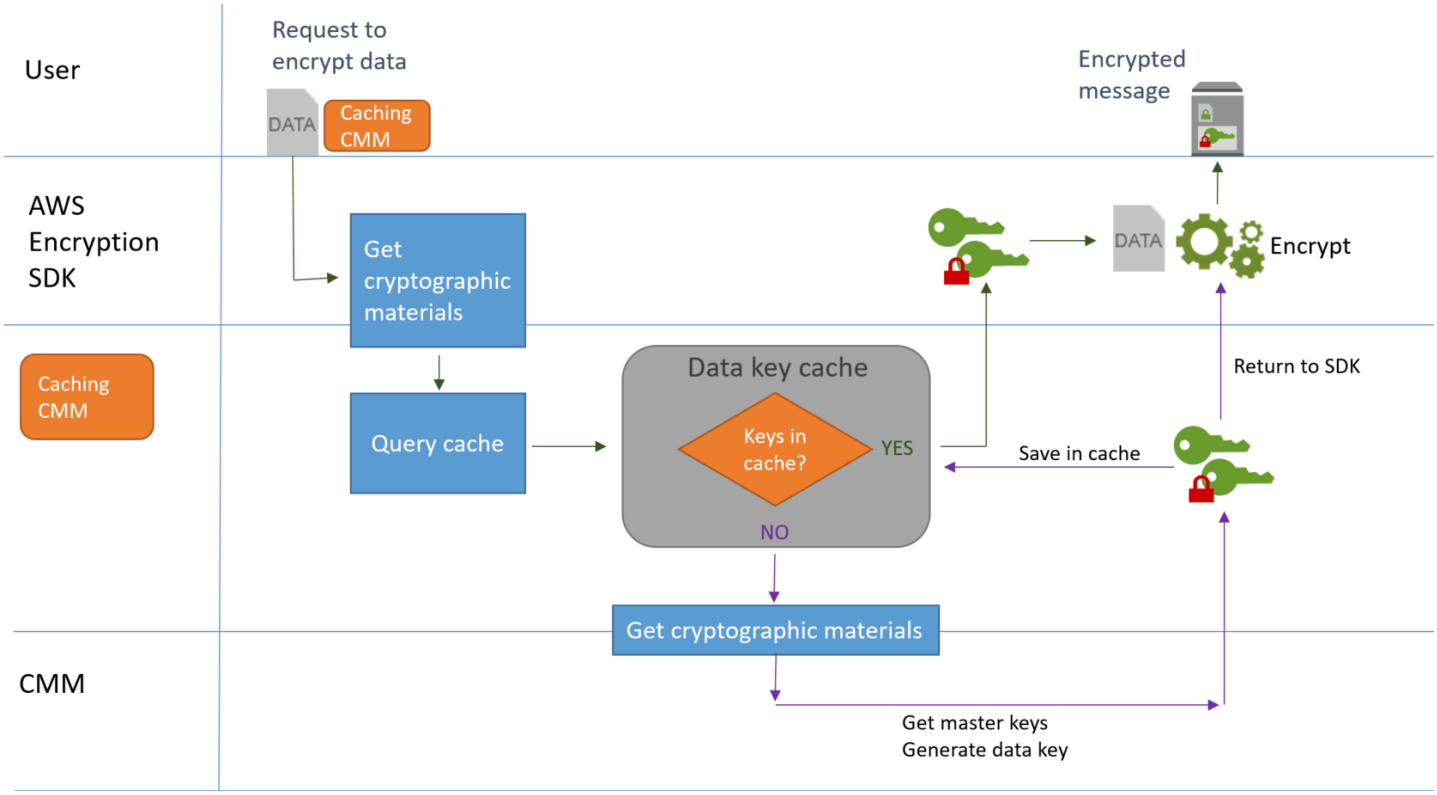
      Si l'entrée de cache est conforme à sonseuils de sécurité, le CMM de mise en cache le renvoie au kit SDK. Dans le cas contraire, il indique au cache d'expulser l'entrée et poursuit comme s'il n'y avait pas de correspondance.

   b. Si le cache ne trouve pas de correspondance valide, le CMM de mise en cache demande au CMM sous-jacent de générer une nouvelle clé de données.

      Le CMM sous-jacent obtient les matériaux de chiffrement à partir de son porte-clés (C etJavaScript) ou du fournisseur de clé principale (Java et Python). Pour cela, il peut être nécessaire d'appeler un service, comme AWS Key Management Service. Le CMM sous-jacent renvoie les copies en texte brut et chiffrées de la clé de données au CMM de mise en cache.

      Le CMM de mise en cache enregistre les nouveaux matériaux de chiffrement dans le cache.

4. Le CMM de mise en cache renvoie les matériaux de chiffrement au kitAWS Encryption SDK.
5. Le kit AWS Encryption SDK utilise la clé de données en texte brut pour chiffrer les données. Il stocke les données chiffrées et des clés de données chiffrées dans un message chiffré, qu'il renvoie à l'utilisateur.



Création d'un cache de matériaux de chiffrement

Le kit AWS Encryption SDK définit les exigences concernant un cache de matériaux de chiffrement utilisé pour la mise en cache de clés de données. Il fournit également un cache local, qui est un configurable, en mémoire,Cache LRU (LRU) le moins récemment utilisé. Pour créer une instance du cache local, utilisez le`LocalCryptoMaterialsCache`Constructeur Java et Python, legetLocalCryptographicMaterialsCachefonction dansJavaScript, ou le`aws_cryptosdk_materials_cache_local_new`Constructor en C.

Le cache local inclut une logique pour la gestion de cache de base, y compris l'ajout, l'expulsion et la correspondance des entrées mises en cache, ainsi que la gestion du cache. Vous n'avez pas besoin d'écrire une logique de gestion de cache personnalisée. Vous pouvez utiliser le cache local tel quel, le personnaliser ou le remplacer par un cache compatible.

Lorsque vous créez un cache local, vous définissez soncapacité, c'est-à-dire le nombre maximal d'entrées que le cache peut contenir. Ce paramètre vous permet de concevoir un cache efficace avec une réutilisation des clés de données limitée.

LeKit SDK de chiffrement AWS pour Javaet l'Kit SDK de chiffrement AWS pour Pythonfournir également unCache de matériaux de chiffrement null(NullCryptoMaterialsCache). Le

NullCryptoMaterialsCache renvoie un message d'échec pour toutes les opérations GET et ne répond pas aux opérations PUT. Vous pouvez utiliser NullCryptoMaterialsCache pour les tests ou pour désactiver temporairement la mise en cache dans une application qui inclut un code de mise en cache.

DansAWS Encryption SDK, chaque cache de matériaux de chiffrement est associé à unGestionnaire de matériaux de chiffrement de mise en cache(mise en cache CMM). Le CMM de mise en cache récupère des clés de données du cache, stocke les données des clés dans le cache, et applique les applicationsseuils de sécuritéque vous avez fixé. Lorsque vous créez un CMM de mise en cache, vous spécifiez le cache qu'il utilise et le CMM ou le fournisseur de clés principales sous-jacent qui génère les clés de données qu'il met en cache.

Création d'un gestionnaire des matériaux de chiffrement de mise en cache

Pour activer la mise en cache des clés de données, vous créez uncacheet unGestionnaire de matériaux de chiffrement de mise en cache(mise en cache CMM). Ensuite, dans vos demandes de chiffrement ou de déchiffrement des données, vous spécifiez un CMM de mise en cache au lieu d'un standard.Gestionnaire de matériaux de chiffrement (CMM), ouFournisseur de clés principalesouPorte-clés.

Il existe deux types de GMT. Les deux récupèrent des clés de données (et les matériaux de chiffrement associés), mais de différentes façons, comme indiqué ci-dessous :

- Un CMM est associé à un porte-clés (C ouJavaScript) ou un fournisseur de clé principale (Java et Python). Lorsque le kit SDK demande au CMM les matériaux de chiffrement ou de déchiffrement, le CMM obtient les matériaux à partir de son porte-clés ou du fournisseur de clé principale. Dans Java et Python, le CMM utilise ensuite les clés principales pour générer, chiffrer ou déchiffrer les clés de données. En C et JavaScript, le porte-clés génère, chiffre et renvoie les matériaux de chiffrement.
- Un CMM de mise en cache est associé à un cache, tel qu'unCache local, et un CMM sous-jacent. Lorsque le kit SDK demande des matériaux de chiffrement au CMM de mise en cache, le CMM de mise en cache tente de les récupérer à partir du cache. S'il ne trouve aucune correspondance, le CMM de mise en cache demande les matériaux à son CMM sous-jacent. Il met ensuite en cache les nouveaux matériaux cryptographiques avant de les renvoyer au mandataire.

Le CMM de mise en cache applique égalementseuils de sécuritéque vous définissez pour chaque entrée de cache. Etant donné que les seuils de sécurité sont définis et appliqués par le CMM de mise

en cache, vous pouvez utiliser n'importe quel cache compatible, même si le cache n'est pas conçu pour les matériaux sensibles.

Qu'est-ce qu'une entrée de cache de clé de données ?

La mise en cache des clés de données stocke les clés de données et les matériaux de chiffrement connexes dans un cache. Chaque entrée comprend les éléments répertoriés ci-dessous. Ces informations peuvent vous être utiles lorsque vous décidez si vous souhaitez utiliser la fonction de mise en cache des clés de données, et lorsque vous définissez les seuils de sécurité pour un gestionnaire de matériaux cryptographiques de mise en cache (CMM de mise en cache).

Entrées mises en cache pour les demandes de chiffrement

Les entrées qui sont ajoutées à un cache de clé de données suite à une opération de chiffrement incluent les éléments suivants :

- Clé de données en texte brut
- Clés de données chiffrées (une ou plusieurs)
- Contexte de chiffrement
- Clé de signature de message (le cas échéant)
- Suite d'algorithmes
- Métadonnées, y compris les compteurs d'utilisation pour appliquer les seuils de sécurité

Entrées mises en cache pour les demandes de déchiffrement

Les entrées qui sont ajoutées à un cache de clé de données suite à une opération de déchiffrement incluent les éléments suivants :

- Clé de données en texte brut
- Clé de vérification de signature (le cas échéant)
- Métadonnées, y compris les compteurs d'utilisation pour appliquer les seuils de sécurité

Contexte de chiffrement : Sélection des entrées de cache

Vous pouvez spécifier un contexte de chiffrement dans toute demande afin de chiffrer des données. Toutefois, le contexte de chiffrement joue un rôle particulier dans la mise en cache des clés de

données. Il vous permet de créer des sous-groupes de clés de données dans votre cache, même lorsque les clés de données proviennent du même CMM de mise en cache.

Un contexte de chiffrement est un ensemble de paires clé-valeur contenant les données non secrètes arbitraires. Pendant le chiffrement, le contexte de chiffrement est lié de façon chiffrée aux données chiffrées. Le même contexte de chiffrement est donc requis pour déchiffrer les données. Dans le kit AWS Encryption SDK, le contexte de chiffrement est stocké dans le message chiffré, ainsi que les données chiffrées et les clés de données.

Lorsque vous utilisez un cache de clé de données, vous pouvez également utiliser le contexte de chiffrement pour sélectionner des clés de données mises en cache particulières pour vos opérations de chiffrement. Le contexte de chiffrement est enregistré dans l'entrée de cache avec la clé de données (il fait partie de l'ID de l'entrée de cache). Les clés de données mises en cache sont réutilisées uniquement en cas de correspondance de leurs contextes de chiffrement. Si vous souhaitez réutiliser certaines clés de données pour une demande de chiffrement, spécifiez le même contexte de chiffrement. Si vous souhaitez éviter ces clés de données, spécifiez un autre contexte de chiffrement.

Le contexte de chiffrement est toujours facultatif, mais il est recommandé. Si vous ne spécifiez pas de contexte de chiffrement dans votre demande, un contexte de chiffrement vide est inclus dans l'ID de l'entrée de cache et associé à chaque demande.

Mon application utilise-t-elle des clés de données mises en cache ?

La mise en cache des clés de données est une stratégie d'optimisation très efficace pour certaines applications et charges de travail. Cependant, comme cela comporte un certain risque, il est important de déterminer dans quelle mesure il est susceptible d'être efficace pour votre situation, puis de décider si les avantages l'emportent sur les risques.

Étant donné que la mise en cache des clés de données réutilise les clés de données, l'effet le plus évident est de réduire le nombre d'appels pour générer de nouvelles clés de données. Lorsque la mise en cache de clé de données est implémentée, le kit AWS Encryption SDK appelle l'opération AWS KMS `GenerateDataKey` uniquement pour créer la clé de données initiale et lorsque le cache manque. Mais la mise en cache améliore les performances de manière perceptible uniquement dans les applications qui génèrent de nombreuses clés de données avec les mêmes caractéristiques, y compris le même contexte de chiffrement et la même suite d'algorithmes.

Pour déterminer si votre implémentation du kit AWS Encryption SDK utilise réellement des clés de données du cache, essayez les techniques suivantes.

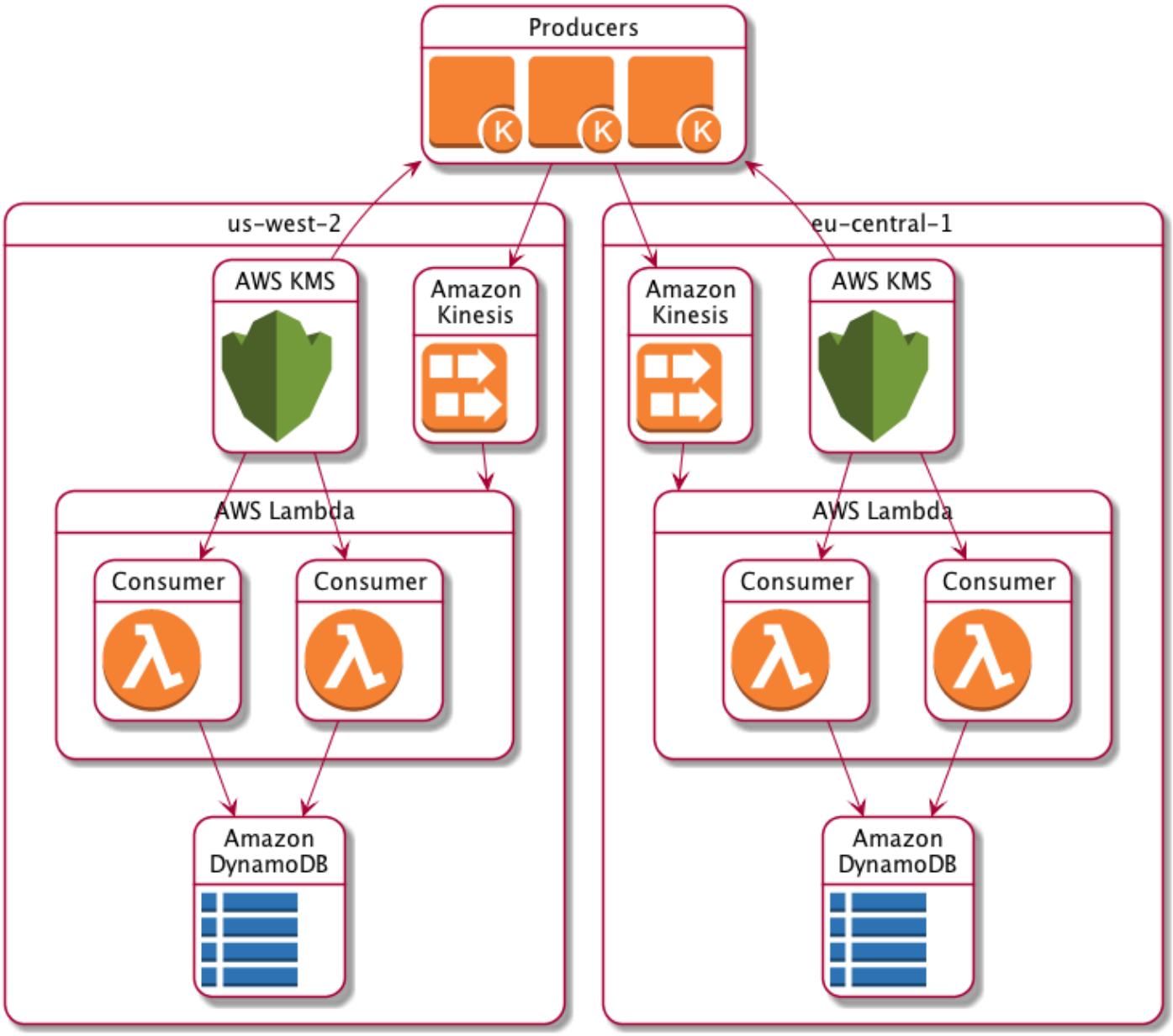
- Dans les journaux de votre infrastructure de clé principale, vérifiez la fréquence des appels pour créer de nouvelles clés de données. Lorsque la mise en cache des clés de données est efficace, le nombre d'appels pour créer de nouvelles clés doit diminuer sensiblement. Par exemple, si vous utilisez unAWS KMSfournisseur de clés principal ou porte-clés, recherchez dans leCloudTrailJournaux pourGenerateDataCléappels.
- Comparez les messages chiffrés renvoyés par AWS Encryption SDK en réponse à différentes demandes de chiffrement. Par exemple, si vous utilisez le kitKit SDK de chiffrement AWS pour Java, comparez lesParsedCiphertextobjet provenant de différents appels de chiffrement. DansKit SDK de chiffrement AWS pour JavaScript, comparez le contenu du kit`encryptedDataKeys`propriété duMessageHeader. Lorsque les clés de données sont réutilisées, les clés de données chiffrées dans le message chiffré sont identiques.

Exemple de mise en cache des clés de données

Cet exemple utiliseMise en cache des clés de donnéesavec unCache localpour accélérer une application dans laquelle les données générées par plusieurs périphériques sont chiffrées et stockées dans différentes régions.

Dans ce scénario, plusieurs producteurs de données génèrent des données, les chiffrent et écrivent dans unFlux Kinesisdans chaque région.AWS LambdaLes fonctions (consommateurs) déchiffrent les flux et écrivent des données en texte brut dans une table DynamoDB de la région. Les producteurs et consommateurs de données utilisent leAWS Encryption SDKet unAWS KMSFournisseur de clés principales. Pour limiter les appels à KMS, chaque producteur et consommateur dispose de son propre cache local.

Vous pouvez trouver le code source pour ces exemples dansJava et Python. L'exemple inclut également un modèle AWS CloudFormation qui définit les ressources pour les exemples.



Résultats du cache local

Le tableau suivant montre qu'un cache local réduit le nombre total des appels vers KMS (par seconde et par région) dans cet exemple à 1 % de sa valeur d'origine.

Demandes de producteurs

	Demandes par seconde et par client			Clients par région	Demandes moyennes
	Générer des clés de	Chiffrer des clés de	Total (par région)		

	données (us-west-2)	données (eu-central-1)			par seconde et par région
Aucun cache	1	1	1	500	500
Cache local	1 rps / 100 utilisations	1 rps / 100 utilisations	1 rps / 100 utilisations	500	5

Demandes de consommateurs

	Demandes par seconde et par client			Client par région	Demandes moyennes par seconde et par région
	Déchiffrer des clés de données	Producteurs	Total		
Aucun cache	1 rps par producteur	500	500	2	1 000
Cache local	1 rps par producteur / 100 utilisations	500	5	2	10

Exemple de code de mise en cache des clés de données

Cet exemple de code crée une implémentation simple de la mise en cache des clés de données avec un cache local en Java et Python. Le code crée deux instances d'un cache local : l'une pour les producteurs de données qui chiffrent les données et l'autre pour les consommateurs de données (AWS Lambdafonctions) qui les déchiffrent. Pour plus de détails sur l'implémentation de la mise en cache des clés de données dans chaque langage, consultez la documentation Javadoc et Python du. AWS Encryption SDK

La mise en cache des clés de données est disponible pour tous les langages de programmation pris en AWS Encryption SDK charge.

Pour obtenir des exemples complets et testés d'utilisation de la mise en cache de clés de données dans le AWS Encryption SDK, veuillez consulter :

- C/C++ : caching_cmm.cpp
- Java : SimpleDataKeyCachingExample.java
- JavaScript Navigateur : caching_cmm.ts
- JavaScript Node.js : caching_cmm.ts
- Python : data_key_caching_basic.py

Producer

Le producteur obtient une carte, la convertit en JSON, l'utilise AWS Encryption SDK pour la chiffrer et envoie l'enregistrement de texte chiffré vers un flux Kinesis dans chacune d'elles. Région AWS

Le code définit un gestionnaire de matériel cryptographique de mise en cache (CMM de mise en cache) et l'associe à un cache local et à un fournisseur de clé principale sous-jacentAWS KMS. Le CMM de mise en cache met en cache les clés de données (et le matériel cryptographique associé) provenant du fournisseur de clé principale. Il interagit également avec le cache au nom du kit SDK et applique les seuils de sécurité que vous définissez.

Étant donné que l'appel à la méthode de chiffrement spécifie un CMM de mise en cache, au lieu d'un gestionnaire de matériel cryptographique (CMM) ou d'un fournisseur de clé principale standard, le chiffrement utilisera la mise en cache des clés de données.

Java

L'exemple suivant utilise la version 2. x duKit SDK de chiffrement AWS pour Java. Version 3. x du rend Kit SDK de chiffrement AWS pour Java obsolète le CMM de mise en cache des clés de données. Avec la version 3. x, vous pouvez également utiliser le trousseau de clés AWS KMS hiérarchique, une solution alternative de mise en cache des matériaux cryptographiques.

```
/*
 * Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License"). You may not use
 this file except
 * in compliance with the License. A copy of the License is located at
 *
 * http://aws.amazon.com/apache2.0
```

```
 *
 * or in the "license" file accompanying this file. This file is distributed on an
 "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
 License for the
 * specific language governing permissions and limitations under the License.
 */
package com.amazonaws.crypto.examples.kinesisdatakeycaching;

import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CommitmentPolicy;
import com.amazonaws.encryptionsdk.CryptoResult;
import com.amazonaws.encryptionsdk.MasterKeyProvider;
import com.amazonaws.encryptionsdk.caching.CachingCryptoMaterialsManager;
import com.amazonaws.encryptionsdk.caching.LocalCryptoMaterialsCache;
import com.amazonaws.encryptionsdk.kmssdkv2.KmsMasterKey;
import com.amazonaws.encryptionsdk.kmssdkv2.KmsMasterKeyProvider;
import com.amazonaws.encryptionsdk.multi.MultipleProviderFactory;
import com.amazonaws.util.json.Jackson;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.concurrent.TimeUnit;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisClient;
import software.amazon.awssdk.services.kms.KmsClient;

/**
 * Pushes data to Kinesis Streams in multiple Regions.
 */
public class MultiRegionRecordPusher {

    private static final long MAX_ENTRY_AGE_MILLISECONDS = 300000;
    private static final long MAX_ENTRY_USES = 100;
    private static final int MAX_CACHE_ENTRIES = 100;
    private final String streamName_;
    private final ArrayList<KinesisClient> kinesisClients_;
    private final CachingCryptoMaterialsManager cachingMaterialsManager_;
    private final AwsCrypto crypto_;
```

```
   /**
    * Creates an instance of this object with Kinesis clients for all target
Regions and a cached
    * key provider containing KMS master keys in all target Regions.
    */
   public MultiRegionRecordPusher(final Region[] regions, final String
kmsAliasName,
       final String streamName) {
       streamName_ = streamName;
       crypto_ = AwsCrypto.builder()
           .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
           .build();
       kinesisClients_ = new ArrayList<>();

       AwsCredentialsProvider credentialsProvider =
DefaultCredentialsProvider.builder().build();

       // Build KmsMasterKey and AmazonKinesisClient objects for each target region
       List<KmsMasterKey> masterKeys = new ArrayList<>();
       for (Region region : regions) {
           kinesisClients_.add(KinesisClient.builder()
               .credentialsProvider(credentialsProvider)
               .region(region)
               .build());

           KmsMasterKey regionMasterKey = KmsMasterKeyProvider.builder()
               .defaultRegion(region)
               .builderSupplier(() ->
KmsClient.builder().credentialsProvider(credentialsProvider))
               .buildStrict(kmsAliasName)
               .getMasterKey(kmsAliasName);

           masterKeys.add(regionMasterKey);
       }

       // Collect KmsMasterKey objects into single provider and add cache
       MasterKeyProvider<?> masterKeyProvider =
MultipleProviderFactory.buildMultiProvider(
           KmsMasterKey.class,
           masterKeys
       );

       cachingMaterialsManager_ = CachingCryptoMaterialsManager.newBuilder()
```

```
            .withMasterKeyProvider(masterKeyProvider)
            .withCache(new LocalCryptoMaterialsCache(MAX_CACHE_ENTRIES))
            .withMaxAge(MAX_ENTRY_AGE_MILLISECONDS, TimeUnit.MILLISECONDS)
            .withMessageUseLimit(MAX_ENTRY_USES)
            .build();
    }

    /**
     * JSON serializes and encrypts the received record data and pushes it to all
 target streams.
     */
    public void putRecord(final Map<Object, Object> data) {
        String partitionKey = UUID.randomUUID().toString();
        Map<String, String> encryptionContext = new HashMap<>();
        encryptionContext.put("stream", streamName_);

        // JSON serialize data
        String jsonData = Jackson.toJsonString(data);

        // Encrypt data
        CryptoResult<byte[], ?> result = crypto_.encryptData(
            cachingMaterialsManager_,
            jsonData.getBytes(),
            encryptionContext
        );
        byte[] encryptedData = result.getResult();

        // Put records to Kinesis stream in all Regions
        for (KinesisClient regionalKinesisClient : kinesisClients_) {
            regionalKinesisClient.putRecord(builder ->
                builder.streamName(streamName_)
                    .data(SdkBytes.fromByteArray(encryptedData))
                    .partitionKey(partitionKey));
        }
    }
}
```

Python

```
"""
Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
Licensed under the Apache License, Version 2.0 (the "License"). You may not use this
 file except
in compliance with the License. A copy of the License is located at

https://aws.amazon.com/apache-2-0/

or in the "license" file accompanying this file. This file is distributed on an "AS
 IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
 License for the
specific language governing permissions and limitations under the License.
"""
import json
import uuid

from aws_encryption_sdk import EncryptionSDKClient, StrictAwsKmsMasterKeyProvider,
 CachingCryptoMaterialsManager, LocalCryptoMaterialsCache, CommitmentPolicy
from aws_encryption_sdk.key_providers.kms import KMSMasterKey
import boto3


class MultiRegionRecordPusher(object):
    """Pushes data to Kinesis Streams in multiple Regions."""
    CACHE_CAPACITY = 100
    MAX_ENTRY_AGE_SECONDS = 300.0
    MAX_ENTRY_MESSAGES_ENCRYPTED = 100

    def __init__(self, regions, kms_alias_name, stream_name):
        self._kinesis_clients = []
        self._stream_name = stream_name

        # Set up EncryptionSDKClient
        _client =
 EncryptionSDKClient(CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT)

        # Set up KMSMasterKeyProvider with cache
        _key_provider = StrictAwsKmsMasterKeyProvider(kms_alias_name)

        # Add MasterKey and Kinesis client for each Region
        for region in regions:
            self._kinesis_clients.append(boto3.client('kinesis',
 region_name=region))
            regional_master_key = KMSMasterKey(
                client=boto3.client('kms', region_name=region),
```

```
                key_id=kms_alias_name
            )
            _key_provider.add_master_key_provider(regional_master_key)

        cache = LocalCryptoMaterialsCache(capacity=self.CACHE_CAPACITY)
        self._materials_manager = CachingCryptoMaterialsManager(
            master_key_provider=_key_provider,
            cache=cache,
            max_age=self.MAX_ENTRY_AGE_SECONDS,
            max_messages_encrypted=self.MAX_ENTRY_MESSAGES_ENCRYPTED
        )

    def put_record(self, record_data):
        """JSON serializes and encrypts the received record data and pushes it to
all target streams.

        :param dict record_data: Data to write to stream
        """
        # Kinesis partition key to randomize write load across stream shards
        partition_key = uuid.uuid4().hex

        encryption_context = {'stream': self._stream_name}

        # JSON serialize data
        json_data = json.dumps(record_data)

        # Encrypt data
        encrypted_data, _header = _client.encrypt(
            source=json_data,
            materials_manager=self._materials_manager,
            encryption_context=encryption_context
        )

        # Put records to Kinesis stream in all Regions
        for client in self._kinesis_clients:
            client.put_record(
                StreamName=self._stream_name,
                Data=encrypted_data,
                PartitionKey=partition_key
            )
```

Consommateur

Le consommateur de données est une AWS Lambdafonction déclenchée par des événements Kinesis. Il déchiffre et désérialise chaque enregistrement, puis écrit l'enregistrement en texte brut dans une table Amazon DynamoDB de la même région.

À l'instar du code producteur, le code consommateur permet la mise en cache des clés de données en utilisant un gestionnaire de matériel cryptographique de mise en cache (CMM de mise en cache) lors des appels à la méthode de déchiffrement.

Le code Java crée un fournisseur de clé principale en mode strict avec une valeur spécifiéeAWS KMS key. Le mode strict n'est pas obligatoire lors du déchiffrement, mais il s'agit d'une bonne pratique. Le code Python utilise le mode découverte, qui permet d'AWS Encryption SDKutiliser n'importe quelle clé d'encapsulation qui a chiffré une clé de données pour la déchiffrer.

Java

L'exemple suivant utilise la version 2. x duKit SDK de chiffrement AWS pour Java. Version 3. x du rend Kit SDK de chiffrement AWS pour Java obsolète le CMM de mise en cache des clés de données. Avec la version 3. x, vous pouvez également utiliser le trousseau de clés AWS KMS hiérarchique, une solution alternative de mise en cache des matériaux cryptographiques.

Ce code crée un fournisseur de clé principale pour le déchiffrement en mode strict. Ils ne AWS Encryption SDK peuvent utiliser que celui que AWS KMS keys vous spécifiez pour déchiffrer votre message.

```
/*
 * Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License"). You may not use
 this file except
 * in compliance with the License. A copy of the License is located at
 *
 * http://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed on an
 "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
 License for the
 * specific language governing permissions and limitations under the License.
 */
package com.amazonaws.crypto.examples.kinesisdatakeycaching;
```

```
import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CommitmentPolicy;
import com.amazonaws.encryptionsdk.CryptoResult;
import com.amazonaws.encryptionsdk.caching.CachingCryptoMaterialsManager;
import com.amazonaws.encryptionsdk.caching.LocalCryptoMaterialsCache;
import com.amazonaws.encryptionsdk.kmssdkv2.KmsMasterKeyProvider;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent.KinesisEventRecord;
import com.amazonaws.util.BinaryUtils;
import java.io.UnsupportedEncodingException;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.concurrent.TimeUnit;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;

/**
 * Decrypts all incoming Kinesis records and writes records to DynamoDB.
 */
public class LambdaDecryptAndWrite {

    private static final long MAX_ENTRY_AGE_MILLISECONDS = 600000;
    private static final int MAX_CACHE_ENTRIES = 100;
    private final CachingCryptoMaterialsManager cachingMaterialsManager_;
    private final AwsCrypto crypto_;
    private final DynamoDbTable<Item> table_;

    /**
     * Because the cache is used only for decryption, the code doesn't set the max
 bytes or max
     * message security thresholds that are enforced only on on data keys used for
 encryption.
     */
    public LambdaDecryptAndWrite() {
        String kmsKeyArn = System.getenv("CMK_ARN");
        cachingMaterialsManager_ = CachingCryptoMaterialsManager.newBuilder()

 .withMasterKeyProvider(KmsMasterKeyProvider.builder().buildStrict(kmsKeyArn))
            .withCache(new LocalCryptoMaterialsCache(MAX_CACHE_ENTRIES))
            .withMaxAge(MAX_ENTRY_AGE_MILLISECONDS, TimeUnit.MILLISECONDS)
            .build();
```

```
        crypto_ = AwsCrypto.builder()
            .withCommitmentPolicy(CommitmentPolicy.RequireEncryptRequireDecrypt)
            .build();

        String tableName = System.getenv("TABLE_NAME");
        DynamoDbEnhancedClient dynamodb = DynamoDbEnhancedClient.builder().build();
        table_ = dynamodb.table(tableName, TableSchema.fromClass(Item.class));
    }

    /**
     * @param event
     * @param context
     */
    public void handleRequest(KinesisEvent event, Context context)
        throws UnsupportedEncodingException {
        for (KinesisEventRecord record : event.getRecords()) {
            ByteBuffer ciphertextBuffer = record.getKinesis().getData();
            byte[] ciphertext = BinaryUtils.copyAllBytesFrom(ciphertextBuffer);

            // Decrypt and unpack record
            CryptoResult<byte[], ?> plaintextResult =
 crypto_.decryptData(cachingMaterialsManager_,
                ciphertext);

            // Verify the encryption context value
            String streamArn = record.getEventSourceARN();
            String streamName = streamArn.substring(streamArn.indexOf("/") + 1);
            if (!
streamName.equals(plaintextResult.getEncryptionContext().get("stream"))) {
                throw new IllegalStateException("Wrong Encryption Context!");
            }

            // Write record to DynamoDB
            String jsonItem = new String(plaintextResult.getResult(),
 StandardCharsets.UTF_8);
            System.out.println(jsonItem);
            table_.putItem(Item.fromJSON(jsonItem));
        }
    }

    private static class Item {

        static Item fromJSON(String jsonText) {
```

```
            // Parse JSON and create new Item
            return new Item();
        }
    }
}
```

Python

Ce code Python est déchiffré avec un fournisseur de clé principale en mode découverte. Il permet d'AWS Encryption SDKutiliser n'importe quelle clé d'encapsulation qui a chiffré une clé de données pour la déchiffrer. Le mode strict, dans lequel vous spécifiez les clés d'encapsulation qui peuvent être utilisées pour le déchiffrement, est une bonne pratique.

```
"""
Copyright 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Licensed under the Apache License, Version 2.0 (the "License"). You may not use this
 file except
in compliance with the License. A copy of the License is located at

https://aws.amazon.com/apache-2-0/

or in the "license" file accompanying this file. This file is distributed on an "AS
 IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
 License for the
specific language governing permissions and limitations under the License.
"""
import base64
import json
import logging
import os

from aws_encryption_sdk import EncryptionSDKClient,
 DiscoveryAwsKmsMasterKeyProvider, CachingCryptoMaterialsManager,
 LocalCryptoMaterialsCache, CommitmentPolicy
import boto3

_LOGGER = logging.getLogger(__name__)
_is_setup = False
CACHE_CAPACITY = 100
MAX_ENTRY_AGE_SECONDS = 600.0
```

```
def setup():
    """Sets up clients that should persist across Lambda invocations."""
    global encryption_sdk_client
    encryption_sdk_client =
 EncryptionSDKClient(CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT)

    global materials_manager
    key_provider = DiscoveryAwsKmsMasterKeyProvider()
    cache = LocalCryptoMaterialsCache(capacity=CACHE_CAPACITY)

    #  Because the cache is used only for decryption, the code doesn't set
    #   the max bytes or max message security thresholds that are enforced
    #   only on on data keys used for encryption.
    materials_manager = CachingCryptoMaterialsManager(
        master_key_provider=key_provider,
        cache=cache,
        max_age=MAX_ENTRY_AGE_SECONDS
    )
    global table
    table_name = os.environ.get('TABLE_NAME')
    table = boto3.resource('dynamodb').Table(table_name)
    global _is_setup
    _is_setup = True


def lambda_handler(event, context):
    """Decrypts all incoming Kinesis records and writes records to DynamoDB."""
    _LOGGER.debug('New event:')
    _LOGGER.debug(event)
    if not _is_setup:
        setup()
    with table.batch_writer() as batch:
        for record in event.get('Records', []):
            # Record data base64-encoded by Kinesis
            ciphertext = base64.b64decode(record['kinesis']['data'])

            # Decrypt and unpack record
            plaintext, header = encryption_sdk_client.decrypt(
                source=ciphertext,
                materials_manager=materials_manager
            )
            item = json.loads(plaintext)

            # Verify the encryption context value
```

```
            stream_name = record['eventSourceARN'].split('/', 1)[1]
            if stream_name != header.encryption_context['stream']:
                raise ValueError('Wrong Encryption Context!')

            # Write record to DynamoDB
            batch.put_item(Item=item)
```

Exemple de mise en cache des clés de données :AWS CloudFormationmodèle

CetteAWS CloudFormationconfigure tous les éléments nécessairesAWSressources pour reproduire leExemple de mise en cache des clés de données.

JSON

```
{
    "Parameters": {
        "SourceCodeBucket": {
            "Type": "String",
            "Description": "S3 bucket containing Lambda source code zip files"
        },
        "PythonLambdaS3Key": {
            "Type": "String",
            "Description": "S3 key containing Python Lambda source code zip file"
        },
        "PythonLambdaObjectVersionId": {
            "Type": "String",
            "Description": "S3 version id for S3 key containing Python Lambda source
 code zip file"
        },
        "JavaLambdaS3Key": {
            "Type": "String",
            "Description": "S3 key containing Python Lambda source code zip file"
        },
        "JavaLambdaObjectVersionId": {
            "Type": "String",
            "Description": "S3 version id for S3 key containing Python Lambda source
 code zip file"
        },
        "KeyAliasSuffix": {
```

```
            "Type": "String",
            "Description": "Suffix to use for KMS key Alias (ie: alias/
<KeyAliasSuffix>)"
        },
        "StreamName": {
            "Type": "String",
            "Description": "Name to use for Kinesis Stream"
        }
    },
    "Resources": {
        "InputStream": {
            "Type": "AWS::Kinesis::Stream",
            "Properties": {
                "Name": {
                    "Ref": "StreamName"
                },
                "ShardCount": 2
            }
        },
        "PythonLambdaOutputTable": {
            "Type": "AWS::DynamoDB::Table",
            "Properties": {
                "AttributeDefinitions": [
                    {
                        "AttributeName": "id",
                        "AttributeType": "S"
                    }
                ],
                "KeySchema": [
                    {
                        "AttributeName": "id",
                        "KeyType": "HASH"
                    }
                ],
                "ProvisionedThroughput": {
                    "ReadCapacityUnits": 1,
                    "WriteCapacityUnits": 1
                }
            }
        },
        "PythonLambdaRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
```

```
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": "lambda.amazonaws.com"
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "ManagedPolicyArns": [
                    "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
                ],
                "Policies": [
                    {
                        "PolicyName": "PythonLambdaAccess",
                        "PolicyDocument": {
                            "Version": "2012-10-17",
                            "Statement": [
                                {
                                    "Effect": "Allow",
                                    "Action": [
                                        "dynamodb:DescribeTable",
                                        "dynamodb:BatchWriteItem"
                                    ],
                                    "Resource": {
                                        "Fn::Sub": "arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${PythonLambdaOutputTable}"
                                    }
                                },
                                {
                                    "Effect": "Allow",
                                    "Action": [
                                        "dynamodb:PutItem"
                                    ],
                                    "Resource": {
                                        "Fn::Sub": "arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${PythonLambdaOutputTable}*"
                                    }
                                },
                                {
                                    "Effect": "Allow",
```

```
                                    "Action": [
                                        "kinesis:GetRecords",
                                        "kinesis:GetShardIterator",
                                        "kinesis:DescribeStream",
                                        "kinesis:ListStreams"
                                    ],
                                    "Resource": {
                                        "Fn::Sub": "arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}"
                                    }
                                }
                            ]
                        }
                    }
                ]
            }
        },
        "PythonLambdaFunction": {
            "Type": "AWS::Lambda::Function",
            "Properties": {
                "Description": "Python consumer",
                "Runtime": "python2.7",
                "MemorySize": 512,
                "Timeout": 90,
                "Role": {
                    "Fn::GetAtt": [
                        "PythonLambdaRole",
                        "Arn"
                    ]
                },
                "Handler":
 "aws_crypto_examples.kinesis_datakey_caching.consumer.lambda_handler",
                "Code": {
                    "S3Bucket": {
                        "Ref": "SourceCodeBucket"
                    },
                    "S3Key": {
                        "Ref": "PythonLambdaS3Key"
                    },
                    "S3ObjectVersion": {
                        "Ref": "PythonLambdaObjectVersionId"
                    }
                },
                "Environment": {
```

```
                    "Variables": {
                        "TABLE_NAME": {
                            "Ref": "PythonLambdaOutputTable"
                        }
                    }
                }
            }
        },
        "PythonLambdaSourceMapping": {
            "Type": "AWS::Lambda::EventSourceMapping",
            "Properties": {
                "BatchSize": 1,
                "Enabled": true,
                "EventSourceArn": {
                    "Fn::Sub": "arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}"
                },
                "FunctionName": {
                    "Ref": "PythonLambdaFunction"
                },
                "StartingPosition": "TRIM_HORIZON"
            }
        },
        "JavaLambdaOutputTable": {
            "Type": "AWS::DynamoDB::Table",
            "Properties": {
                "AttributeDefinitions": [
                    {
                        "AttributeName": "id",
                        "AttributeType": "S"
                    }
                ],
                "KeySchema": [
                    {
                        "AttributeName": "id",
                        "KeyType": "HASH"
                    }
                ],
                "ProvisionedThroughput": {
                    "ReadCapacityUnits": 1,
                    "WriteCapacityUnits": 1
                }
            }
        },
```

```
        "JavaLambdaRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": "lambda.amazonaws.com"
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "ManagedPolicyArns": [
                    "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
                ],
                "Policies": [
                    {
                        "PolicyName": "JavaLambdaAccess",
                        "PolicyDocument": {
                            "Version": "2012-10-17",
                            "Statement": [
                                {
                                    "Effect": "Allow",
                                    "Action": [
                                        "dynamodb:DescribeTable",
                                        "dynamodb:BatchWriteItem"
                                    ],
                                    "Resource": {
                                        "Fn::Sub": "arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${JavaLambdaOutputTable}"
                                    }
                                },
                                {
                                    "Effect": "Allow",
                                    "Action": [
                                        "dynamodb:PutItem"
                                    ],
                                    "Resource": {
                                        "Fn::Sub": "arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${JavaLambdaOutputTable}*"
```

```
                                    }
                                },
                                {
                                    "Effect": "Allow",
                                    "Action": [
                                        "kinesis:GetRecords",
                                        "kinesis:GetShardIterator",
                                        "kinesis:DescribeStream",
                                        "kinesis:ListStreams"
                                    ],
                                    "Resource": {
                                        "Fn::Sub": "arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}"
                                    }
                                }
                            ]
                        }
                    }
                ]
            }
        },
        "JavaLambdaFunction": {
            "Type": "AWS::Lambda::Function",
            "Properties": {
                "Description": "Java consumer",
                "Runtime": "java8",
                "MemorySize": 512,
                "Timeout": 90,
                "Role": {
                    "Fn::GetAtt": [
                        "JavaLambdaRole",
                        "Arn"
                    ]
                },
                "Handler":
 "com.amazonaws.crypto.examples.kinesisdatakeycaching.LambdaDecryptAndWrite::handleRequest",
                "Code": {
                    "S3Bucket": {
                        "Ref": "SourceCodeBucket"
                    },
                    "S3Key": {
                        "Ref": "JavaLambdaS3Key"
                    },
                    "S3ObjectVersion": {
```

```
                        "Ref": "JavaLambdaObjectVersionId"
                    }
                },
                "Environment": {
                    "Variables": {
                        "TABLE_NAME": {
                            "Ref": "JavaLambdaOutputTable"
                        },
                        "CMK_ARN": {
                            "Fn::GetAtt": [
                                "RegionKinesisCMK",
                                "Arn"
                            ]
                        }
                    }
                }
            }
        },
        "JavaLambdaSourceMapping": {
            "Type": "AWS::Lambda::EventSourceMapping",
            "Properties": {
                "BatchSize": 1,
                "Enabled": true,
                "EventSourceArn": {
                    "Fn::Sub": "arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}"
                },
                "FunctionName": {
                    "Ref": "JavaLambdaFunction"
                },
                "StartingPosition": "TRIM_HORIZON"
            }
        },
        "RegionKinesisCMK": {
            "Type": "AWS::KMS::Key",
            "Properties": {
                "Description": "Used to encrypt data passing through Kinesis Stream
 in this region",
                "Enabled": true,
                "KeyPolicy": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
```

```
                        "Principal": {
                            "AWS": {
                                "Fn::Sub": "arn:aws:iam::${AWS::AccountId}:root"
                            }
                        },
                        "Action": [
                            "kms:Encrypt",
                            "kms:GenerateDataKey",
                            "kms:CreateAlias",
                            "kms:DeleteAlias",
                            "kms:DescribeKey",
                            "kms:DisableKey",
                            "kms:EnableKey",
                            "kms:PutKeyPolicy",
                            "kms:ScheduleKeyDeletion",
                            "kms:UpdateAlias",
                            "kms:UpdateKeyDescription"
                        ],
                        "Resource": "*"
                    },
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "AWS": [
                                {
                                    "Fn::GetAtt": [
                                        "PythonLambdaRole",
                                        "Arn"
                                    ]
                                },
                                {
                                    "Fn::GetAtt": [
                                        "JavaLambdaRole",
                                        "Arn"
                                    ]
                                }
                            ]
                        },
                        "Action": "kms:Decrypt",
                        "Resource": "*"
                    }
                ]
            }
        }
```

```
        },
        "RegionKinesisCMKAlias": {
            "Type": "AWS::KMS::Alias",
            "Properties": {
                "AliasName": {
                    "Fn::Sub": "alias/${KeyAliasSuffix}"
                },
                "TargetKeyId": {
                    "Ref": "RegionKinesisCMK"
                }
            }
        }
    }
}
```

YAML

```
Parameters:
    SourceCodeBucket:
        Type: String
        Description: S3 bucket containing Lambda source code zip files
    PythonLambdaS3Key:
        Type: String
        Description: S3 key containing Python Lambda source code zip file
    PythonLambdaObjectVersionId:
        Type: String
        Description: S3 version id for S3 key containing Python Lambda source code
 zip file
    JavaLambdaS3Key:
        Type: String
        Description: S3 key containing Python Lambda source code zip file
    JavaLambdaObjectVersionId:
        Type: String
        Description: S3 version id for S3 key containing Python Lambda source code
 zip file
    KeyAliasSuffix:
        Type: String
        Description: 'Suffix to use for KMS CMK Alias (ie: alias/<KeyAliasSuffix>)'
    StreamName:
        Type: String
        Description: Name to use for Kinesis Stream
Resources:
    InputStream:
```

```
        Type: AWS::Kinesis::Stream
        Properties:
            Name: !Ref StreamName
            ShardCount: 2
    PythonLambdaOutputTable:
        Type: AWS::DynamoDB::Table
        Properties:
            AttributeDefinitions:
                -
                    AttributeName: id
                    AttributeType: S
            KeySchema:
                -
                    AttributeName: id
                    KeyType: HASH
            ProvisionedThroughput:
                ReadCapacityUnits: 1
                WriteCapacityUnits: 1
    PythonLambdaRole:
        Type: AWS::IAM::Role
        Properties:
            AssumeRolePolicyDocument:
                Version: 2012-10-17
                Statement:
                    -
                        Effect: Allow
                        Principal:
                            Service: lambda.amazonaws.com
                        Action: sts:AssumeRole
            ManagedPolicyArns:
                - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
            Policies:
                -
                    PolicyName: PythonLambdaAccess
                    PolicyDocument:
                        Version: 2012-10-17
                        Statement:
                            -
                                Effect: Allow
                                Action:
                                    - dynamodb:DescribeTable
                                    - dynamodb:BatchWriteItem
                                Resource: !Sub arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${PythonLambdaOutputTable}
```

```
                            -
                                Effect: Allow
                                Action:
                                    - dynamodb:PutItem
                                Resource: !Sub arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${PythonLambdaOutputTable}*
                            -
                                Effect: Allow
                                Action:
                                    - kinesis:GetRecords
                                    - kinesis:GetShardIterator
                                    - kinesis:DescribeStream
                                    - kinesis:ListStreams
                                Resource: !Sub arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}
    PythonLambdaFunction:
        Type: AWS::Lambda::Function
        Properties:
            Description: Python consumer
            Runtime: python2.7
            MemorySize: 512
            Timeout: 90
            Role: !GetAtt PythonLambdaRole.Arn
            Handler:
 aws_crypto_examples.kinesis_datakey_caching.consumer.lambda_handler
            Code:
                S3Bucket: !Ref SourceCodeBucket
                S3Key: !Ref PythonLambdaS3Key
                S3ObjectVersion: !Ref PythonLambdaObjectVersionId
            Environment:
                Variables:
                    TABLE_NAME: !Ref PythonLambdaOutputTable
    PythonLambdaSourceMapping:
        Type: AWS::Lambda::EventSourceMapping
        Properties:
            BatchSize: 1
            Enabled: true
            EventSourceArn: !Sub arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}
            FunctionName: !Ref PythonLambdaFunction
            StartingPosition: TRIM_HORIZON
    JavaLambdaOutputTable:
        Type: AWS::DynamoDB::Table
        Properties:
```

```
            AttributeDefinitions:
                -
                    AttributeName: id
                    AttributeType: S
            KeySchema:
                -
                    AttributeName: id
                    KeyType: HASH
            ProvisionedThroughput:
                ReadCapacityUnits: 1
                WriteCapacityUnits: 1
    JavaLambdaRole:
        Type: AWS::IAM::Role
        Properties:
            AssumeRolePolicyDocument:
                Version: 2012-10-17
                Statement:
                    -
                        Effect: Allow
                        Principal:
                            Service: lambda.amazonaws.com
                        Action: sts:AssumeRole
            ManagedPolicyArns:
                - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
            Policies:
                -
                    PolicyName: JavaLambdaAccess
                    PolicyDocument:
                        Version: 2012-10-17
                        Statement:
                            -
                                Effect: Allow
                                Action:
                                    - dynamodb:DescribeTable
                                    - dynamodb:BatchWriteItem
                                Resource: !Sub arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${JavaLambdaOutputTable}
                            -
                                Effect: Allow
                                Action:
                                    - dynamodb:PutItem
                                Resource: !Sub arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${JavaLambdaOutputTable}*
                            -
```

```
                                Effect: Allow
                                Action:
                                    - kinesis:GetRecords
                                    - kinesis:GetShardIterator
                                    - kinesis:DescribeStream
                                    - kinesis:ListStreams
                                Resource: !Sub arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}
    JavaLambdaFunction:
        Type: AWS::Lambda::Function
        Properties:
            Description: Java consumer
            Runtime: java8
            MemorySize: 512
            Timeout: 90
            Role: !GetAtt JavaLambdaRole.Arn
            Handler:
 com.amazonaws.crypto.examples.kinesisdatakeycaching.LambdaDecryptAndWrite::handleRequest
            Code:
                S3Bucket: !Ref SourceCodeBucket
                S3Key: !Ref JavaLambdaS3Key
                S3ObjectVersion: !Ref JavaLambdaObjectVersionId
            Environment:
                Variables:
                    TABLE_NAME: !Ref JavaLambdaOutputTable
                    CMK_ARN: !GetAtt RegionKinesisCMK.Arn
    JavaLambdaSourceMapping:
        Type: AWS::Lambda::EventSourceMapping
        Properties:
            BatchSize: 1
            Enabled: true
            EventSourceArn: !Sub arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}
            FunctionName: !Ref JavaLambdaFunction
            StartingPosition: TRIM_HORIZON
    RegionKinesisCMK:
        Type: AWS::KMS::Key
        Properties:
            Description: Used to encrypt data passing through Kinesis Stream in this
 region
            Enabled: true
            KeyPolicy:
                Version: 2012-10-17
                Statement:
```

```
                -
                    Effect: Allow
                    Principal:
                        AWS: !Sub arn:aws:iam::${AWS::AccountId}:root
                    Action:
                        # Data plane actions
                        - kms:Encrypt
                        - kms:GenerateDataKey
                        # Control plane actions
                        - kms:CreateAlias
                        - kms:DeleteAlias
                        - kms:DescribeKey
                        - kms:DisableKey
                        - kms:EnableKey
                        - kms:PutKeyPolicy
                        - kms:ScheduleKeyDeletion
                        - kms:UpdateAlias
                        - kms:UpdateKeyDescription
                    Resource: '*'
                -
                    Effect: Allow
                    Principal:
                        AWS:
                            - !GetAtt PythonLambdaRole.Arn
                            - !GetAtt JavaLambdaRole.Arn
                    Action: kms:Decrypt
                    Resource: '*'
    RegionKinesisCMKAlias:
        Type: AWS::KMS::Alias
        Properties:
            AliasName: !Sub alias/${KeyAliasSuffix}
            TargetKeyId: !Ref RegionKinesisCMK
```

Versions du AWS Encryption SDK

Les implémentations du AWS Encryption SDK langage utilisent le versionnement sémantique pour vous permettre d'identifier plus facilement l'ampleur des changements dans chaque version. Une modification du numéro de version principal, tel que 1. x. x à 2. x. x, indique un changement radical susceptible de nécessiter des modifications de code et un déploiement planifié. Les modifications apportées à une nouvelle version peuvent ne pas avoir d'impact sur tous les cas d'utilisation. Consultez les notes de publication pour voir si vous êtes concerné. Une modification apportée à une version mineure, telle que x .1. x à x 2.2. x, est toujours rétrocompatible, mais peut inclure des éléments obsolètes.

Dans la mesure du possible, utilisez la dernière version du AWS Encryption SDK dans le langage de programmation de votre choix. La politique de maintenance et de support pour chaque version diffère selon les implémentations du langage de programmation. Pour plus de détails sur les versions prises en charge dans votre langage de programmation préféré, consultez le `SUPPORT_POLICY.rst` fichier dans son GitHubréférentiel.

Lorsque les mises à niveau incluent de nouvelles fonctionnalités nécessitant une configuration spéciale pour éviter les erreurs de chiffrement ou de déchiffrement, nous fournissons une version intermédiaire et des instructions détaillées pour son utilisation. Par exemple, les versions 1.7. x et 1.8. x sont conçus pour être des versions de transition qui vous aident à effectuer une mise à niveau à partir de versions antérieures à 1.7. x vers les versions 2.0. x et versions ultérieures. Pour plus de détails, consultez Migrer votreAWS Encryption SDK.

> **Note**
>
> Le x dans un numéro de version représente n'importe quel correctif de la version majeure ou mineure. Par exemple, la version 1.7. x représente toutes les versions qui commencent par 1.7, y compris les versions 1.7.1 et 1.7.9.
> Les nouvelles fonctionnalités de sécurité ont été initialement publiées dans les versions 1.7 de la CLI de AWS chiffrement. x et 2.0. x. Cependant, AWS Encryption CLI version 1.8. x remplace la version 1.7. x et CLI de AWS chiffrement 2.1. x remplace 2.0. x. Pour plus de détails, consultez l'avis de sécurité correspondant dans le aws-encryption-sdk-cliréférentiel sur GitHub.

Les tableaux suivants fournissent un aperçu des principales différences entre les versions prises en charge AWS Encryption SDK pour chaque langage de programmation.

C

Pour une description détaillée de toutes les modifications, consultez le fichier ChangeLog.md dans le référentiel sur. aws-encryption-sdk-c GitHub

Version majeure	Détails		Phase du cycle de vie de la version majeure du SDK
1. x	1.0	Première version.	Phase de fin de support
	1,7	Mises à jour AWS Encryption SDK qui aident les utilisateurs des versions antérieur es à effectuer la mise à niveau vers les versions 2.0. x et versions ultérieures. Pour plus d'informa tions, consultez la version 1.7. x.	
2. x	2.0	Mises à jour du AWS Encryption SDK. Pour plus d'informations, consultez la version 2.0. x.	Disponibilité générale (GA)
	2.2	Améliorations apportées au processus de déchiffrement des messages.	

	2.3	Ajoute la prise en charge des clés AWS KMS multirégionales.	

C#//.NET

Pour une description détaillée de toutes les modifications, consultez le fichier ChangeLog.md dans le référentiel sur. `aws-encryption-sdk-net` GitHub

Version majeure	Détails		Phase du cycle de vie de la version majeure du SDK
3. x	3.0	Première version.	Disponibilité générale (GA) La version 3.x de AWS Encryption SDK for .NET passera en mode maintenance le 13 mai 2024.
4. x	4.0	Ajoute la prise en charge du AWS KMS jeu de clés hiérarchique, du contexte de chiffrement requis CMM et des jeux de clés RSA asymétriques. AWS KMS	Disponibilité générale (GA)

Interface de ligne de commande (CLI)

Pour une description détaillée de toutes les modifications, voir Versions duAWSCLI de chiffrement et le fichier ChangeLog.rst dans le référentiel sur. aws-encryption-sdk-cli GitHub

Version majeure	Détails		Phase du cycle de vie de la version majeure du SDK
1. x	1.0	Première version.	Phase de fin de support
	1,7	Mises à jour AWS Encryption SDK qui aident les utilisateurs des versions antérieures à effectuer la mise à niveau vers les versions 2.0. x et versions ultérieures. Pour plus d'informations, consultez la version 1.7. x.	
2. x	2.0	Mises à jour du AWS Encryption SDK. Pour plus d'informations, consultez la version 2.0. x.	Phase de fin de support
	2.1	Supprime le `--discovery` paramètre et le remplace par l'`discovery` attribut du `--wrapping-keys` paramètre. La version 2.1.0 de la CLI de AWS chiffrement est équivalente à la version 2.0 dans	

		d'autres langages de programmation.	
	2.2	Améliorations apportées au processus de déchiffrement des messages.	
3. x	3.0	Ajoute la prise en charge des clés AWS KMS multirégionales.	Phase de fin de support
4. x	4.0	La CLI de AWS chiffrement ne prend plus en charge Python 2 ou Python 3.4. À partir de la version majeure 4. x de la CLI de AWS chiffrement, seul Python 3.5 ou version ultérieure est pris en charge.	Disponibilité générale (GA)
	4.1	La CLI de AWS chiffrement ne prend plus en charge Python 3.5. À partir de la version 4.1. x de la CLI de AWS chiffrement, seul Python 3.6 ou version ultérieure est pris en charge.	

	4.2	La CLI de AWS chiffrement ne prend plus en charge Python 3.6. À partir de la version 4.2. x de la CLI de AWS chiffrement, seul Python 3.7 ou version ultérieure est pris en charge.	

Java

Pour une description détaillée de toutes les modifications, consultez le fichier ChangeLog.rst dans le référentiel sur. aws-encryption-sdk-java GitHub

Version majeure	Détails		Phase du cycle de vie de la version majeure du SDK
1. x	1.0	Première version.	Phase de fin de support
	1.3	Ajoute la prise en charge du gestionnaire de matériel cryptographique et de la mise en cache des clés de données. Passage à la génération IV déterministe.	
	1.6.1	Obsolète `AwsCrypto.encryptString()` `AwsCrypto.decryptS`	

		`tring()` et les remplace par et. `AwsCrypto.encryptData()` `AwsCrypto.decryptData()`	
	1,7	Mises à jour AWS Encryption SDK qui aident les utilisateurs des versions antérieures à effectuer la mise à niveau vers les versions 2.0. x et versions ultérieures. Pour plus d'informations, consultez la version 1.7. x.	
2. x	2.0	Mises à jour du AWS Encryption SDK. Pour plus d'informations, consultez la version 2.0. x.	Disponibilité générale (GA) La version 2.x Kit SDK de chiffrement AWS pour Java entrera en mode maintenance en 2024.
	2.2	Améliorations apportées au processus de déchiffrement des messages.	
	2.3	Ajoute la prise en charge des clés AWS KMS multirégionales.	
	2,4	Ajoute le support pour AWS SDK for Java 2.x.	

3. x	3.0	Intègre Kit SDK de chiffrement AWS pour Java la bibliothèque des fournisseurs de matériaux. Ajoute la prise en charge des porte-clés RSA symétriques et asymétriques, des AWS KMS porte-clés AWS KMS hiérarchiques, des porte-clés AES bruts, des porte-clés RSA bruts, des porte-clés multiples et du contexte de chiffrement requis CMM.	Disponibilité générale (GA)

JavaScript

Pour une description détaillée de toutes les modifications, consultez le fichier ChangeLog.md dans le référentiel sur. aws-encryption-sdk-javascript GitHub

Version majeure	Détails		Phase du cycle de vie de la version majeure du SDK
1. x	1.0	Première version.	Phase de fin de support
	1,7	Mises à jour AWS Encryption SDK qui aident les utilisateurs des versions antérieures à effectuer la mise	

		à niveau vers les versions 2.0. x et versions ultérieures. Pour plus d'informations, consultez la version 1.7. x.	
2. x	2.0	Mises à jour du AWS Encryption SDK. Pour plus d'informations, consultez la version 2.0. x.	Phase de fin de support
	2.2	Améliorations apportées au processus de déchiffrement des messages.	
	2.3	Ajoute la prise en charge des clés AWS KMS multirégionales.	
3. x	3.0	Supprime la couverture CI pour le nœud 10. Met à niveau les dépendances pour ne plus prendre en charge les nœuds 8 et 10.	Maintenance Support pour la version 3.x du Kit SDK de chiffrement AWS pour JavaScript prendra fin le 17 janvier 2024.

4. x	4.0	Nécessite la version 3 Kit SDK de chiffrement AWS pour JavaScript des `kms-client` pour utiliser le AWS KMS porte-clés.	Disponibilité générale (GA)

Python

Pour une description détaillée de toutes les modifications, consultez le fichier ChangeLog.rst dans le référentiel sur. aws-encryption-sdk-python GitHub

Version majeure	Détails		Phase du cycle de vie de la version majeure du SDK
1. x	1.0	Première version.	Phase de fin de support
	1.3	Ajoute la prise en charge du gestionnaire de matériel cryptographique et de la mise en cache des clés de données. Passage à la génération IV déterministe.	
	1,7	Mises à jour AWS Encryption SDK qui aident les utilisateurs des versions antérieures à effectuer la mise à niveau vers les versions 2.0. x et	

		versions ultérieures. Pour plus d'informations, consultez la version 1.7. x.	
2. x	2.0	Mises à jour du AWS Encryption SDK. Pour plus d'informations, consultez la version 2.0. x.	Phase de fin de support
	2.2	Améliorations apportées au processus de déchiffrement des messages.	
	2.3	Ajoute la prise en charge des clés AWS KMS multirégionales.	
3. x	3.0	Python 2 ou Python 3.4 Kit SDK de chiffrement AWS pour Python ne sont plus pris en charge. À partir de la version majeure 3. x du Kit SDK de chiffrement AWS pour Python, seul Python 3.5 ou version ultérieure est pris en charge.	Disponibilité générale (GA)

Détails de la version

La liste suivante décrit les principales différences entre les versions prises en charge du AWS Encryption SDK.

Rubriques

Versions antérieures à 1.7. x

> **Note**
>
> Tous les 1. x. x versions de celui-ci AWS Encryption SDK sont en end-of-supportcours de phase. Passez à la dernière version disponible AWS Encryption SDK de votre langage de programmation dès que possible. Pour effectuer une mise à niveau depuis une AWS Encryption SDK version antérieure à 1.7. x, vous devez d'abord passer à la version 1.7. x. Pour plus de détails, consultez Migrer votreAWS Encryption SDK.

Versions des versions AWS Encryption SDK antérieures à 1.7. x fournissent des fonctionnalités de sécurité importantes, notamment le chiffrement à l'aide de l'algorithme Advanced Encryption Standard en mode Galois/Counter (AES-GCM), une fonction de dérivation de extract-and-expand clé basée sur HMAC (HKDF), la signature et une clé de chiffrement 256 bits. Cependant, ces versions ne prennent pas en charge les meilleures pratiques que nous recommandons, y compris les engagements clés.

La version 1.7. x

> **Note**
>
> Tous les 1. x. x versions de celui-ci AWS Encryption SDK sont en end-of-supportcours de phase.

La version 1.7. x est conçu pour aider les utilisateurs des versions antérieures du AWS Encryption SDK à passer aux versions 2.0. x et versions ultérieures. Si vous découvrez le AWS Encryption SDK, vous pouvez ignorer cette version et commencer par la dernière version disponible dans votre langage de programmation.

La version 1.7. x est entièrement rétrocompatible ; il n'introduit aucun changement radical ni ne modifie le comportement du AWS Encryption SDK. Il est également compatible avec les transmissions ; il vous permet de mettre à jour votre code afin qu'il soit compatible avec la version 2.0. x. Il inclut de nouvelles fonctionnalités, mais ne les active pas complètement. Et cela nécessite des valeurs de configuration qui vous empêchent d'adopter immédiatement toutes les nouvelles fonctionnalités tant que vous n'êtes pas prêt.

La version 1.7. x inclut les modifications suivantes :

AWS KMS mises à jour du fournisseur de clés principales (obligatoire)

La version 1.7. x introduit de nouveaux constructeurs dans le Kit SDK de chiffrement AWS pour Java et Kit SDK de chiffrement AWS pour Python qui créent explicitement des fournisseurs de clés AWS KMS principales en mode strict ou en mode découverte. Cette version apporte des modifications similaires à l'interface de AWS Encryption SDK ligne de commande (CLI). Pour plus de détails, consultez Mise à jourAWS KMS des fournisseurs de clés principales.

- En mode strict, les fournisseurs de clés AWS KMS principales ont besoin d'une liste de clés d'encapsulation, et ils chiffrent et déchiffrent uniquement avec les clés d'encapsulation que vous spécifiez. Il s'agit d'une AWS Encryption SDK bonne pratique qui garantit que vous utilisez les clés d'encapsulage que vous avez l'intention d'utiliser.
- En mode découverte, les fournisseurs de clés AWS KMS principales n'acceptent aucune clé d'encapsulation. Vous ne pouvez pas les utiliser pour le chiffrement. Lors du déchiffrement, ils peuvent utiliser n'importe quelle clé d'encapsulation pour déchiffrer une clé de données chiffrée. Cependant, vous pouvez limiter les clés d'encapsulation utilisées pour le déchiffrement à celles-ci en particulier Comptes AWS. Le filtrage des comptes est facultatif, mais c'est une bonne pratique que nous recommandons.

Les constructeurs qui créent des versions antérieures des fournisseurs de clés AWS KMS principales sont obsolètes dans la version 1.7. x et supprimé dans la version 2.0. x. Ces constructeurs instancient des fournisseurs de clés principales qui chiffrent à l'aide des clés d'encapsulage que vous spécifiez. Cependant, ils déchiffrent les clés de données chiffrées à l'aide de la clé d'encapsulation qui les a chiffrées, sans tenir compte des clés d'encapsulation spécifiées. Les utilisateurs peuvent involontairement déchiffrer des messages à l'aide de clés

d'encapsulation qu'ils n'ont pas l'intention d'utiliser, y compris AWS KMS keys dans d'autres régions ou régions Comptes AWS .

Aucune modification n'a été apportée aux constructeurs des clés AWS KMS principales. Lors du chiffrement et du déchiffrement, les clés AWS KMS principales utilisent uniquement celles que vous spécifiez AWS KMS key .

AWS KMS mises à jour du trousseau de clés (facultatif)

La version 1.7. x ajoute un nouveau filtre aux Kit SDK de chiffrement AWS pour JavaScript implémentations Kit SDK de chiffrement AWS pour C et qui limite les porte-clés de AWS KMS découverte à des données spécifiques. Comptes AWS Ce nouveau filtre de compte est facultatif, mais c'est une bonne pratique que nous recommandons. Pour plus de détails, consultez Mise à jour en coursAWS KMSporte-clés.

Aucune modification n'a été apportée aux constructeurs des AWS KMS porte-clés. Les AWS KMS porte-clés standard se comportent comme des fournisseurs de clés principales en mode strict. AWS KMS les porte-clés de découverte sont créés explicitement en mode découverte.

Transmission d'un identifiant clé pour le AWS KMS déchiffrer

À partir de la version 1.7. x, lors du déchiffrement de clés de données chiffrées, le spécifie AWS Encryption SDK toujours un AWS KMS key dans ses appels à l' AWS KMS opération Decrypt. AWS Encryption SDK obtient la valeur de l'ID de clé pour le AWS KMS key à partir des métadonnées de chaque clé de données cryptée. Cette fonctionnalité ne nécessite aucune modification de code.

AWS KMS key Il n'est pas nécessaire de spécifier l'ID de clé pour déchiffrer le texte chiffré sous une clé KMS de chiffrement symétrique, mais il s'agit d'une bonne pratique.AWS KMS Tout comme la spécification des clés d'encapsulation dans votre fournisseur de clés, cette pratique garantit le déchiffrement AWS KMS uniquement à l'aide de la clé d'encapsulation que vous avez l'intention d'utiliser.

Déchiffrez le texte chiffré avec un engagement clé

La version 1.7. x peut déchiffrer le texte chiffré avec ou sans clé d'engagement. Cependant, il ne peut pas crypter le texte chiffré avec un engagement clé. Cette propriété vous permet de déployer entièrement des applications capables de déchiffrer le texte chiffré avec clé d'engagement avant qu'elles ne rencontrent un tel texte chiffré. Comme cette version déchiffre les messages chiffrés sans engagement de clé, il n'est pas nécessaire de rechiffrer le texte chiffré.

Pour implémenter ce comportement, version 1.7. x inclut un nouveau paramètre de configuration de la politique d'engagement qui détermine s'ils AWS Encryption SDK peuvent chiffrer ou déchiffrer avec un engagement clé. Dans la version 1.7. x, la seule valeur valide pour la politique d'engagement`ForbidEncryptAllowDecrypt`, est utilisée dans toutes les opérations de chiffrement et de déchiffrement. Cette valeur AWS Encryption SDK empêche le chiffrement avec l'une ou l'autre des nouvelles suites d'algorithmes qui incluent un engagement de clé. Il permet de AWS Encryption SDK déchiffrer le texte chiffré avec et sans engagement de clé.

Bien qu'il n'y ait qu'une seule valeur de politique d'engagement valide dans la version 1.7. x, nous exigeons que vous puissiez définir cette valeur de manière explicite lorsque vous utilisez les nouvelles API introduites dans cette version. La définition explicite de la valeur empêche que votre politique d'engagement soit automatiquement modifiée `require-encrypt-require-decrypt` lorsque vous passez à la version 2.1. x. Au lieu de cela, vous pouvez migrer votre politique d'engagement par étapes.

Des suites d'algorithmes avec un engagement clé

La version 1.7. x inclut deux nouvelles suites d'algorithmes qui soutiennent les engagements clés. L'un inclut la signature, l'autre non. Comme les suites d'algorithmes précédemment prises en charge, ces deux nouvelles suites d'algorithmes incluent le chiffrement avec AES-GCM, une clé de chiffrement 256 bits et une fonction de dérivation de clé basée sur HMAC extract-and-expand (HKDF).

Cependant, la suite d'algorithmes par défaut utilisée pour le chiffrement ne change pas. Ces suites d'algorithmes sont ajoutées à la version 1.7. x pour préparer votre application à les utiliser dans les versions 2.0. x et versions ultérieures.

Changements de mise en œuvre du CMM

La version 1.7. x introduit des modifications à l'interface du gestionnaire de matériaux cryptographiques (CMM) par défaut pour soutenir l'engagement clé. Cette modification ne vous concerne que si vous avez écrit un CMM personnalisé. Pour plus de détails, consultez la documentation ou le GitHub référentiel de l'API correspondant à votre langage de programmation.

La version 2.0. x

La version 2.0. x prend en charge les nouvelles fonctionnalités de sécurité proposées dans le AWS Encryption SDK, notamment les clés d'emballage spécifiées et l'engagement des clés. Pour prendre en charge ces fonctionnalités, version 2.0. x inclut les modifications majeures apportées aux versions antérieures du AWS Encryption SDK. Vous pouvez vous préparer à ces modifications en déployant la

version 1.7. x. La version 2.0. x inclut toutes les nouvelles fonctionnalités introduites dans la version 1.7. x avec les ajouts et modifications suivants.

> **Note**
>
> Version 2. x. x du Kit SDK de chiffrement AWS pour Python Kit SDK de chiffrement AWS pour JavaScript, et la CLI de AWS chiffrement sont en end-of-supportphase.
> Pour plus d'informations sur le support et la maintenance de cette AWS Encryption SDK version dans votre langage de programmation préféré, consultez le `SUPPORT_POLICY.rst` fichier dans son GitHubréférentiel.

AWS KMS fournisseurs de clés principales

Les constructeurs du fournisseur de clés AWS KMS principales d'origine qui étaient obsolètes dans la version 1.7. x sont supprimés dans la version 2.0. x. Vous devez explicitement créer des fournisseurs de clés AWS KMS principales en mode strict ou en mode découverte.

Chiffrez et déchiffrez le texte chiffré avec un engagement clé

La version 2.0. x peut chiffrer et déchiffrer le texte chiffré avec ou sans clé d'engagement. Son comportement est déterminé par le paramétrage de la politique d'engagement. Par défaut, il chiffre toujours avec un engagement par clé et ne déchiffre que le texte chiffré chiffré avec un engagement par clé. À moins que vous ne changiez la politique d'engagement, les textes chiffrés ne AWS Encryption SDK seront pas déchiffrés par une version antérieure du AWS Encryption SDK, y compris la version 1.7. x.

> **Important**
>
> Par défaut, version 2.0. x ne déchiffrera aucun texte chiffré chiffré sans engagement de clé. Si votre application risque de rencontrer un texte chiffré sans engagement de clé, définissez une valeur de politique d'engagement avec. `AllowDecrypt`

Dans la version 2.0. x, le paramètre de la politique d'engagement comporte trois valeurs valides :

- `ForbidEncryptAllowDecrypt`— Ils AWS Encryption SDK ne peuvent pas chiffrer avec un engagement clé. Il peut déchiffrer les textes chiffrés avec ou sans clé d'engagement.
- `RequireEncryptAllowDecrypt`— Ils AWS Encryption SDK doivent chiffrer avec un engagement clé. Il peut déchiffrer les textes chiffrés avec ou sans clé d'engagement.

- `RequireEncryptRequireDecrypt`(par défaut) — Le chiffrement AWS Encryption SDK doit être effectué avec un engagement clé. Il ne déchiffre que les textes chiffrés avec un engagement clé.

Si vous migrez d'une version antérieure AWS Encryption SDK vers la version 2.0. x, définissez la politique d'engagement sur une valeur garantissant que vous pouvez déchiffrer tous les textes chiffrés existants que votre application est susceptible de rencontrer. Il est probable que vous modifiiez ce paramètre au fil du temps.

La version 2.2. x

Prend en charge les signatures numériques et limite les clés de données chiffrées.

> **Note**
>
> Version 2. x. x du Kit SDK de chiffrement AWS pour Python Kit SDK de chiffrement AWS pour JavaScript, et la CLI de AWS chiffrement sont en end-of-supportphase.
> Pour plus d'informations sur le support et la maintenance de cette AWS Encryption SDK version dans votre langage de programmation préféré, consultez le `SUPPORT_POLICY.rst` fichier dans son GitHubréférentiel.

Signatures numériques

Pour améliorer la gestion des signatures numériques lors du déchiffrement, les fonctionnalités suivantes sont AWS Encryption SDK incluses :

- Mode non diffusé : renvoie du texte brut uniquement après avoir traité toutes les entrées, y compris la vérification de la signature numérique si elle est présente. Cette fonctionnalité vous empêche d'utiliser du texte brut avant de vérifier la signature numérique. Utilisez cette fonctionnalité chaque fois que vous déchiffrez des données chiffrées à l'aide de signatures numériques (suite d'algorithmes par défaut). Par exemple, étant donné que la CLI de AWS chiffrement traite toujours les données en mode streaming, utilisez le `- -buffer` paramètre lors du déchiffrement du texte chiffré à l'aide de signatures numériques.
- Mode de déchiffrement non signé uniquement : cette fonctionnalité ne déchiffre que le texte chiffré non signé. Si le déchiffrement rencontre une signature numérique dans le texte chiffré, l'opération échoue. Utilisez cette fonctionnalité pour éviter de traiter involontairement le texte brut des messages signés avant de vérifier la signature.

Limiter les clés de données chiffrées

Vous pouvez limiter le nombre de clés de données chiffrées dans un message chiffré. Cette fonctionnalité peut vous aider à détecter un fournisseur de clé principale ou un jeu de clés mal configuré lors du chiffrement, ou à identifier un texte chiffré malveillant lors du déchiffrement.

Vous devez limiter les clés de données chiffrées lorsque vous déchiffrez des messages provenant d'une source non fiable. Cela évite les appels inutiles, coûteux et potentiellement exhaustifs vers votre infrastructure clé.

La version 2.3. x

Ajoute la prise en charge des clés AWS KMS multirégionales. Pour plus de détails, consultez Utilisation de plusieurs régions AWS KMS keys.

> **Note**
>
> La CLI AWS de chiffrement prend en charge les clés multirégionales à partir de la version 3.0. x.
> Version 2. x. x du Kit SDK de chiffrement AWS pour Python Kit SDK de chiffrement AWS pour JavaScript, et la CLI de AWS chiffrement sont en end-of-supportphase.
> Pour plus d'informations sur le support et la maintenance de cette AWS Encryption SDK version dans votre langage de programmation préféré, consultez le `SUPPORT_POLICY.rst` fichier dans son GitHubréférentiel.

Migrer votreAWS Encryption SDK

Dans laAWS Encryption SDKprend en charge plusieurs systèmes interopérablesimplémentations de langage de programmation, chacun d'entre eux étant développé dans un référentiel open source sur GitHub. En tant quebonne pratique, nous vous recommandons d'utiliser la dernière version duAWS Encryption SDKpour chaque langue.

Vous pouvez effectuer la mise à niveau vers la version 2.0.xou plus tardAWS Encryption SDKvers la dernière version. Cependant, le 2.0.xversion duAWS Encryption SDKintroduit de nouvelles fonctionnalités de sécurité importantes, dont certaines constituent des changements majeurs. Pour effectuer une mise à niveau depuis des versions antérieures à 1.7.xvers les versions 2.0.xet plus tard, vous devez d'abord passer à la dernière version 1.xversion. Les rubriques de cette section sont conçues pour vous aider à comprendre les modifications, à sélectionner la version appropriée pour votre application et à migrer en toute sécurité et avec succès vers les dernières versions deAWS Encryption SDK.

Pour plus d'informations sur les versions importantes deAWS Encryption SDK, voir...Versions du AWS Encryption SDK.

> **Important**
>
> N'effectuez pas de mise à niveau directe à partir d'une version antérieure à 1.7.xvers la version 2.0.xou version ultérieure sans mise à niveau préalable vers la dernière version 1.xversion. Si vous effectuez la mise à niveau vers la version 2.0.xou plus tard et activez immédiatement toutes les nouvelles fonctionnalités,AWS Encryption SDKne sera pas en mesure de déchiffrer le texte chiffré sous les anciennes versions duAWS Encryption SDK.

> **Note**
>
> La version la plus ancienne duAWS Encryption SDKpour .NET est la version 3.0.x. Toutes les versions duAWS Encryption SDKpour .NET, prenez en charge les meilleures pratiques de sécurité introduites dans la version 2.0.xde laAWS Encryption SDK. Vous pouvez passer à la dernière version en toute sécurité sans aucune modification du code ou des données. AWSCLI de chiffrement : Lorsque vous lisez ce guide de migration, utilisez la version 1.7.xinstructions de migration pourAWSVersion de chiffrement 1.8.xet utilisez le

2.0.xinstructions de migration pourAWSVersion de chiffrement 2.1.x. Pour plus d'informations, consultez.Versions duAWSCLI de chiffrement
Les nouvelles fonctionnalités de sécurité ont été initialement publiées enAWSVersion de chiffrement 1.7.xet 2.0 et 2.0.x. Cependant, les XAWSVersion 1.8 de la CLI de chiffrementxremplace la version 1.7.xetAWSVersion de chiffrement 2.1.xremplace 2.0.x. Pour plus d'informations, consultez laavis de sécuritédans leaws-encryption-sdk-cliréférentiel sur GitHub.

Nouveaux utilisateurs

Si vous débutez avecAWS Encryption SDK, installez la dernière version duAWS Encryption SDKpour votre langage de programmation. Les valeurs par défaut activent toutes les fonctionnalités de sécurité duAWS Encryption SDK, y compris le chiffrement avec signature, dérivation de clés etEngagement clé. de laAWS Encryption SDK

Utilisateurs actuels

Nous vous recommandons de passer de la version actuelle vers la dernière version disponible. Tous les 1.xversions duAWS Encryption SDKsont dans leend-of-support phase, comme le sont les versions ultérieures de certains langages de programmation. Pour plus de détails sur l'état du support et de la maintenance duAWS Encryption SDKdans votre langage de programmation, voirSupport et maintenance.

AWS Encryption SDKversion 2.0.xet fournit ultérieurement de nouvelles fonctionnalités de sécurité pour protéger vos données. Cependant, les XAWS Encryption SDKversion 2.0.xinclut les modifications de rupture qui ne sont pas rétrocompatibles. Pour assurer une transition en toute sécurité, commencez par migrer de votre version actuelle vers la version 1 la plus récente.xdans votre langage de programmation. Quand votre dernier 1.xLa version est entièrement déployée et fonctionne correctement, vous pouvez migrer en toute sécurité vers les versions 2.0.xet plus tard. Ceprocessus en deux étapesest essentiel, en particulier pour les applications distribuées.

Pour plus d'informations sur leAWS Encryption SDKles fonctionnalités de sécurité qui sous-tendent ces modifications, voirChiffrement côté client amélioré : Explicite KeyIds et engagement clédans leAWSBlog de sécurité.

Vous recherchez de l'aide pour utiliser leKit SDK de chiffrement AWS pour Javaavec leAWS SDK for Java 2.x? Consultez Prerequisites (Prérequis).

Rubriques

Comment migrer et déployerAWS Encryption SDK

Lors de la migration depuis unAWS Encryption SDKversion ultérieure à 1.7.xvers la version 2.0.xou ultérieurement, vous devez passer en toute sécurité au chiffrement avecEngagement clé. Sinon, votre application rencontrera des textes chiffrés qu'elle ne pourra pas déchiffrer. Si vous utilisezAWS KMSfournisseurs de clés principales, vous devez passer à de nouveaux constructeurs qui créent des fournisseurs de clés principales en mode strict ou en mode découverte.

> **Note**
>
> Cette rubrique est destinée aux utilisateurs qui effectuent une migration depuis des versions antérieures deAWS Encryption SDKvers la version 2.0.xou plus tard. Si vous êtes un nouvel utilisateur duAWS Encryption SDK, vous pouvez commencer à utiliser la dernière version disponible immédiatement avec les paramètres par défaut.

Pour éviter une situation critique dans laquelle vous ne pouvez pas déchiffrer le texte chiffré que vous devez lire, nous vous recommandons de procéder à la migration et au déploiement en plusieurs étapes distinctes. Vérifiez que chaque étape est complète et entièrement déployée avant de commencer l'étape suivante. Cela est particulièrement important pour les applications distribuées avec plusieurs hôtes.

Étape 1 : : Mettez à jour votre application vers la dernière version.xversion

Mise à jour vers la dernière version 1.xversion pour votre langage de programmation. Testez attentivement, déployez vos modifications et confirmez que la mise à jour s'est propagée à tous les hôtes de destination avant de commencer l'étape 2.

> **Important**
>
> Vérifiez que votre dernier 1.xversion 1.7.xou une version ultérieure duAWS Encryption SDK.

Le dernier 1.xversions duAWS Encryption SDKsont rétrocompatibles avec les anciennes versions duAWS Encryption SDKet versions ultérieures compatibles avec les versions 2.0.xet plus tard. Ils incluent les nouvelles fonctionnalités de la version 2.0.x, mais incluez des paramètres par défaut sûrs conçus pour cette migration. Ils vous permettent de mettre à jour votreAWS KMSprincipaux fournisseurs de clés, si nécessaire, et pour un déploiement complet avec des suites d'algorithmes capables de déchiffrer le texte chiffré avec un engagement clé.

- Remplacez les éléments obsolètes, y compris les constructeurs pour les anciensAWS KMSprincipaux fournisseurs de clés. DansPython, veillez à activer les avertissements d'obsolescence. Éléments de code obsolètes dans la dernière version 1.xles versions sont supprimées des versions 2.0.xet plus tard.
- Définissez explicitement votre politique d'engagement pour`ForbidEncryptAllowDecrypt`. Bien que ce soit la seule valeur valide dans la dernière valeur 1.xversions, ce paramètre est requis lorsque vous utilisez les API introduites dans cette version. Il empêche votre application de rejeter le texte chiffré crypté sans engagement clé lorsque vous migrez vers la version 2.0.xet plus tard. Pour plus d'informations, consultez.the section called “Définition de votre politique d'engagement”
- Si vous utilisez l'optionAWS KMSfournisseurs de clés principales, vous devez mettre à jour vos fournisseurs de clés principales existants pour qu'ils prennent en chargeMode strictetmode découverte. Cette mise à jour est requise pourKit SDK de chiffrement AWS pour Java,Kit SDK de chiffrement AWS pour Python, et leAWSCLI de chiffrement. Si vous utilisez des fournisseurs de clés principales en mode découverte, nous vous recommandons d'implémenter le filtre de découverte qui limite les clés d'encapsulation utilisées à ceux en particulierComptes AWS. Cette mise à jour est facultative, mais il s'agit d'unbonne pratiqueque nous recommandons. Pour plus d'informations, consultez.Mise à jourAWS KMS des fournisseurs de clés principales
- Si vous utilisez l'optionAWS KMSporte-clés, nous vous recommandons d'inclure un filtre de découverte qui limite les clés d'encapsulage utilisées pour le déchiffrement à celles en particulierComptes AWS. Cette mise à jour est facultative, mais il s'agit d'unbonne pratiqueque nous recommandons. Pour plus d'informations, consultez.Mise à jour en coursAWS KMSporte-clés

Étape 2 : Mettez à jour votre application vers la dernière version

Après avoir déployé la dernière version 1.xversion réussie sur tous les hôtes, vous pouvez passer à la version 2.0.xet plus tard. Version 2.0xinclut des modifications majeures pour toutes les versions précédentes deAWS Encryption SDK. Toutefois, si vous apportez les modifications de code recommandées à l'étape 1, vous pouvez éviter les erreurs lors de la migration vers la dernière version.

Avant de passer à la dernière version, vérifiez que votre politique d'engagement est toujours définie sur`ForbidEncryptAllowDecrypt`. Ensuite, en fonction de la configuration de vos données, vous pouvez migrer à votre rythme vers`RequireEncryptAllowDecrypt`puis au réglage par défaut,`RequireEncryptRequireDecrypt`. Nous recommandons une série d'étapes de transition, comme le modèle suivant.

1. Commencez par votrepolitique d'engagementparamétré sur`ForbidEncryptAllowDecrypt`. Dans laAWS Encryption SDKpeut décrypter les messages avec un engagement clé, mais il ne le chiffre pas encore avec un engagement clé.
2. Une fois que vous êtes prêt, mettez à jour votre politique d'engagement pour`RequireEncryptAllowDecrypt`. Dans laAWS Encryption SDKcommence à chiffrer vos données avecEngagement clé. Il peut déchiffrer le texte chiffré avec et sans engagement clé.

   Avant de mettre à jour votre politique d'engagement pour`RequireEncryptAllowDecrypt`, vérifiez que votre dernier numéro 1.xLa version est déployée sur tous les hôtes, y compris les hôtes de toutes les applications qui déchiffrent le texte chiffré que vous produisez. Versions duAWS Encryption SDKavant la version 1.7.xImpossible de décrypter les messages cryptés avec un engagement clé.

   C'est également le bon moment pour ajouter des mesures à votre application afin de déterminer si vous êtes toujours en train de traiter du texte chiffré sans engagement clé. Cela vous aidera à déterminer quand il est sûr de mettre à jour les paramètres de votre politique d'engagement pour`RequireEncryptRequireDecrypt`. Pour certaines applications, telles que celles qui cryptent les messages dans une file d'attente Amazon SQS, cela peut impliquer d'attendre suffisamment longtemps pour que tout le texte chiffré sous les anciennes versions soit recrypté ou supprimé. Pour d'autres applications, telles que les objets S3 cryptés, vous devrez peut-être télécharger, chiffrer à nouveau et charger à nouveau tous les objets.
3. Lorsque vous êtes certain qu'aucun message n'est crypté sans engagement clé, vous pouvez mettre à jour votre politique d'engagement pour`RequireEncryptRequireDecrypt`. Cette valeur garantit que vos données sont toujours cryptées et décryptées avec un engagement clé. Il s'agit

du paramètre par défaut, vous n'êtes donc pas obligé de le définir explicitement, mais nous le recommandons. Un paramètre explicite serad[débogage]et toutes les annulations potentielles qui pourraient être nécessaires si votre application reçoit du texte chiffré sans engagement de clé.

Mise à jourAWS KMS des fournisseurs de clés principales

Pour migrer vers la dernière version 1. version x duAWS Encryption SDK, puis à la version 2.0. x ou version ultérieure, vous devez remplacer les fournisseursAWS KMS de clés principales existants par des fournisseurs de clés principales créés explicitement en mode strict ou en mode découverte. Les fournisseurs de clés principales vers la version 1.7. x et supprimé dans la version 2.0. x. Cette modification est requise pour les applications et les scripts qui utilisent le Kit SDK de chiffrement AWS pour JavaKit SDK de chiffrement AWS pour Python, et l'interface de ligne de commande deAWS chiffrement. Les exemples de cette section vous montreront comment mettre à jour votre code.

> **Note**
>
> En Python, activez les avertissements d'obsolescence. Cela vous aidera à identifier les parties de votre code que vous devez mettre à jour.

Si vous utilisez une cléAWS KMS principale (et non un fournisseur de clé principale), vous pouvez ignorer cette étape. AWS KMSles clés principales ne sont ni obsolètes ni supprimées. Ils chiffrent et déchiffrent uniquement à l'aide des clés d'encapsulation que vous spécifiez.

Les exemples de cette section se concentrent sur les éléments de votre code que vous devez modifier. Pour un exemple complet du code mis à jour, consultez la section Exemples du GitHub référentiel correspondant à votre langage de programmation. De plus, ces exemples utilisent généralement des ARN clés pour représenterAWS KMS keys. Lorsque vous créez un fournisseur de clé principale pour le chiffrement, vous pouvez utiliser n'importe quel identifiant deAWS KMS clé valide pour représenter unAWS KMS key. Lorsque vous créez un fournisseur de clé principale pour le déchiffrement, vous devez utiliser un ARN de clé.

En savoir plus sur la migration

Pour tous lesAWS Encryption SDK utilisateurs, découvrez comment définir votre politique d'engagement dansthe section called “Définition de votre politique d'engagement”.

PourKit SDK de chiffrement AWS pour C lesKit SDK de chiffrement AWS pour JavaScript utilisateurs finaux, découvrez une mise à jour facultative des porte-clés dansMise à jour en coursAWS KMSporte-clés.

Rubriques

Migration vers le mode strict

Après la mise à jour vers la dernière version 1. version x duAWS Encryption SDK, remplacez vos anciens fournisseurs de clés principales par des fournisseurs de clés principales en mode strict. En mode strict, vous devez spécifier les clés d'encapsulation à utiliser lors du chiffrement et du déchiffrement. AWS Encryption SDKutilise uniquement les clés d'encapsulation que vous spécifiez. Les fournisseurs de clés principales obsolètes peuvent déchiffrer les données à l'aide deAWS KMS key n'importe quelle clé de données cryptée, y comprisAWS KMS keys dans différentesComptes AWS régions.

Les fournisseurs de clés principales en mode strict sont introduits dans laAWS Encryption SDK version 1.7. x. Ils remplacent les anciens fournisseurs de clés principales, qui sont obsolètes depuis la version 1.7. x et supprimé dans la version 2.0. x. L'utilisation de fournisseurs de clés principales en mode strict est uneAWS Encryption SDK bonne pratique.

Le code suivant crée un fournisseur de clé principale en mode strict que vous pouvez utiliser pour chiffrer et déchiffrer.

Java

Cet exemple représente le code d'une application qui utilise la version 1.6.2 ou une version antérieure duKit SDK de chiffrement AWS pour Java.

Ce code utilise la`KmsMasterKeyProvider.builder()` méthode pour instancier un fournisseur de cléAWS KMS principale qui en utilise unAWS KMS key comme clé d'encapsulation.

```
// Create a master key provider
// Replace the example key ARN with a valid one
String awsKmsKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
```

```
KmsMasterKeyProvider masterKeyProvider = KmsMasterKeyProvider.builder()
    .withKeysForEncryption(awsKmsKey)
    .build();
```

Cet exemple représente le code d'une application qui utilise la version 1.7. x ou version ultérieure duKit SDK de chiffrement AWS pour Java. Pour un exemple complet, consultez BasicEncryptionExample.java.

Les`Builder.withKeysForEncryption()` méthodes`Builder.build()` et utilisées dans l'exemple précédent sont obsolètes dans la version 1.7. x et sont supprimés de la version 2.0. x.

Pour passer à un fournisseur de clé principale en mode strict, ce code remplace les appels à des méthodes obsolètes par un appel à la nouvelle`Builder.buildStrict()` méthode. Cet exemple en spécifie uneAWS KMS key comme clé d'encapsulation, mais la`Builder.buildStrict()` méthode peut prendre une liste de plusieursAWS KMS keys.

```
// Create a master key provider in strict mode
// Replace the example key ARN with a valid one from your Compte AWS.
String awsKmsKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

KmsMasterKeyProvider masterKeyProvider = KmsMasterKeyProvider.builder()
    .buildStrict(awsKmsKey);
```

Python

Cet exemple représente le code d'une application qui utilise la version 1.4.1 duKit SDK de chiffrement AWS pour Python. Ce code utilise`KMSMasterKeyProvider`, qui est obsolète dans la version 1.7. x et supprimé de la version 2.0. x. Lors du déchiffrement, il utilise toute clé de données cryptée sans tenir compte deAWS KMS key celleAWS KMS keys que vous avez spécifiée.

Notez qu'`KMSMasterKey`il n'est ni obsolète ni supprimé. Lors du chiffrement et du déchiffrement, il utilise uniquement ce queAWS KMS key vous spécifiez.

```
# Create a master key provider
# Replace the example key ARN with a valid one
key_1 = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
key_2 = "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321"
```

```
aws_kms_master_key_provider = KMSMasterKeyProvider(
   key_ids=[key_1, key_2]
)
```

Cet exemple représente le code d'une application qui utilise la version 1.7. x duKit SDK de chiffrement AWS pour Python. Pour un exemple complet, consultez basic_encryption.py.

Pour passer à un fournisseur de clé principale en mode strict, ce code remplace l'appel à`KMSMasterKeyProvider()` par un appel à`StrictAwsKmsMasterKeyProvider()`.

```
# Create a master key provider in strict mode
# Replace the example key ARNs with valid values from your Compte AWS
key_1 = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
key_2 = "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321"

aws_kms_master_key_provider = StrictAwsKmsMasterKeyProvider(
    key_ids=[key_1, key_2]
)
```

AWS Encryption CLI

Cet exemple montre comment chiffrer et déchiffrer à l'aide de la version 1.1.7 ou antérieure de la CLI deAWS chiffrement.

Dans les versions 1.1.7 et antérieures, lors du chiffrement, vous spécifiez une ou plusieurs clés principales (ou clés d'encapsulation), telles qu'uneAWS KMS key. Lors du déchiffrement, vous ne pouvez spécifier aucune clé d'encapsulation à moins d'utiliser un fournisseur de clé principale personnalisé. La CLI deAWS chiffrement peut utiliser n'importe quelle clé d'encapsulation qui a chiffré une clé de données.

```
\\ Replace the example key ARN with a valid one
$ keyArn=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

\\ Encrypt your plaintext data
$ aws-encryption-cli --encrypt \
                     --input hello.txt \
                     --master-keys key=$keyArn \
                     --metadata-output ~/metadata \
```

```
                    --encryption-context purpose=test \
                    --output .

\\ Decrypt your ciphertext
$ aws-encryption-cli --decrypt \
                    --input hello.txt.encrypted \
                    --encryption-context purpose=test \
                    --metadata-output ~/metadata \
                    --output .
```

Cet exemple montre comment chiffrer et déchiffrer à l'aide de la version 1.7 de l'interface de ligne de commande deAWS chiffrement. x ou version ultérieure. Pour des exemples complets, reportez-vous à la sectionExemples deAWSCLI de chiffrement.

Le`--master-keys` paramètre est obsolète dans la version 1.7. x et supprimé dans la version 2.0. x. Il est remplacé par le`--wrapping-keys` paramètre by, qui est requis dans les commandes de chiffrement et de déchiffrement. Ce paramètre prend en charge le mode strict et le mode découverte. Le mode strict est uneAWS Encryption SDK bonne pratique qui garantit que vous utilisez la clé d'encapsulation que vous souhaitez.

Pour passer en mode strict, utilisez l'attribut key du`--wrapping-keys` paramètre pour spécifier une clé d'encapsulation lors du chiffrement et du déchiffrement.

```
\\ Replace the example key ARN with a valid value
$ keyArn=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

\\ Encrypt your plaintext data
$ aws-encryption-cli --encrypt \
                    --input hello.txt \
                    --wrapping-keys key=$keyArn \
                    --metadata-output ~/metadata \
                    --encryption-context purpose=test \
                    --output .

\\ Decrypt your ciphertext
$ aws-encryption-cli --decrypt \
                    --input hello.txt.encrypted \
                    --wrapping-keys key=$keyArn \
                    --encryption-context purpose=test \
                    --metadata-output ~/metadata \
                    --output .
```

Migration vers le mode découverte

À partir de la version 1.7. x, il est recommandé d'AWS Encryption SDKutiliser le mode strict pour les fournisseurs deAWS KMS clés principales, c'est-à-dire de spécifier des clés d'encapsulage lors du chiffrement et du déchiffrement. Vous devez toujours spécifier des clés d'encapsulation lors du chiffrement. Mais il existe des situations dans lesquelles la spécification des ARN clésAWS KMS keys pour le déchiffrement n'est pas pratique. Par exemple, si vous utilisez des alias pour vous identifierAWS KMS keys lors du chiffrement, vous perdez l'avantage des alias si vous devez répertorier les ARN clés lors du déchiffrement. De plus, étant donné que les fournisseurs de clés principales en mode découverte se comportent comme les fournisseurs de clés principales d'origine, vous pouvez les utiliser temporairement dans le cadre de votre stratégie de migration, puis passer à des fournisseurs de clés principales en mode strict ultérieurement.

Dans de tels cas, vous pouvez utiliser des fournisseurs de clés principales en mode découverte. Ces fournisseurs de clés principales ne vous permettent pas de spécifier des clés d'encapsulation. Vous ne pouvez donc pas les utiliser pour le chiffrement. Lors du déchiffrement, ils peuvent utiliser n'importe quelle clé d'encapsulation qui a chiffré une clé de données. Mais contrairement aux anciens fournisseurs de clés principales, qui se comportent de la même manière, vous les créez explicitement en mode découverte. Lorsque vous utilisez des fournisseurs de clés principales en mode découverte, vous pouvez limiter les clés d'encapsulation pouvant être utilisées à ces fournisseurs en particulierComptes AWS. Ce filtre de découverte est facultatif, mais c'est une bonne pratique que nous recommandons. Pour plus d'informations surAWS les partitions et les comptes, consultez Amazon Resource Names dans le Références générales AWS.

Les exemples suivants créent un fournisseur de cléAWS KMS principale en mode strict pour le chiffrement et un fournisseur de cléAWS KMS principale en mode découverte pour le déchiffrement. Le fournisseur de clé principale en mode découverte utilise un filtre de découverte pour limiter les clés d'encapsulation utilisées pour le déchiffrement à la`aws` partition et à un exemple particulierComptes AWS. Bien que le filtre de compte ne soit pas nécessaire dans cet exemple très simple, il s'agit d'une bonne pratique qui s'avère très utile lorsqu'une application chiffre des données et qu'une autre application les déchiffre.

Java

> Cet exemple représente le code d'une application qui utilise la version 1.7. x ou version ultérieure duKit SDK de chiffrement AWS pour Java. Pour un exemple complet, consultez DiscoveryDecryptionExample.java.

Pour instancier un fournisseur de clé principale en mode strict à des fins de chiffrement, cet exemple utilise la`Builder.buildStrict()` méthode. Pour instancier un fournisseur de clé principale en mode découverte à des fins de déchiffrement, il utilise la`Builder.buildDiscovery()` méthode. La`Builder.buildDiscovery()` méthode prend un`DiscoveryFilter` qui limite les deuxAWS Encryption SDKAWS KMS keys à laAWS partition et aux comptes spécifiés.

```
// Create a master key provider in strict mode for encrypting
// Replace the example alias ARN with a valid one from your Compte AWS.
String awsKmsKey = "arn:aws:kms:us-west-2:111122223333:alias/ExampleAlias";

KmsMasterKeyProvider encryptingKeyProvider = KmsMasterKeyProvider.builder()
    .buildStrict(awsKmsKey);

// Create a master key provider in discovery mode for decrypting
// Replace the example account IDs with valid values.
DiscoveryFilter accounts = new DiscoveryFilter("aws", Arrays.asList("111122223333",
 "444455556666"));

KmsMasterKeyProvider decryptingKeyProvider = KmsMasterKeyProvider.builder()
    .buildDiscovery(accounts);
```

Python

Cet exemple représente le code d'une application qui utilise la version 1.7. x ou version ultérieure duKit SDK de chiffrement AWS pour Python. Pour un exemple complet, consultez discovery_kms_provider.py.

Pour créer un fournisseur de clé principale en mode strict à des fins de chiffrement, cet exemple utilise`StrictAwsKmsMasterKeyProvider`. Pour créer un fournisseur de clé principale en mode découverte à des fins de déchiffrement, il utilise`DiscoveryAwsKmsMasterKeyProvider` with a`DiscoveryFilter` qui limite leAWS Encryption SDK àAWS KMS keys à laAWS partition et aux comptes spécifiés.

```
# Create a master key provider in strict mode
# Replace the example key ARN and alias ARNs with valid values from your Compte AWS.
key_1 = "arn:aws:kms:us-west-2:111122223333:alias/ExampleAlias"
key_2 = "arn:aws:kms:us-
west-2:444455556666:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d"

aws_kms_master_key_provider = StrictAwsKmsMasterKeyProvider(
```

```
    key_ids=[key_1, key_2]
)

# Create a master key provider in discovery mode for decrypting
# Replace the example account IDs with valid values
accounts = DiscoveryFilter(
    partition="aws",
    account_ids=["111122223333", "444455556666"]
)
aws_kms_master_key_provider = DiscoveryAwsKmsMasterKeyProvider(
        discovery_filter=accounts
)
```

AWS Encryption CLI

Cet exemple montre comment chiffrer et déchiffrer à l'aide de la version 1.7 de l'interface de ligne de commande deAWS chiffrement. x ou version ultérieure. À partir de la version 1.7. x, le`--wrapping-keys` paramètre est requis lors du chiffrement et du déchiffrement. Le`--wrapping-keys` paramètre prend en charge le mode strict et le mode découverte. Pour des exemples complets, reportez-vous à la sectionthe section called “Exemples”.

Lors du chiffrement, cet exemple spécifie une clé d'encapsulation, qui est requise. Lors du déchiffrement, il choisit explicitement le mode de découverte en utilisant l'`discovery`attribut du`--wrapping-keys` paramètre avec une valeur de`true`.

Pour limiter les clés d'encapsulationAWS Encryption SDK pouvant être utilisées en mode découverte à celles-ci en particulierComptes AWS, cet exemple utilise les`discovery-account` attributs`discovery-partition` et du`--wrapping-keys` paramètre. Ces attributs facultatifs ne sont valides que lorsque l'`discovery`attribut est défini sur`true`. Vous devez utiliser les`discovery-account` attributs`discovery-partition` et ensemble ; aucun des deux n'est valide seul.

```
\\ Replace the example key ARN with a valid value
$ keyAlias=arn:aws:kms:us-west-2:111122223333:alias/ExampleAlias

\\ Encrypt your plaintext data
$ aws-encryption-cli --encrypt \
                     --input hello.txt \
                     --wrapping-keys key=$keyAlias \
                     --metadata-output ~/metadata \
                     --encryption-context purpose=test \
```

```
                    --output .

\\ Decrypt your ciphertext
\\ Replace the example account IDs with valid values
$ aws-encryption-cli --decrypt \
                    --input hello.txt.encrypted \
                    --wrapping-keys discovery=true \
                                    discovery-partition=aws \
                                    discovery-account=111122223333 \
                                    discovery-account=444455556666 \
                    --encryption-context purpose=test \
                    --metadata-output ~/metadata \
                    --output .
```

Mise à jour en coursAWS KMSporte-clés

Dans laAWS KMSporte-clés dans leKit SDK de chiffrement AWS pour C, leAWS Encryption SDKpour .NET, et leKit SDK de chiffrement AWS pour JavaScriptsoutenirbonnes pratiquesen vous permettant de spécifier des clés d'encapsulation lors du chiffrement et du déchiffrement. Si vous créez unAWS KMSporte-clés Discovery, vous le faites de manière explicite.

> **Note**
>
> La version la plus ancienne duAWS Encryption SDKpour .NET, c'est la version 3.0.x. Toutes les versions duAWS Encryption SDKpour .NET, prenez en charge les meilleures pratiques de sécurité introduites dans la version 2.0.xde laAWS Encryption SDK. Vous pouvez passer à la dernière version en toute sécurité sans aucune modification du code ou des données.

Lorsque vous effectuez une mise à jour vers la dernière version 1.xversion duAWS Encryption SDK, vous pouvez utiliserfiltre de découvertepour limiter les clés d'encapsulation qu'unAWS KMSporte-clés DiscoveryouAWS KMSporte-clés Découverte régionaleutilisations lors du déchiffrement vers ceux en particulierComptes AWS. Le filtrage d'un trousseau de découverte est uneAWS Encryption SDK bonne pratique.

Les exemples de cette section vous montreront comment ajouter le filtre de découverte à unAWS KMSporte-clés de découverte régional.

En savoir plus sur la migration

Pour toutesAWS Encryption SDKutilisateurs, découvrez comment définir votre politique d'engagement dansthe section called “Définition de votre politique d'engagement”.

PourKit SDK de chiffrement AWS pour Java,Kit SDK de chiffrement AWS pour Python, etAWSUtilisateurs de la CLI de chiffrement, découvrez la mise à jour requise pour les fournisseurs de clés principales dansthe section called “Mise à jourAWS KMS des fournisseurs de clés principales”.

Vous pouvez avoir du code comme celui-ci dans votre application. Cet exemple crée unAWS KMStrousseau de découverte régional qui ne peut utiliser les clés d'enrobage que dans la Région USA Ouest (Oregon) (us-west-2). Cet exemple représente le code dansAWS Encryption SDKversions antérieures à 1.7.x. Cependant, il est toujours valide dans les versions 1.7.xet plus tard.

C

```
struct aws_cryptosdk_keyring *kms_regional_keyring =
 Aws::Cryptosdk::KmsKeyring::Builder()
       .WithKmsClient(create_kms_client(Aws::Region::US_WEST_2)).BuildDiscovery());
```

JavaScript Browser

```
const clientProvider = getClient(KMS, { credentials })

const discovery = true
const clientProvider = limitRegions(['us-west-2'], getKmsClient)
const keyring = new KmsKeyringBrowser({ clientProvider, discovery })
```

JavaScript Node.js

```
const discovery = true
const clientProvider = limitRegions(['us-west-2'], getKmsClient)
const keyring = new KmsKeyringNode({ clientProvider, discovery })
```

À partir de la version 1.7.x, vous pouvez ajouter un filtre de découverte à n'importe quelAWS KMSporte-clés Discovery. Ce filtre de découverte limiteAWS KMS keysque leAWS Encryption SDKpeut être utilisé pour le déchiffrement de ceux qui se trouvent dans la partition et les comptes spécifiés. Avant d'utiliser ce code, modifiez la partition, si nécessaire, et remplacez les exemples d'ID de compte par des identifiants valides.

C

Pour obtenir un exemple complet, consultezkms_discovery.cpp.

```
std::shared_ptr<KmsKeyring::DiscoveryFilter> discovery_filter(
    KmsKeyring::DiscoveryFilter::Builder("aws")
        .AddAccount("111122223333")
        .AddAccount("444455556666")
        .Build());

struct aws_cryptosdk_keyring *kms_regional_keyring =
 Aws::Cryptosdk::KmsKeyring::Builder()

 .WithKmsClient(create_kms_client(Aws::Region::US_WEST_2)).BuildDiscovery(discovery_filter))
```

JavaScript Browser

```
const clientProvider = getClient(KMS, { credentials })

const discovery = true
const clientProvider = limitRegions(['us-west-2'], getKmsClient)
const keyring = new KmsKeyringBrowser(clientProvider, {
    discovery,
    discoveryFilter: { accountIDs: ['111122223333', '444455556666'], partition:
 'aws' }
})
```

JavaScript Node.js

Pour obtenir un exemple complet, consultezkms_filtered_discovery.ts.

```
const discovery = true
const clientProvider = limitRegions(['us-west-2'], getKmsClient)
const keyring = new KmsKeyringNode({
    clientProvider,
    discovery,
    discoveryFilter: { accountIDs: ['111122223333', '444455556666'], partition:
 'aws' }
})
```

Définition de votre politique d'engagement

Un engagement clé garantit que vos données cryptées sont toujours déchiffrées dans le même texte brut. Pour fournir cette propriété de sécurité, à partir de la version 1.7. x,AWS Encryption SDK utilise de nouvelles suites d'algorithmes avec un engagement clé. Pour déterminer si vos données sont cryptées et décryptées avec un engagement clé, utilisez le paramètre de configuration de la politique d'engagement. Le chiffrement et le déchiffrement des données avec un engagement clé constituent une AWS Encryption SDKbonne pratique.

La définition d'une politique d'engagement est un élément important de la deuxième étape du processus de migration, à savoir la migration à partir de la dernière version 1. x versions desAWS Encryption SDK deux versions 2.0. x et versions ultérieures. Après avoir défini et modifié votre politique d'engagement, assurez-vous de tester minutieusement votre application avant de la déployer en production. Pour obtenir des conseils sur la migration, consultezComment migrer et déployerAWS Encryption SDK.

Le paramètre de politique d'engagement possède trois valeurs valides dans les versions 2.0. x et versions ultérieures. Dans la dernière version 1. x versions (à partir de la version 1.7. x), seul`ForbidEncryptAllowDecrypt` est valide.

- `ForbidEncryptAllowDecrypt`— IlsAWS Encryption SDK ne peuvent pas crypter avec un engagement clé. Il peut déchiffrer des textes chiffrés avec ou sans saisie de clé.

  Dans la dernière version 1. versions x, c'est la seule valeur valide. Cela garantit que vous ne chiffrez pas avec un engagement clé tant que vous n'êtes pas complètement prêt à le déchiffrer avec un engagement clé. La définition explicite de cette valeur empêche la modification automatique de votre politique d'engagement`require-encrypt-require-decrypt` lors de la mise à niveau vers les versions 2.0. x ou version ultérieure. Au lieu de cela, vous pouvez migrer votre politique d'engagement par étapes.
- `RequireEncryptAllowDecrypt`— Il crypteAWS Encryption SDK toujours avec un engagement clé. Il peut déchiffrer des textes chiffrés avec ou sans saisie de clé. Cette valeur a été ajoutée dans la version dans la version dans la version dans la version x.
- `RequireEncryptRequireDecrypt`— Il chiffre et déchiffreAWS Encryption SDK toujours avec un engagement clé. Cette valeur a été ajoutée dans la version dans la version dans la version dans la version x. Il s'agit de la valeur par défaut dans les versions 2.0. x et versions ultérieures.

Dans la dernière version 1. versions x, la seule valeur de politique d'engagement valide est`ForbidEncryptAllowDecrypt`. Après la migration vers la version dans la version dans la version dans la la x ou version ultérieure, vous pouvez modifier votre politique d'engagement par étapes à mesure que vous êtes prêt. Ne mettez pas à jour votre politique d'engagement`RequireEncryptRequireDecrypt` tant que vous n'êtes pas certain qu'aucun message n'est crypté sans engagement de clé.

Ces exemples vous montrent comment définir votre politique d'engagement dans la dernière version 1. versions x et versions 2.0. x et versions ultérieures. La technique dépend de votre langage de programmation.

En savoir plus sur la migration

PourKit SDK de chiffrement AWS pour Java,Kit SDK de chiffrement AWS pour Python, et la CLI deAWS chiffrement, découvrez les modifications requises pour les fournisseurs de clés principales dansthe section called “Mise à jourAWS KMS des fournisseurs de clés principales”.

PourKit SDK de chiffrement AWS pour C etKit SDK de chiffrement AWS pour JavaScript, découvrez une mise à jour facultative des porte-clés dansMise à jour en coursAWS KMSporte-clés.

Comment définir votre politique d'engagement

La technique que vous utilisez pour définir votre politique d'engagement diffère légèrement d'une implémentation linguistique à l'autre. Ces exemples montrent comment procéder. Avant de modifier votre politique d'engagement, passez en revue l'approche en plusieurs étapes dansComment migrer et déployer.

C

À partir de la version 1.7. x deKit SDK de chiffrement AWS pour C, vous utilisez la`aws_cryptosdk_session_set_commitment_policy` fonction pour définir la politique d'engagement pour vos sessions de chiffrement et de déchiffrement. La politique d'engagement que vous définissez s'applique à toutes les opérations de chiffrement et de déchiffrement appelées au cours de cette session.

Les`aws_cryptosdk_session_new_from_cmm` fonctions`aws_cryptosdk_session_new_from_keyring` et sont obsolètes dans la version 1.7. x et supprimé dans la version 2.0. x. Ces fonctions sont remplacées par`aws_cryptosdk_session_new_from_keyring_2` des`aws_cryptosdk_session_new_from_cmm_2` fonctions qui renvoient une session.

Lorsque vous utilisez le`aws_cryptosdk_session_new_from_keyring_2` et`aws_cryptosdk_session_new_from_cmm_2` dans la dernière version 1. versions x, vous devez appeler la`aws_cryptosdk_session_set_commitment_policy` fonction avec la valeur de la politique d'`COMMITMENT_POLICY_FORBID_ENCRYPT_ALLOW_DECRYPT`engagement. Dans les versions dans les versions dans la 2.0 x et versions ultérieures, l'appel de cette fonction est facultatif et prend toutes les valeurs valides. La politique d'engagement par défaut pour les versions 2.0. x et les versions ultérieures sont`COMMITMENT_POLICY_REQUIRE_ENCRYPT_REQUIRE_DECRYPT`.

Pour obtenir un exemple complet, veuillez consulter string.cpp.

```
/* Load error strings for debugging */
aws_cryptosdk_load_error_strings();

/* Create an AWS KMS keyring */
const char * key_arn = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
struct aws_cryptosdk_keyring *kms_keyring =
 Aws::Cryptosdk::KmsKeyring::Builder().Build(key_arn);

/* Create an encrypt session with a CommitmentPolicy setting */
struct aws_cryptosdk_session *encrypt_session =
 aws_cryptosdk_session_new_from_keyring_2(
    alloc, AWS_CRYPTOSDK_ENCRYPT, kms_keyring);

aws_cryptosdk_keyring_release(kms_keyring);
aws_cryptosdk_session_set_commitment_policy(encrypt_session,
    COMMITMENT_POLICY_FORBID_ENCRYPT_ALLOW_DECRYPT);

...
/* Encrypt your data */

size_t plaintext_consumed_output;
aws_cryptosdk_session_process(encrypt_session,
                              ciphertext_output,
                              ciphertext_buf_sz_output,
                              ciphertext_len_output,
                              plaintext_input,
                              plaintext_len_input,
                              &plaintext_consumed_output)
...
```

```
/* Create a decrypt session with a CommitmentPolicy setting */

struct aws_cryptosdk_keyring *kms_keyring =
 Aws::Cryptosdk::KmsKeyring::Builder().Build(key_arn);
struct aws_cryptosdk_session **decrypt_session =
 *aws_cryptosdk_session_new_from_keyring_2(
        alloc, AWS_CRYPTOSDK_DECRYPT, kms_keyring);
aws_cryptosdk_keyring_release(kms_keyring);
aws_cryptosdk_session_set_commitment_policy(decrypt_session,
        COMMITMENT_POLICY_FORBID_ENCRYPT_ALLOW_DECRYPT);

/* Decrypt your ciphertext */
size_t ciphertext_consumed_output;
aws_cryptosdk_session_process(decrypt_session,
                              plaintext_output,
                              plaintext_buf_sz_output,
                              plaintext_len_output,
                              ciphertext_input,
                              ciphertext_len_input,
                              &ciphertext_consumed_output)
```

C# / .NET

La`require-encrypt-require-decrypt` valeur est la politique d'engagement par défaut dans toutes les versions deAWS Encryption SDK pour .NET. Vous pouvez l'lorsqu'il n'utilisez pas l'lorsqu'il n'utilisez pas l'lorsqu'il n'utilisez pas l'lorsqu'il n'utilisez pas l'lorsqu'il n'utilisez pas l' Toutefois, si vous utilisez leAWS Encryption SDK pour .NET pour déchiffrer un texte chiffré qui a été chiffré par une implémentation dans un autre langage de l'engagementAWS Encryption SDK sans clé, vous devez remplacer la valeur de la politique d'engagement par`REQUIRE_ENCRYPT_ALLOW_DECRYPT` ou`FORBID_ENCRYPT_ALLOW_DECRYPT`. Dans le cas contraire, les tentatives de déchiffrement du texte chiffré échoueront.

Dans leAWS Encryption SDK pour .NET, vous définissez la politique d'engagement sur une instance deAWS Encryption SDK. Instanciez un`AwsEncryptionSdkConfig` objet avec un`CommitmentPolicy` paramètre et utilisez l'objet de configuration pour créer l'AWS Encryption SDKinstance. Appelez ensuite les`Decrypt()` méthodes`Encrypt()` et de l'AWS Encryption SDKinstance configurée.

Cet exemple définit la politique d'engagement à`require-encrypt-allow-decrypt`.

```
// Instantiate the material providers
var materialProviders =
```

```
 AwsCryptographicMaterialProvidersFactory.CreateDefaultAwsCryptographicMaterialProviders();

// Configure the commitment policy on the AWS Encryption SDK instance
var config = new AwsEncryptionSdkConfig
{
    CommitmentPolicy = CommitmentPolicy.REQUIRE_ENCRYPT_ALLOW_DECRYPT
};
var encryptionSdk = AwsEncryptionSdkFactory.CreateAwsEncryptionSdk(config);

string keyArn = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

var encryptionContext = new Dictionary<string, string>()
{
    {"purpose", "test"}encryptionSdk
};

var createKeyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = keyArn
};
var keyring = materialProviders.CreateAwsKmsKeyring(createKeyringInput);

// Encrypt your plaintext data
var encryptInput = new EncryptInput
{
    Plaintext = plaintext,
    Keyring = keyring,
    EncryptionContext = encryptionContext
};
var encryptOutput = encryptionSdk.Encrypt(encryptInput);

// Decrypt your ciphertext
var decryptInput = new DecryptInput
{
    Ciphertext = ciphertext,
    Keyring = keyring
};
var decryptOutput = encryptionSdk.Decrypt(decryptInput);
```

AWS Encryption CLI

Pour définir une politique d'engagement dans la CLI deAWS chiffrement, utilisez le`--commitment-policy` paramètre. Ce paramètre a été introduit dans la version 1.8. x.

Dans la dernière version 1. version x, lorsque vous utilisez le`--wrapping-keys` paramètre dans une`--decrypt` commande`--encrypt` ou, un`--commitment-policy` paramètre avec la`forbid-encrypt-allow-decrypt` valeur est requis. Dans le cas contraire, le`--commitment-policy` paramètre n'est pas valide.

Dans les versions 2.1. x et versions ultérieures, le`--commitment-policy` paramètre est facultatif et prend par défaut la`require-encrypt-require-decrypt` valeur, qui ne chiffrera ni ne déchiffrera aucun texte chiffré sans saisie de clé. Cependant, nous vous recommandons de définir la politique d'engagement à l'aide de la maintenance et du dépannage.

Cet exemple définit la politique d'engagement. Il utilise également le`--wrapping-keys` paramètre qui remplace le`--master-keys` paramètre à partir de la version 1.8. x. Pour plus de détails, consultez the section called “Mise à jourAWS KMS des fournisseurs de clés principales”. Pour des exemples complets, reportez-vous à la sectionExemples deAWSCLI de chiffrement.

```
\\ To run this example, replace the fictitious key ARN with a valid value.
$ keyArn=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

\\ Encrypt your plaintext data - no change to algorithm suite used
$ aws-encryption-cli --encrypt \
                     --input hello.txt \
                     --wrapping-keys key=$keyArn \
                     --commitment-policy forbid-encrypt-allow-decrypt \
                     --metadata-output ~/metadata \
                     --encryption-context purpose=test \
                     --output .

\\ Decrypt your ciphertext - supports key commitment on 1.7 and later
$ aws-encryption-cli --decrypt \
                     --input hello.txt.encrypted \
                     --wrapping-keys key=$keyArn \
                     --commitment-policy forbid-encrypt-allow-decrypt \
                     --encryption-context purpose=test \
                     --metadata-output ~/metadata \
                     --output .
```

Java

À partir de la version 1.7. x duKit SDK de chiffrement AWS pour Java, vous définissez la politique d'engagement sur votre instance de`AwsCrypto`, l'objet qui représente le`AWS Encryption SDK` client. Ce paramètre de politique d'engagement s'applique à toutes les opérations de chiffrement et de déchiffrement appelées sur ce client.

Le`AwsCrypto()` constructeur est obsolète dans la dernière version 1. les versions x duKit SDK de chiffrement AWS pour Java et sont supprimées dans la version 2.0. x. Elle est remplacée par une nouvelle`Builder` classe, une nouvelle`Builder.withCommitmentPolicy()` méthode et le type`CommitmentPolicy` énuméré.

Dans la dernière version 1. versions x, la`Builder` classe nécessite la`Builder.withCommitmentPolicy()` méthode et l'`CommitmentPolicy.ForbidEncryptAllowDecrypt`argument. À partir de la version dans la version dans la version x, la`Builder.withCommitmentPolicy()` méthode est facultative ; la valeur par défaut est`CommitmentPolicy.RequireEncryptRequireDecrypt`.

Pour un exemple complet, consultez SetCommitmentPolicyExample.java.

```
// Instantiate the client
final AwsCrypto crypto = AwsCrypto.builder()
    .withCommitmentPolicy(CommitmentPolicy.ForbidEncryptAllowDecrypt)
    .build();

// Create a master key provider in strict mode
String awsKmsKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

KmsMasterKeyProvider masterKeyProvider = KmsMasterKeyProvider.builder()
    .buildStrict(awsKmsKey);

// Encrypt your plaintext data
CryptoResult<byte[], KmsMasterKey> encryptResult = crypto.encryptData(
    masterKeyProvider,
    sourcePlaintext,
    encryptionContext);
byte[] ciphertext = encryptResult.getResult();

// Decrypt your ciphertext
CryptoResult<byte[], KmsMasterKey> decryptResult = crypto.decryptData(
        masterKeyProvider,
```

```
        ciphertext);
byte[] decrypted = decryptResult.getResult();
```

JavaScript

À partir de la version 1.7. x deKit SDK de chiffrement AWS pour JavaScript, vous pouvez définir la politique d'engagement lorsque vous appelez la nouvelle`buildClient` fonction qui instancie unAWS Encryption SDK client. La`buildClient` fonction prend une valeur énumérée qui représente votre politique d'engagement. Il renvoie des mises à jour`encrypt` et`decrypt` des fonctions qui appliquent votre politique d'engagement lorsque vous cryptez et déchiffrez.

Dans la dernière version 1. versions x, la`buildClient` fonction nécessite l'`CommitmentPolicy.FORBID_ENCRYPT_ALLOW_DECRYPT`argument. À partir de la version dans la version dans la version x, l'argument de la politique d'engagement est facultatif et la valeur par défaut est`CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT`.

Le code pour Node.js et le navigateur sont identiques à cette fin, sauf que le navigateur a besoin d'une instruction pour définir les informations d'identification.

L'exemple suivant chiffre des données à l'aide d'unAWS KMS jeu de clés. La nouvelle`buildClient` fonction définit la politique d'engagement à`FORBID_ENCRYPT_ALLOW_DECRYPT` la valeur par défaut la plus récente de 1. 2 versions. La mise à niveau`encrypt` et`decrypt` les fonctions`buildClient` renvoyées appliquent la politique d'engagement que vous avez définie.

```
import { buildClient } from '@aws-crypto/client-node'
const { encrypt, decrypt } =
 buildClient(CommitmentPolicy.FORBID_ENCRYPT_ALLOW_DECRYPT)

// Create an AWS KMS keyring
const generatorKeyId = 'arn:aws:kms:us-west-2:111122223333:alias/ExampleAlias'
const keyIds = ['arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab']
const keyring = new KmsKeyringNode({ generatorKeyId, keyIds })

// Encrypt your plaintext data
const { ciphertext } = await encrypt(keyring, plaintext, { encryptionContext:
 context })

// Decrypt your ciphertext
const { decrypted, messageHeader } = await decrypt(keyring, ciphertext)
```

Python

À partir de la version 1.7. x duKit SDK de chiffrement AWS pour Python, vous définissez la politique d'engagement sur votre instance de`EncryptionSDKClient`, un nouvel objet qui représente leAWS Encryption SDK client. La politique d'engagement que vous définissez s'applique à tous`encrypt` les`decrypt` appels utilisant cette instance du client.

Dans la dernière version 1. versions x, le`EncryptionSDKClient` constructeur a besoin de la valeur`CommitmentPolicy.FORBID_ENCRYPT_ALLOW_DECRYPT` énumérée. À partir de la version dans la version dans la version x, l'argument de la politique d'engagement est facultatif et la valeur par défaut est`CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT`.

Cet exemple utilise le nouveau`EncryptionSDKClient` constructeur et définit la politique d'engagement sur le 1.7. x valeur par défaut. Le constructeur instancie un client qui représente leAWS Encryption SDK. Lorsque vous appelez les`stream` méthodes`encrypt`,`decrypt`, ou sur ce client, elles appliquent la politique d'engagement que vous avez définie. Cet exemple utilise également le nouveau constructeur de la`StrictAwsKmsMasterKeyProvider` classe, qui spécifieAWS KMS keys lors du chiffrement et du déchiffrement.

Pour un exemple complet, consultez set_commitment.py.

```
# Instantiate the client
client =
 aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.FORBID_ENCRYPT_AL

// Create a master key provider in strict mode
aws_kms_key = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
aws_kms_strict_master_key_provider = StrictAwsKmsMasterKeyProvider(
        key_ids=[aws_kms_key]
)

# Encrypt your plaintext data
ciphertext, encrypt_header = client.encrypt(
        source=source_plaintext,
        encryption_context=encryption_context,
        master_key_provider=aws_kms_strict_master_key_provider
)

# Decrypt your ciphertext
decrypted, decrypt_header = client.decrypt(
        source=ciphertext,
```

```
        master_key_provider=aws_kms_strict_master_key_provider
)
```

Résolution des problèmes de migration vers les dernières versions

Avant de mettre à jour votre application vers la version 2.0.xou une version ultérieure duAWS Encryption SDK, mise à jour vers la dernière version 1.xversion duAWS Encryption SDKet déployez-le complètement. Cela vous aidera à éviter la plupart des erreurs que vous pourriez rencontrer lors de la mise à jour vers les versions 2.0.xet plus tard. Pour des instructions détaillées, y compris des exemples, voirMigrer votreAWS Encryption SDK.

> **Important**
>
> Vérifiez que votre dernier 1.xversion 1.7.xou une version ultérieure duAWS Encryption SDK.

> **Note**
>
> AWSCLI de chiffrement : Références dans ce guide à la version 1.7.xde laAWS Encryption SDKappliquer à la version 1.8.xde laAWSCLI de chiffrement. Références dans ce guide à la version 2.0.xde laAWS Encryption SDKappliquer au 2.1.xde laAWSCLI de chiffrement.
> Les nouvelles fonctionnalités de sécurité ont été initialement publiées enAWSVersions CLI de chiffrement 1.7.xet 2.0.x. Cependant,AWSVersion 1.8 de la CLI de chiffrementxremplace la version 1.7.xetAWSCryptage CLI 2.1.xremplace 2.0.x. Pour plus d'informations, consultez laAvis de sécuritédans leaws-encryption-sdk-cliréférentiel sur GitHub.

Cette rubrique est conçue pour vous aider à identifier et à résoudre les erreurs les plus courantes que vous pouvez rencontrer.

Rubriques

- Objets obsolètes ou supprimés
- Conflit de configuration Politique d'engagement et suite d'algorithmes
- Conflit de configuration Politique d'engagement et texte chiffré
- La validation des engagements clés a échoué
- Autres échecs de chiffrement

- Autres échecs de déchiffrement
- Considérations relatives à l'

Objets obsolètes ou supprimés

version 2.0.xinclut plusieurs modifications majeures, notamment la suppression des anciens constructeurs, méthodes, fonctions et classes qui étaient obsolètes dans la version 1.7.x. Pour éviter les erreurs de compilation, les erreurs d'importation, les erreurs de syntaxe et les erreurs de symbole introuvable (en fonction de votre langage de programmation), passez d'abord à la dernière version 1.xversion duAWS Encryption SDKpour votre langage de programmation. (Il doit s'agir de la version 1.7.xou plus tard.) Lors de l'utilisation de la dernière version 1.xversion, vous pouvez commencer à utiliser les éléments de remplacement avant que les symboles d'origine ne soient supprimés.

Si vous devez effectuer une mise à niveau vers la version 2.0.xou plus tard immédiatement,consulter le changelogpour votre langage de programmation, et remplacez les anciens symboles par les symboles recommandés par le journal des modifications.

Conflit de configuration Politique d'engagement et suite d'algorithmes

Si vous spécifiez une suite d'algorithmes qui entre en conflit avec votrepolitique d'engagement d', l'appel de chiffrement échoue en renvoyant unConflit deErreur.

Pour éviter ce type d'erreur, ne spécifiez pas de suite d'algorithmes. Par défaut, le plugin XAWS Encryption SDKchoisit l'algorithme le plus sécurisé compatible avec votre politique d'engagement. Toutefois, si vous devez spécifier une suite d'algorithmes, par exemple sans signature, veillez à choisir une suite d'algorithmes compatible avec votre politique d'engagement.

Politique d'engagement d'	Suites d'algorithmes compatibles
ForbidEncryptAllowDecrypt	Toute suite d'algorithmessansun engagement clé, tel que : `AES_256_GCM_IV12_TAG16_HKDF_SHA384_ECDSA_P384` (03 78 x) (avec signature) `AES_256_GCM_IV12_TAG16_HKDF_SHA256` (01 78 x) (sans signature)

Politique d'engagement d'	Suites d'algorithmes compatibles
RequireEncryptAllowDecrypt RequireEncryptRequireDecrypt	Toute suite d'algorithmesavecun engagement clé, tel que : `AES_256_GCM_HKDF_SHA512_COMMIT_KEY_ECDSA_P384` (05 78 x) (avec signature) `AES_256_GCM_HKDF_SHA512_COMMIT_KEY` (04 78 x) (sans signature)

Si vous rencontrez cette erreur alors que vous n'avez pas spécifié de suite d'algorithmes, il est possible que la suite d'algorithmes en conflit ait été choisie par votregestionnaire de matériel cryptographique(CM). Le CMM par défaut ne sélectionnera pas une suite d'algorithmes conflictuelle, mais un CMM personnalisé le fera. Pour obtenir de l'aide, consultez la documentation de votre CMM personnalisé.

Conflit de configuration Politique d'engagement et texte chiffré

Dans la`RequireEncryptRequireDecrypt` politique d'engagement d'n'autorise pasAWS Encryption SDKpour décrypter un message qui a été crypté sansEngagé de durée. Si vous demandezAWS Encryption SDKpour déchiffrer un message sans engagement clé, il renvoie unConflit deErreur.

Pour éviter cette erreur, avant de définir le`RequireEncryptRequireDecrypt`politique d'engagement, assurez-vous que tous les textes chiffrés cryptés sans engagement de clé sont déchiffrés et recryptés avec engagement de clé, ou gérés par une autre application. Si vous rencontrez cette erreur, vous pouvez renvoyer une erreur concernant le texte chiffré en conflit ou modifier temporairement votre politique d'engagement en`RequireEncryptAllowDecrypt`.

Si vous rencontrez cette erreur parce que vous avez effectué une mise à niveau vers la version 2.0.xou une version ultérieure à partir d'une version antérieure à 1.7.xsans passer au préalable à la dernière version 1.xversion (version 1.7).xou plus tard), considérezrestauration de durée Xau plus tard 1.xversion et déploiement de cette version sur tous les hôtes avant la mise à niveau vers la version 2.0.xou plus tard. Pour obtenir de l'aide, veuillez consulter Comment migrer et déployerAWS Encryption SDK.

La validation des engagements clés a échoué

Lorsque vous déchiffrez des messages cryptés avec un engagement clé, vous pouvez obtenir unLa validation des engagements clés a échouéMessage d'erreur. Cela indique que l'appel de déchiffrement a échoué en raison d'une clé de données dans unMessage chiffdé den'est pas identique à la clé de données unique du message. En validant la clé de données lors du déchiffrement,Engagé de duréevous empêche de déchiffrer un message susceptible de générer plusieurs textes en clair.

Cette erreur indique que le message crypté que vous essayez de déchiffrer n'a pas été renvoyé parAWS Encryption SDK. Il peut s'agir d'un message créé manuellement ou du résultat d'une corruption de données. Si vous rencontrez cette erreur, votre application peut rejeter le message et continuer, ou arrêter de traiter les nouveaux messages.

Autres échecs de chiffrement

Le chiffrement peut échouer pour de multiples raisons. Vous ne pouvez pas utiliser un pluginAWS KMSporte-clés Discoveryou unfournisseur de clés principales en mode découvertepour crypter un message.

Assurez-vous de spécifier un porte-clés ou un fournisseur de clés principales avec les clés d'encapsulage dont vous disposezautorisation d'utilisationpour le chiffrement. Pour obtenir de l'aide concernant les autorisationsAWS KMS keys, voir.Affichage d'une politique de cléetDéterminer l'accès à unAWS KMS keydans leAWS Key Management ServiceManuel du développeur.

Autres échecs de déchiffrement

Si votre tentative de déchiffrement d'un message crypté échoue, cela signifie queAWS Encryption SDKn'a pas pu (ou n'a pas voulu) déchiffrer les clés de données cryptées du message.

Si vous avez utilisé un jeu de clés ou un fournisseur de clés principales qui spécifie des clés d'encapsulation,AWS Encryption SDKutilise uniquement les clés d'encapsulation que vous avez spécifiées. Vérifiez que vous utilisez les clés d'encapsulage que vous souhaitez et que vous disposezkms:Decryptautorisation sur au moins une des clés d'emballage. Si vous utilisez des XAWS KMS keys, comme solution de rechange, vous pouvez essayer de déchiffrer le message à l'aide d'unAWS KMSporte-clés Discoveryou unfournisseur de clés principales en mode découverte. Si l'opération aboutit, avant de renvoyer le texte en clair, vérifiez que la clé utilisée pour déchiffrer le message est une clé fiable.

Considérations relatives à l'

Si votre application ne parvient pas à chiffrer ou à déchiffrer les données, vous pouvez généralement résoudre le problème en mettant à jour les symboles de code, les trousseaux de clés, les fournisseurs de clés principales oupolitique d'engagement. Toutefois, dans certains cas, vous pouvez décider qu'il est préférable de restaurer votre application vers une version précédente deAWS Encryption SDK.

Si vous devez revenir en arrière, faites-le avec prudence. Versions duAWS Encryption SDKversion 1.7.xImpossible de déchiffrer le texte chiffré avecEngagé de durée.

- En arrière par rapport à la dernière version 1.xversion d'une version précédente duAWS Encryption SDKest généralement sûr. Vous devrez peut-être annuler les modifications que vous avez apportées à votre code pour utiliser des symboles et des objets qui ne sont pas pris en charge dans les versions précédentes.
- Une fois que vous avez commencé à crypter avec un engagement clé (en définissant votre politique d'engagement sur`RequireEncryptAllowDecrypt`) dans la version 2.0.xou version ultérieure, vous pouvez revenir à la version 1.7.x, mais pas vers une version antérieure. Versions duAWS Encryption SDKversion 1.7.xImpossible de déchiffrer le texte chiffré avecEngagé de durée.

Si vous activez accidentellement le chiffrement avec engagement de clé avant que tous les hôtes ne puissent le déchiffrer avec engagement de clé, il peut être préférable de poursuivre le déploiement plutôt que de revenir en arrière. Si les messages sont transitoires ou peuvent être supprimés en toute sécurité, vous pouvez envisager une restauration avec perte de messages. Si une restauration est requise, vous pouvez envisager d'écrire un outil qui déchiffre et rechiffre tous les messages.

Questions fréquentes (FAQ)

Quelle est la différence entre le kit AWS Encryption SDK et les autres kits SDK AWS ?

LeAWSKits SDKfournir des bibliothèques pour interagir avec Amazon Web Services (AWS), y compris AWS Key Management Service (AWS KMS). Certaines des implémentations linguistiques duAWS Encryption SDK, tels que leAWS Encryption SDKpour .NET, requiert toujours leAWSSDK dans le même langage de programmation. Les autres implémentations linguistiques requièrent les paramètres correspondantsAWSSDK uniquement lorsque vous utilisezAWS KMSclés de vos porte-clés ou de vos principaux fournisseurs de clés. Pour plus d'informations, consultez la rubrique concernant votre langage de programmation dans la pageLangages de programmation AWS Encryption SDK.

Vous pouvez utiliser le pluginAWSSDK avec lesquels interagirAWS KMS, notamment le chiffrement et le déchiffrement de petites quantités de données (jusqu'à 4 096 octets avec une

clé de chiffrement symétrique) et la génération de clés de données pour le chiffrement côté client. Toutefois, lorsque vous générez une clé de données, vous devez gérer l'intégralité du processus de chiffrement et de déchiffrement, y compris le chiffrement de vos données avec la clé de données en dehors deAWS KMS, en supprimant en toute sécurité la clé de données en texte brut, en stockant la clé de données chiffrée, puis en déchiffrant la clé de données et en déchiffrant vos données. LeAWS Encryption SDKgère ce processus pour vous.

LeAWS Encryption SDKfournit une bibliothèque qui chiffre et déchiffre les données à l'aide des normes et des bonnes pratiques du secteur. Il génère la clé de données, la chiffre sous les clés d'encapsulation que vous spécifiez et renvoie unMessage chiffré, un objet de données portable qui inclut les données chiffrées et les clés de données chiffrées dont vous avez besoin pour les déchiffrer. Lorsqu'il est temps de déchiffrer, vous transmettez le message chiffré et au moins une des clés d'encapsulation (facultatif), et leAWS Encryption SDKrenvoie vos données en texte brut.

Vous pouvez utiliserAWS KMS keyscomme clés d'encapsulation dans leAWS Encryption SDK, mais n'est pas nécessaire. Vous pouvez utiliser des clés de chiffrement que vous générez et celles de votre gestionnaire de clés ou de votre module de sécurité matérielle local. Vous pouvez utiliser le pluginAWS Encryption SDKMême si vous n'avez pas deAWS.

Fonctionnement du kitAWS Encryption SDKDifférent du client de chiffrement Amazon S3 ?

LeClient de chiffrement Amazon S3dans leAWSLes kits SDK fournissent le chiffrement et le déchiffrement pour les données que vous stockez dans Amazon Simple Storage Service (Amazon S3). Ces clients sont étroitement liés à Amazon S3 et sont conçus pour être utilisés uniquement avec les données qui y sont stockées.

Le kit AWS Encryption SDK fournit le chiffrement et le déchiffrement pour les données que vous pouvez stocker n'importe où. LeAWS Encryption SDKet le client de chiffrement Amazon S3 ne sont pas compatibles, car ils produisent des textes chiffrés avec différents formats de données.

Quels algorithmes de chiffrement sont pris en charge par le kit AWS Encryption SDK, et lequel est l'algorithme par défaut ?

LeAWS Encryption SDKutilise l'algorithme symétrique AES (Advanced Encryption Standard) en mode GCM (Galois/Counter Mode), appelé AES-GCM, pour chiffrer vos données. Il vous permet de choisir parmi plusieurs algorithmes symétriques et asymétriques pour chiffrer les clés de données qui chiffrent vos données.

Pour AES-GCM, la suite d'algorithmes par défaut est AES-GCM avec une clé de 256 bits, dérivation de clé (HKDF),signatures numériques, etEngagement clé.AWS Encryption SDKprend

également en charge également les clés de chiffrement et les algorithmes de chiffrement 192 et 128 bits sans signature numérique et sans engagement de clé.

Dans tous les cas, la longueur du vecteur d'initialisation est de 12 octets ; la longueur de la balise d'authentification est de 16 octets. Par défaut, le kit SDK utilise la clé de données en tant qu'entrée du kit HMAC. extract-and-expand fonction de dérivation de clés (HKDF) pour dériver la clé de chiffrement AES-GCM, et ajoute également une signature ECDSA (Elliptic Curve Digital Signature Algorithm).

Pour plus d'informations sur le choix de l'algorithme à utiliser, consultez Suites d'algorithmes prises en charge.

Pour obtenir des détails sur l'implémentation des algorithmes pris en charge, consultez Référence relative aux algorithmes du kit.

Comment est généré le vecteur d'initialisation et où est-il stocké ?

LeAWS Encryption SDKutilise une méthode déterministe pour construire une valeur de vecteur d'initialisation différente pour chaque cadre. Cette procédure garantit que les IV ne sont jamais répétées dans un message. (Avant la version 1.3.0 du kitKit SDK de chiffrement AWS pour Javaet l'Kit SDK de chiffrement AWS pour Python, leAWS Encryption SDKGénéré de façon aléatoire une valeur de vecteur d'initialisation pour chaque cadre.)

Le vecteur d'initialisation est stocké dans le message chiffré que leAWS Encryption SDKrenvoie. Pour plus d'informations, consultez le AWS Encryption SDK référence de format de message.

Comment est générée, chiffrée et déchiffrée chaque clé de données ?

La méthode dépend du porte-clés ou du fournisseur de clés principales que vous utilisez.

LeAWS KMSporte-clés et fournisseurs de clés principales dans leAWS Encryption SDKUtilisez le kitAWS KMS GenerateDataKeyOpération API pour générer chaque clé de données et la chiffrer sous sa clé d'encapsulation. Pour chiffrer des copies de la clé de données sous des clés KMS supplémentaires, ils utilisent leAWS KMS Encrypt. Pour déchiffrer les clés de données, ils utilisent leAWS KMS Decrypt. Pour plus d'informations, consultez .AWS KMSPorte-clésdans leAWS Encryption SDKSpécification en GitHub.

D'autres porte-clés génèrent la clé de données, chiffrent et déchiffrent à l'aide des meilleures pratiques pour chaque langage de programmation. Pour plus de détails, reportez-vous aux spécifications du porte-clés ou du fournisseur de clés principales dans leSection FrameworkduAWS Encryption SDKSpécification en GitHub.

Comment puis-je effectuer le suivi des clés de données qui ont été utilisées pour chiffrer mes données ?

Le kit AWS Encryption SDK le fait automatiquement pour vous. Lorsque vous chiffrez des données, le kit SDK chiffre la clé de données et stocke la clé chiffrée ainsi que les données chiffrées dans le message chiffré qu'il renvoie. Lorsque vous déchiffrez des données, le kit AWS Encryption SDK extrait la clé de données chiffrée du message chiffré, la déchiffre, puis l'utilise pour déchiffrer les données.

Comment le kit AWS Encryption SDK stocke-t-il les clés de données chiffrées avec leurs données chiffrées ?

Les opérations de chiffrement du kit AWS Encryption SDK renvoient un message chiffré, une structure de données unique qui contient les données chiffrées et leurs clés de données chiffrées. Le format de message se compose d'au moins deux parties : un en-tête et un corps. L'en-tête du message contient les clés de données chiffrées, ainsi que des informations sur la manière dont le corps du message est formé. Le corps du message contient les données chiffrées. Si la suite d'algorithmes inclut unsignature numérique, le format du message inclut unebas de pagequi contient la signature. Pour plus d'informations, consultez AWS Encryption SDK référence de format de message.

Quelle est la surcharge ajoutée par le format de message du kit AWS Encryption SDK à mes données chiffrées ?

La surcharge ajoutée par le kit AWS Encryption SDK dépend de plusieurs facteurs, notamment :

- La taille des données en texte brut
- L'algorithme pris en charge utilisé
- Le fait que les données authentifiées supplémentaires (données AAD) sont fournies ou non, et la longueur de ces données AAD
- Le nombre et le type de clés d'encapsulage ou de clés principales
- La taille du cadre (lorsque des données encadrées sont utilisées)

Lorsque vous utilisez le kitAWS Encryption SDKavec sa configuration par défaut (uneAWS KMS keyEn tant que clé d'encapsulage (ou clé principale), pas d'AAD, de données non encadrées et d'un algorithme de chiffrement avec signature), la surcharge est d'environ 600 octets. En général, vous pouvez raisonnablement considérer que le kit AWS Encryption SDK ajoute une surcharge de 1 Ko au maximum, sans compter les données AAD fournies. Pour plus d'informations, consultez AWS Encryption SDK référence de format de message.

Puis-je utiliser mon propre fournisseur de clés principales ?

Oui. Les détails d'implémentation varient en fonction du langage de programmation pris en charge que vous utilisez. Cependant, toutes les langues prises en charge vous permettent de définir des langues personnaliséesgestionnaires de matériaux cryptographiques (GMT), fournisseurs de clés principales, porte-clés, clés principales et clés d'encapsulation.

Puis-je chiffrer des données sous plusieurs clés d'encapsulage ?

Oui. Vous pouvez chiffrer la clé de données avec des clés d'encapsulage supplémentaires (ou des clés principales) afin d'accroître la redondance lorsque la clé se trouve dans une autre région ou n'est pas disponible pour le déchiffrement.

Pour chiffrer des données sous plusieurs clés d'encapsulage, créez un porte-clés ou un fournisseur de clés principales avec plusieurs clés d'encapsulage. Lorsque vous travaillez avec des porte-clés, vous pouvez créer un seul porte-clés avec plusieurs clés d'encapsulage ou un porte-clés multiple.

Lorsque vous chiffrez des données avec plusieurs clés d'encapsulation, leAWS Encryption SDKutilise une clé d'encapsulation pour générer une clé de données en texte clair. La clé de données est unique et mathématiquement sans rapport avec la clé d'encapsulation. Cette opération renvoie la clé de données en texte brut et une copie de la clé de données chiffrée par la clé d'encapsulage. Ensuite, la méthode de chiffrement chiffre la clé de données avec les autres clés d'encapsulage. Le résultatMessage chiffréinclut les données chiffrées et une clé de données chiffrée pour chaque clé d'encapsulage.

Le message chiffré peut être déchiffré à l'aide de l'une des clés d'encapsulage utilisées dans l'opération de chiffrement. LeAWS Encryption SDKutilise une clé d'encapsulage pour déchiffrer une clé de données chiffrée. Ensuite, il utilise la clé de données en texte brut pour déchiffrer les données.

Quels types de données puis-je chiffrer avec le kit AWS Encryption SDK ?

La plupart des implémentations de langages de programmation du kitAWS Encryption SDKpeut chiffrer des octets (tableaux d'octets) bruts, des flux d'E/S (flux d'octets) et des chaînes. LeAWS Encryption SDKpour .NET ne prend pas en charge les flux d'E/S. Nous fournissons des exemples de code pour chacun des langages de programmation pris en charge.

Comment le kit AWS Encryption SDK chiffre-t-il et déchiffre-t-il les flux de données d'entrée/sortie (I/O) ?

Le kit AWS Encryption SDK crée un flux de chiffrement ou de déchiffrement qui encapsule un flux d'I/O sous-jacent. Le flux de chiffrement ou de déchiffrement effectue une opération de chiffrement sur un appel en lecture ou en écriture. Par exemple, il peut lire les données en texte brut sur le flux sous-jacent et les chiffrer avant de renvoyer le résultat. Ou il peut lire un texte chiffré à partir d'un flux sous-jacent et le déchiffrer avant de renvoyer le résultat. Nous fournissons des exemples de code pour le chiffrement et le déchiffrement de flux pour chacun deslangage de programmation pris en chargequi prend en charge le streaming.

LeAWS Encryption SDKpour .NET ne prend pas en charge les flux d'E/S.

AWS Encryption SDK référence

Les informations de cette page constituent une référence pour le développement de votre propre bibliothèque de chiffrement compatible avec le kit AWS Encryption SDK. Si vous ne créez pas votre propre bibliothèque de chiffrement compatible, vous n'aurez probablement pas besoin de ces informations.

Pour utiliser le AWS Encryption SDK dans l'un des langages de programmation pris en charge, voirLangages de programmation.

Pour la spécification qui définit les éléments d'une AWS Encryption SDK implémentation approprié e, voir la AWS Encryption SDK spécification dans GitHub.

Il AWS Encryption SDK utilise les algorithmes pris en charge pour renvoyer une structure de données ou un message unique contenant des données cryptées et les clés de données cryptées correspondantes. Les rubriques suivantes expliquent les algorithmes et la structure de données. Utilisez ces informations pour générer des bibliothèques qui peuvent lire et écrire des textes chiffrés compatibles avec ce kit SDK.

Rubriques

- AWS Encryption SDK référence de format de message
- AWS Encryption SDK exemples de format de message
- Référence Données authentifiées supplémentaires (données AAD) du corps pour le kit AWS Encryption SDK
- AWS Encryption SDK référence aux algorithmes
- AWS Encryption SDK référence du vecteur d'initialisation
- AWS KMS Détails techniques du porte-clés hiérarchique

AWS Encryption SDK référence de format de message

Les informations de cette page constituent une référence pour le développement de votre propre bibliothèque de chiffrement compatible avec le kit AWS Encryption SDK. Si vous ne créez pas

votre propre bibliothèque de chiffrement compatible, vous n'aurez probablement pas besoin de ces informations.

Pour utiliser le AWS Encryption SDK dans l'un des langages de programmation pris en charge, voirLangages de programmation.

Pour la spécification qui définit les éléments d'une AWS Encryption SDK implémentation approprié e, voir la AWS Encryption SDK spécification dans GitHub.

Les opérations de chiffrement AWS Encryption SDK renvoient une structure de données unique ou un message chiffré contenant les données chiffrées (texte chiffré) et toutes les clés de données chiffrées. Pour comprendre cette structure de données, ou pour créer des bibliothèques capables de la lire et de l'écrire, vous devez comprendre le format de message.

Le format de message se compose d'au moins deux parties : un en-tête et un corps. Dans certains cas, le format de message se compose d'une troisième partie, un pied de page. Le format de message définit une séquence ordonnée d'octets dans l’ordre des octets réseau, également appelé format de poids fort. Le format de message commence par l'en-tête, est suivi du corps, puis du pied de page (le cas échéant).

Les suites d'algorithmes prises en charge AWS Encryption SDK utilisent l'une des deux versions de format de message. Les suites d'algorithmes sans engagement clé utilisent le format de message version 1. Les suites d'algorithmes avec engagement clé utilisent le format de message version 2.

Rubriques

- Structure de l'en-tête
- Structure du corps
- Structure du pied de page

Structure de l'en-tête

L'en-tête du message contient la clé de données chiffrée, ainsi que des informations sur la manière dont le corps du message est formé. Le tableau suivant décrit les champs qui constituent l'en-tête dans les versions 1 et 2 du format de message. Les octets sont ajoutés dans l'ordre indiqué.

La valeur Non présent indique que le champ n'existe pas dans cette version du format de message. Le texte en gras indique des valeurs différentes dans chaque version.

> **Note**
>
> Il peut être nécessaire de faire défiler horizontalement ou verticalement pour afficher toutes les données de ce tableau.

Structure de l'en-tête

Champ	Format de message version 1 Longueur (octets)	Format de message version 2 Longueur (octets)
Version	1	1
Type	1	Absent
ID de l'algorithme	2	2
ID de message	16	32
Longueur AAD	2 Lorsque le contexte de chiffrement est vide, la valeur du champ AAD Length de 2 octets est 0.	2 Lorsque le contexte de chiffrement est vide, la valeur du champ AAD Length de 2 octets est 0.
DONNÉES AAD	Variable. La longueur de ce champ apparaît dans les 2 octets précédents (champ AAD Length). Lorsque le contexte de chiffrement est vide, il n'y a pas de champ AAD dans l'en-tête.	Variable. La longueur de ce champ apparaît dans les 2 octets précédents (champ AAD Length). Lorsque le contexte de chiffrement est vide, il n'y a pas de champ AAD dans l'en-tête.
Nombre de clés de données chiffrées	2	2

Champ	Format de message version 1 Longueur (octets)	Format de message version 2 Longueur (octets)
Clé(s) de données chiffrée(s)	Variable. Déterminé par le nombre de clés de données chiffrées et la longueur de chacune.	Variable. Déterminé par le nombre de clés de données chiffrées et la longueur de chacune.
Type de contenu	1	1
Instances réservées	4	Absent
Longueur du vecteur d'initialisation	1	Absent
Longueur du cadre	4	4
Données de la suite d'algorithmes	Absent	Variable. Déterminé par l'algorithme ayant généré le message.
Authentification de l'en-tête	Variable. Déterminé par l'algorithme ayant généré le message.	Variable. Déterminé par l'algorithme ayant généré le message.

Version

Version de ce format de message. La version est codée en 1 ou 2 sous forme d'octet `01` ou `02` en notation hexadécimale

Type

Type de ce format de message. Le type indique le type de structure. Le seul type pris en charge est décrit comme données chiffrées authentifiées client. Sa valeur type est 128, codée en tant qu'octet `80`, au format hexadécimal.

Ce champ n'est pas présent dans le format de message version 2.

ID de l'algorithme

Identifiant de l'algorithme utilisé. Il s'agit d'une valeur de 2 octets interprétée comme un entier 16 bits non signé. Pour plus d'informations sur ces algorithmes, consultez AWS Encryption SDK référence aux algorithmes.

ID de message

Une valeur générée de manière aléatoire qui identifie le message. L'ID de message :

- identifie de façon unique le message chiffré ;
- lie faiblement l'en-tête du message au corps du message ;
- fournit un mécanisme pour réutiliser une clé de données en toute sécurité avec plusieurs messages chiffrés ;
- protège contre la réutilisation accidentelle d'une clé de données ou l'épuisement des clés dans le kit AWS Encryption SDK.

Cette valeur est de 128 bits dans le format de message version 1 et de 256 bits dans la version 2.

Longueur AAD

Longueur des données authentifiées supplémentaires (données AAD). 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 les données AAD.

Lorsque le contexte de chiffrement est vide, la valeur du champ AAD Length est 0.

DONNÉES AAD

Données authentifiées supplémentaires. Les données AAD constituent un encodage du contexte de chiffrement, un tableau de paires clé-valeur dans lequel chaque clé et valeur est une chaîne de caractères codés UTF-8. Le contexte de chiffrement est converti en une séquence d'octets et utilisé pour la valeur AAD. Lorsque le contexte de chiffrement est vide, il n'y a pas de champ AAD dans l'en-tête.

Lorsque les algorithmes de signature sont utilisés, le contexte de chiffrement doit contenir la paire clé-valeur `{'aws-crypto-public-key', Qtxt}`. Qtxt représente le point Q de la courbe elliptique compressé conformément à SEC 1 version 2.0, puis codé en base64. Le contexte de chiffrement peut contenir des valeurs supplémentaires, mais la longueur maximale des données AAD construites est 2^16 - 1 octets.

Le tableau suivant décrit les champs qui composent les données AAD. Les paires clé-valeur sont triées, par clé, en ordre croissant d'après le code caractère UTF-8. Les octets sont ajoutés dans l'ordre indiqué.

Structure AAD

Champ	Longueur (octets)
Nombre de paires clé-valeur	2
Longueur de clé	2
Clé	Variable. Est égal à la valeur spécifiée dans les 2 octets précédents (Longueur de clé).
Longueur de valeur	2
Valeur	Variable. Est égal à la valeur spécifiée dans les 2 octets précédents (Longueur de valeur).

Nombre de paires clé-valeur

Nombre de paires clé-valeur dans les données AAD. Il s'agit d'une valeur de 2 octets interprétée comme un entier 16 bits non signé qui spécifie le nombre de paires clé-valeur dans les données AAD. Le nombre maximum de paires clé-valeur dans les données AAD est 2^16 - 1.

Lorsqu'il n'y a pas de contexte de chiffrement ou si le contexte de chiffrement est vide, ce champ n'est pas présent dans la structure AAD.

Longueur de clé

Longueur de la clé pour la paire clé-valeur. 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.

Clé

Clé de la paire clé-valeur. Il s'agit d'une séquence d'octets codés en UTF-8.

Longueur de valeur

Longueur de la valeur pour la paire clé-valeur. 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 valeur.

Valeur

Valeur de la paire clé-valeur. Il s'agit d'une séquence d'octets codés en UTF-8.

Nombre de clés de données chiffrées

Nombre de clés de données chiffrées. Il s'agit d'une valeur de 2 octets interprétée comme un entier 16 bits non signé qui spécifie le nombre de clés de données chiffrées. Le nombre maximum de clés de données chiffrées dans chaque message est de 65 535 (2^16 - 1).

Clé(s) de données chiffrée(s)

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 (octets)
Longueur de l'ID du fournisseur des clés	2
ID du fournisseur de clés	Variable. Est égal à la valeur spécifiée dans les 2 octets précédents (Longueur de l'ID du fournisseur de clés).
Longueur de l'information du fournisseur de clés	2
Information du fournisseur de clés	Variable. Est égal à la valeur spécifiée dans les 2 octets précédents (Longueur de l'information du fournisseur de clés).
Longueur de la clé de données chiffrée	2
Clé de données chiffrée	Variable. Est égal à la valeur spécifiée dans les 2 octets précédents (Longueur de la clé de données chiffrée).

Longueur de l'ID du fournisseur des clés

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.

ID du fournisseur de clés

Identifiant du fournisseur de clés. Il est utilisé pour indiquer le fournisseur de la clé de données chiffrée et doit être extensible.

Longueur de l'information du fournisseur de 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.

Information du fournisseur de clés

Information sur le fournisseur de clés. Il est déterminé par le fournisseur de clés.

Lorsqu'il AWS KMS s'agit du fournisseur de la clé principale ou que vous utilisez un AWS KMS trousseau de clés, cette valeur contient le Amazon Resource Name (ARN) du AWS KMS key.

Longueur de la clé de données chiffrée

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 chiffrée

Clé de données chiffrée. Il s'agit de la clé de chiffrement des données chiffrée par le fournisseur de clés.

Type de contenu

Type de données chiffrées, non encadrées ou encadrées.

> **Note**
>
> Dans la mesure du possible, utilisez des données encadrées. Il AWS Encryption SDK prend en charge les données non encadrées uniquement pour une utilisation traditionnelle. Certaines implémentations linguistiques du AWS Encryption SDK peuvent

toujours générer du texte chiffré non encadré. Toutes les implémentations linguistiques prises en charge peuvent déchiffrer le texte chiffré encadré et non cadré.

Les données encadrées sont divisées en parties de même longueur ; chaque partie est cryptée séparément. Le type du contenu encadré est 2, codé en tant qu'octet `02` au format hexadécimal.

Les données non encadrées ne sont pas divisées ; il s'agit d'un seul blob chiffré. Le type du contenu non encadré est 1, codé en tant qu'octet `01` au format hexadécimal.

Instances réservées

Séquence réservée de 4 octets. La valeur doit être 0. Elle est codée en tant qu'octets `00 00 00 00` en notation hexadécimale (c'est-à-dire, une séquence de 4 octets d'une valeur d'entier 32 bits égale à 0).

Ce champ n'est pas présent dans le format de message version 2.

Longueur du vecteur d'initialisation

Longueur du vecteur d'initialisation. Il s'agit d'une valeur de 1 octet interprétée comme un entier 8 bits non signé qui spécifie le nombre d'octets qui contiennent le vecteur d'initialisation. Cette valeur est déterminée par la valeur d'octets du vecteur d'initialisation de l'algorithme qui a généré le message.

Ce champ n'est pas présent dans la version 2 du format de message, qui prend uniquement en charge les suites d'algorithmes utilisant des valeurs IV déterministes dans l'en-tête du message.

Longueur du cadre

La longueur de chaque trame de données encadrées. Il s'agit d'une valeur de 4 octets interprétée comme un entier non signé de 32 bits qui indique le nombre d'octets dans chaque trame. Lorsque les données ne sont pas encadrées, c'est-à-dire lorsque la valeur du `Content Type` champ est 1, cette valeur doit être 0.

> **Note**
>
> Dans la mesure du possible, utilisez des données encadrées. Il AWS Encryption SDK prend en charge les données non encadrées uniquement pour une utilisation traditionnelle. Certaines implémentations linguistiques du AWS Encryption SDK peuvent toujours générer du texte chiffré non encadré. Toutes les implémentations linguistiques prises en charge peuvent déchiffrer le texte chiffré encadré et non cadré.

Données de la suite d'algorithmes

Données supplémentaires nécessaires à l'algorithme qui a généré le message. La longueur et le contenu sont déterminés par l'algorithme. Sa longueur peut être égale à 0.

Ce champ n'est pas présent dans le format de message version 1.

Authentification de l'en-tête

L'authentification de l'en-tête est déterminée par l'algorithme qui a généré le message. L'authentification de l'en-tête est calculée sur l'ensemble de l'en-tête. Elle se compose d'un vecteur d'initialisation et d'une balise d'authentification. Les octets sont ajoutés dans l'ordre indiqué.

Structure de l'authentification de l'en-tête

Champ	Longueur dans la version 1.0 (octets)	Longueur dans la version 2.0 (octets)
IV	Variable. Déterminée par la valeur d'octets du vecteur d'initialisation de l'algorithme qui a généré le message.	N/A
Balise d'authentification	Variable. Déterminée par la valeur d'octets de la balise d'authentification de l'algorithme qui a généré le message.	Variable. Déterminée par la valeur d'octets de la balise d'authentification de l'algorithme qui a généré le message.

IV

Vecteur d'initialisation utilisé pour calculer la balise d'authentification de l'en-tête.

Ce champ n'est pas présent dans l'en-tête du format de message version 2. La version 2 du format de message ne prend en charge que les suites d'algorithmes qui utilisent des valeurs IV déterministes dans l'en-tête du message.

Balise d'authentification

Valeur d'authentification pour l'en-tête. Elle est utilisée pour authentifier l'ensemble du contenu de l'en-tête.

Structure du corps

Le corps du message contient les données chiffrées, appelées texte chiffré. La structure du corps dépend du type de contenu (non encadré ou encadré). Les sections suivantes décrivent le format du corps du message pour chaque type de contenu. La structure du corps du message est la même dans les versions 1 et 2 du format de message.

Rubriques

Données non encadrées

Les données non encadrées sont chiffrées dans un objet blob unique avec un vecteur d'initialisation et des données de corps uniques.

> **Note**
>
> Dans la mesure du possible, utilisez des données encadrées. Il AWS Encryption SDK prend en charge les données non encadrées uniquement pour une utilisation traditionnelle. Certaines implémentations linguistiques du AWS Encryption SDK peuvent toujours générer du texte chiffré non encadré. Toutes les implémentations linguistiques prises en charge peuvent déchiffrer le texte chiffré encadré et non cadré.

Le tableau suivant décrit les champs qui composent les données non encadrées. Les octets sont ajoutés dans l'ordre indiqué.

Structure du corps non encadré

Champ	Longueur, en octets
IV	Variable. Est égal à la valeur spécifiée dans l'octet Longueur du vecteur d'initialisation de l'en-tête.
Longueur du contenu chiffré	8

Champ	Longueur, en octets
Contenu chiffré	Variable. Est égal à la valeur spécifiée dans les 8 octets précédents (Longueur du contenu chiffré).
Balise d'authentification	Variable. Déterminé par l'implémentation d'algorithme utilisée.

IV

Vecteur d'initialisation à utiliser avec l'algorithme de chiffrement.

Longueur du contenu chiffré

Longueur du contenu chiffré, ou texte chiffré. Il s'agit d'une valeur de 8 octets interprétée comme un entier 64 bits non signé qui spécifie le nombre d'octets qui contiennent le contenu chiffré.

Techniquement, la valeur maximale autorisée est de 2^63 - 1, ou 8 exbioctets (8 Eio). Toutefois, dans la pratique, la valeur maximale est de 2^36 - 32, ou 64 gibioctets (64 Gio), en raison des restrictions imposées par les algorithmes implémentés.

> **Note**
>
> L'implémentation Java de ce kit SDK limite encore cette valeur à 2^31 - 1, ou 2 gibioctets (2 Gio), en raison des restrictions du langage.

Contenu chiffré

Contenu chiffré (texte chiffré), tel que renvoyé par l'algorithme de chiffrement.

Balise d'authentification

Valeur d'authentification pour le corps. Elle est utilisée pour authentifier le corps du message.

Données encadrées

Dans les données encadrées, les données en texte brut sont divisées en parties de longueur égale appelées trames. Il AWS Encryption SDK chiffre chaque image séparément avec un IV et un corps AAD uniques.

> **Note**
>
> Dans la mesure du possible, utilisez des données encadrées. Il AWS Encryption SDK prend en charge les données non encadrées uniquement pour une utilisation traditionnelle. Certaines implémentations linguistiques du AWS Encryption SDK peuvent toujours générer du texte chiffré non encadré. Toutes les implémentations linguistiques prises en charge peuvent déchiffrer le texte chiffré encadré et non cadré.

La longueur du cadre, qui est la longueur du contenu chiffré dans le cadre, peut être différente pour chaque message. Le nombre maximal d'octets dans une trame est de 2^32 - 1. Le nombre maximal d'images dans un message est de 2^32 - 1.

Il existe deux types de cadres : régulier et final. Chaque message doit comporter ou inclure une image finale.

Toutes les images régulières d'un message ont la même longueur d'image. Le cadre final peut avoir une longueur de cadre différente.

La composition des cadres dans les données encadrées varie en fonction de la longueur du contenu chiffré.

- Égal à la longueur de trame — Lorsque la longueur du contenu chiffré est identique à celle des trames normales, le message peut consister en une trame normale contenant les données, suivie d'une trame finale de longueur nulle (0). Ou bien, le message peut être composé d'un seul cadre final contenant les données. Dans ce cas, le cadre final a la même longueur de cadre que les cadres normaux.
- Multiple de la longueur de trame — Lorsque la longueur du contenu chiffré est un multiple exact de la longueur d'image des trames normales, le message peut se terminer par une trame normale contenant les données, suivie d'une trame finale de longueur nulle (0). Ou bien, le message peut se terminer dans un cadre final contenant les données. Dans ce cas, le cadre final a la même longueur de cadre que les cadres normaux.
- Pas un multiple de la longueur d'image — Lorsque la longueur du contenu chiffré n'est pas un multiple exact de la longueur d'image des images normales, la dernière image contient les données restantes. La longueur du cadre final est inférieure à la longueur de cadre des cadres normaux.

- Moins que la longueur de trame — Lorsque la longueur du contenu chiffré est inférieure à la longueur de trame des trames normales, le message consiste en une trame finale contenant toutes les données. La longueur du cadre final est inférieure à la longueur de cadre des cadres normaux.

Les tableaux suivants décrivent les champs qui composent les cadres. Les octets sont ajoutés dans l'ordre indiqué.

Structure de corps encadré, cadre régulier

Champ	Longueur, en octets
Numéro de séquence	4
IV	Variable. Est égal à la valeur spécifiée dans l'octet Longueur du vecteur d'initialisation de l'en-tête.
Contenu chiffré	Variable. Est égal à la valeur spécifiée dans le Longueur du cadre de l'en-tête.
Balise d'authentification	Variable. Déterminé par l'algorithme utilisé, comme spécifié dans le ID de l'algorithme de l'en-tête.

Numéro de séquence

Numéro de séquence du cadre. Il s'agit d'un numéro du compteur incrémentiel pour le cadre. Il s'agit d'une valeur de 4 octets interprétée comme un entier 32 bits non signé.

Les données encadrées doivent démarrer au numéro de séquence 1. Les cadres suivants doivent être dans l'ordre et doivent être incrémentés de 1 par rapport au cadre précédent. Dans le cas contraire, le processus de déchiffrement s'arrête et indique une erreur.

IV

Vecteur d'initialisation du cadre. Le kit SDK utilise une méthode déterministe pour construire un vecteur d'initialisation différent pour chaque image du message. Sa longueur est spécifiée par la suite d'algorithmes utilisée.

Contenu chiffré

Contenu chiffré (texte chiffré) du cadre, tel que renvoyé par l'algorithme de chiffrement.

Balise d'authentification

Valeur d'authentification pour le cadre. Elle est utilisée pour authentifier l'ensemble du cadre.

Structure de corps encadré, cadre final

Champ	Longueur, en octets
Fin du numéro de séquence	4
Numéro de séquence	4
IV	Variable. Est égal à la valeur spécifiée dans l'octet Longueur du vecteur d'initialisation de l'en-tête.
Longueur du contenu chiffré	4
Contenu chiffré	Variable. Est égal à la valeur spécifiée dans les 4 octets précédents (Longueur du contenu chiffré).
Balise d'authentification	Variable. Déterminé par l'algorithme utilisé, comme spécifié dans le ID de l'algorithme de l'en-tête.

Fin du numéro de séquence

Indicateur pour le dernier cadre. La valeur est codée en tant que 4 octets `FF FF FF FF` au format hexadécimal.

Numéro de séquence

Numéro de séquence du cadre. Il s'agit d'un numéro du compteur incrémentiel pour le cadre. Il s'agit d'une valeur de 4 octets interprétée comme un entier 32 bits non signé.

Les données encadrées doivent démarrer au numéro de séquence 1. Les cadres suivants doivent être dans l'ordre et doivent être incrémentés de 1 par rapport au cadre précédent. Dans le cas contraire, le processus de déchiffrement s'arrête et indique une erreur.

IV

Vecteur d'initialisation du cadre. Le kit SDK utilise une méthode déterministe pour construire un vecteur d'initialisation différent pour chaque image du message. La longueur du vecteur d'initialisation est spécifiée par la suite d'algorithmes.

Longueur du contenu chiffré

Longueur du contenu chiffré. Il s'agit d'une valeur de 4 octets interprétée comme un entier 32 bits non signé qui spécifie le nombre d'octets qui contiennent le contenu chiffré pour le cadre.

Contenu chiffré

Contenu chiffré (texte chiffré) du cadre, tel que renvoyé par l'algorithme de chiffrement.

Balise d'authentification

Valeur d'authentification pour le cadre. Elle est utilisée pour authentifier l'ensemble du cadre.

Structure du pied de page

Lorsque les algorithmes de signature sont utilisés, le format de message contient un pied de page. Le pied de page du message contient une signature numérique calculée sur l'en-tête et le corps du message. Le tableau suivant décrit les champs qui composent le pied de page. Les octets sont ajoutés dans l'ordre indiqué. La structure du pied de page du message est identique dans les versions 1 et 2 du format de message.

Structure du pied de page

Champ	Longueur, en octets
Longueur de signature	2
Signature	Variable. Est égal à la valeur spécifiée dans les 2 octets précédents (Longueur de signature).

Longueur de signature

Longueur de la signature. 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 signature.

Signature

Signature.

AWS Encryption SDK exemples de format de message

Les informations de cette page constituent une référence pour le développement de votre propre bibliothèque de chiffrement compatible avec le kit AWS Encryption SDK. Si vous ne créez pas votre propre bibliothèque de chiffrement compatible, vous n'aurez probablement pas besoin de ces informations.

Pour utiliser le AWS Encryption SDK dans l'un des langages de programmation pris en charge, voirLangages de programmation.

Pour la spécification qui définit les éléments d'une AWS Encryption SDK implémentation approprié e, voir la AWS Encryption SDK spécification dans GitHub.

Les rubriques suivantes présentent des exemples de format de AWS Encryption SDK message. Chaque exemple illustre les octets bruts, au format hexadécimal, suivis d'une description de ce que ces octets représentent.

Rubriques

- Données encadrées (format de message version 1)
- Données encadrées (format de message version 2)
- Données non encadrées (format de message version 1)

Données encadrées (format de message version 1)

L'exemple suivant montre le format de message pour les données encadrées dans le format de message version 1.

```
+--------+
```

```
| Header |
+--------+
01                                      Version (1.0)
80                                      Type (128, customer authenticated encrypted
 data)
0378                                    Algorithm ID (see Référence relative aux
 algorithmes du kit)
6E7C0FBD 4DF4A999 717C22A2 DDFE1A27     Message ID (random 128-bit value)
008E                                    AAD Length (142)
0004                                    AAD Key-Value Pair Count (4)
0005                                    AAD Key-Value Pair 1, Key Length (5)
30746869 73                             AAD Key-Value Pair 1, Key ("0This")
0002                                    AAD Key-Value Pair 1, Value Length (2)
6973                                    AAD Key-Value Pair 1, Value ("is")
0003                                    AAD Key-Value Pair 2, Key Length (3)
31616E                                  AAD Key-Value Pair 2, Key ("1an")
000A                                    AAD Key-Value Pair 2, Value Length (10)
656E6372 79774690 6F6E                  AAD Key-Value Pair 2, Value ("encryption")
0008                                    AAD Key-Value Pair 3, Key Length (8)
32636F6E 74657874                       AAD Key-Value Pair 3, Key ("2context")
0007                                    AAD Key-Value Pair 3, Value Length (7)
6578616D 706C65                         AAD Key-Value Pair 3, Value ("example")
0015                                    AAD Key-Value Pair 4, Key Length (21)
6177732D 63727970 746F2D70 75626C69     AAD Key-Value Pair 4, Key ("aws-crypto-
public-key")
632D6B65 79
0044                                    AAD Key-Value Pair 4, Value Length (68)
416A4173 7569326F 7430364C 4B77715A     AAD Key-Value Pair 4, Value
 ("AjAsui2ot06LKwqZXDJnU/Aqc2vD+0OkpOZ1cc8Tg2qd7rs5aLTg7lvfUEW/86+/5w==")
58444A6E 552F4171 63327644 2B304F6B
704F5A31 63633854 67327164 37727335
614C5467 376C7666 5545572F 38362B2F
35773D3D
0002                                    EncryptedDataKeyCount (2)
0007                                    Encrypted Data Key 1, Key Provider ID Length
 (7)
6177732D 6B6D73                         Encrypted Data Key 1, Key Provider ID ("aws-
kms")
004B                                    Encrypted Data Key 1, Key Provider
 Information Length (75)
61726E3A 6177733A 6B6D733A 75732D77     Encrypted Data Key 1, Key Provider
 Information ("arn:aws:kms:us-west-2:111122223333:key/715c0818-5825-4245-
a755-138a6d9a11e6")
6573742D 323A3131 31313232 32323333
```

```
33333A6B 65792F37 31356330 3831382D
35383235 2D343234 352D6137 35352D31
33386136 64396131 316536
00A7                                    Encrypted Data Key 1, Encrypted Data Key
 Length (167)
01010200 7857A1C1 F7370545 4ECA7C83     Encrypted Data Key 1, Encrypted Data Key
956C4702 23DCE8D7 16C59679 973E3CED
02A4EF29 7F000000 7E307C06 092A8648
86F70D01 0706A06F 306D0201 00306806
092A8648 86F70D01 0701301E 06096086
48016503 04012E30 11040C3F F02C897B
7A12EB19 8BF2D802 0110803B 24003D1F
A5474FBC 392360B5 CB9997E0 6A17DE4C
A6BD7332 6BF86DAB 60D8CCB8 8295DBE9
4707E356 ADA3735A 7C52D778 B3135A47
9F224BF9 E67E87
0007                                    Encrypted Data Key 2, Key Provider ID Length
 (7)
6177732D 6B6D73                         Encrypted Data Key 2, Key Provider ID ("aws-
kms")
004E                                    Encrypted Data Key 2, Key Provider
 Information Length (78)
61726E3A 6177733A 6B6D733A 63612D63     Encrypted Data Key 2, Key Provider
 Information ("arn:aws:kms:ca-central-1:111122223333:key/9b13ca4b-afcc-46a8-aa47-
be3435b423ff")
656E7472 616C2D31 3A313131 31323232
32333333 333A6B65 792F3962 31336361
34622D61 6663632D 34366138 2D616134
372D6265 33343335 62343233 6666
00A7                                    Encrypted Data Key 2, Encrypted Data Key
 Length (167)
01010200 78FAFFFB D6DE06AF AC72F79B     Encrypted Data Key 2, Encrypted Data Key
0E57BD87 3F60F4E6 FD196144 5A002C94
AF787150 69000000 7E307C06 092A8648
86F70D01 0706A06F 306D0201 00306806
092A8648 86F70D01 0701301E 06096086
48016503 04012E30 11040C36 CD985E12
D218B674 5BBC6102 0110803B 0320E3CD
E470AA27 DEAB660B 3E0CE8E0 8B1A89E4
57DCC69B AAB1294F 21202C01 9A50D323
72EBAAFD E24E3ED8 7168E0FA DB40508F
556FBD58 9E621C
02                                      Content Type (2, framed data)
00000000                                Reserved
```

```
0C                                      IV Length (12)
00000100                                Frame Length (256)
4ECBD5C0 9899CA65 923D2347              IV
0B896144 0CA27950 CA571201 4DA58029     Authentication Tag
+------+
| Body |
+------+
00000001                                Frame 1, Sequence Number (1)
6BD3FE9C ADBCB213 5B89E8F1              Frame 1, IV
1F6471E0 A51AF310 10FA9EF6 F0C76EDF     Frame 1, Encrypted Content
F5AFA33C 7D2E8C6C 9C5D5175 A212AF8E
FBD9A0C3 C6E3FB59 C125DBF2 89AC7939
BDEE43A8 0F00F49E ACBBD8B2 1C785089
A90DB923 699A1495 C3B31B50 0A48A830
201E3AD9 1EA6DA14 7F6496DB 6BC104A4
DEB7F372 375ECB28 9BF84B6D 2863889F
CB80A167 9C361C4B 5EC07438 7A4822B4
A7D9D2CC 5150D414 AF75F509 FCE118BD
6D1E798B AEBA4CDB AD009E5F 1A571B77
0041BC78 3E5F2F41 8AF157FD 461E959A
BB732F27 D83DC36D CC9EBC05 00D87803
57F2BB80 066971C2 DEEA062F 4F36255D
E866C042 E1382369 12E9926B BA40E2FC
A820055F FB47E428 41876F14 3B6261D9
5262DB34 59F5D37E 76E46522 E8213640
04EE3CC5 379732B5 F56751FA 8E5F26AD     Frame 1, Authentication Tag
00000002                                Frame 2, Sequence Number (2)
F1140984 FF25F943 959BE514              Frame 2, IV
216C7C6A 2234F395 F0D2D9B9 304670BF     Frame 2, Encrypted Content
A1042608 8A8BCB3F B58CF384 D72EC004
A41455B4 9A78BAC9 36E54E68 2709B7BD
A884C1E1 705FF696 E540D297 446A8285
23DFEE28 E74B225A 732F2C0C 27C6BDA2
7597C901 65EF3502 546575D4 6D5EBF22
1FF787AB 2E38FD77 125D129C 43D44B96
778D7CEE 3C36625F FF3A985C 76F7D320
ED70B1F3 79729B47 E7D9B5FC 02FCE9F5
C8760D55 7779520A 81D54F9B EC45219D
95941F7E 5CBAEAC8 CEC13B62 1464757D
AC65B6EF 08262D74 44670624 A3657F7F
2A57F1FD E7060503 AC37E197 2F297A84
DF1172C2 FA63CF54 E6E2B9B6 A86F582B
3B16F868 1BBC5E4D 0B6919B3 08D5ABCF
FECDC4A4 8577F08B 99D766A1 E5545670
```

```
A61F0A3B A3E45A84 4D151493 63ECA38F        Frame 2, Authentication Tag
FFFFFFFF                                  Final Frame, Sequence Number End
00000003                                  Final Frame, Sequence Number (3)
35F74F11 25410F01 DD9E04BF                Final Frame, IV
0000008E                                  Final Frame, Encrypted Content Length (142)
F7A53D37 2F467237 6FBD0B57 D1DFE830       Final Frame, Encrypted Content
B965AD1F A910AA5F 5EFFFFF4 BC7D431C
BA9FA7C4 B25AF82E 64A04E3A A0915526
88859500 7096FABB 3ACAD32A 75CFED0C
4A4E52A3 8E41484D 270B7A0F ED61810C
3A043180 DF25E5C5 3676E449 0986557F
C051AD55 A437F6BC 139E9E55 6199FD60
6ADC017D BA41CDA4 C9F17A83 3823F9EC
B66B6A5A 80FDB433 8A48D6A4 21CB
811234FD 8D589683 51F6F39A 040B3E3B       Final Frame, Authentication Tag
+--------+
| Footer |
+--------+
0066                                      Signature Length (102)
30640230 085C1D3C 63424E15 B2244448       Signature
639AED00 F7624854 F8CF2203 D7198A28
758B309F 5EFD9D5D 2E07AD0B 467B8317
5208B133 02301DF7 2DFC877A 66838028
3C6A7D5E 4F8B894E 83D98E7C E350F424
7E06808D 0FE79002 E24422B9 98A0D130
A13762FF 844D
```

Données encadrées (format de message version 2)

L'exemple suivant montre le format de message pour les données encadrées dans le format de message version 2.

```
+--------+
| Header |
+--------+
02                                        Version (2.0)
0578                                      Algorithm ID (see Algorithms reference)
122747eb 21dfe39b 38631c61 7fad7340
cc621a30 32a11cc3 216d0204 fd148459       Message ID (random 256-bit value)
008e                                      AAD Length (142)
0004                                      AAD Key-Value Pair Count (4)
0005                                      AAD Key-Value Pair 1, Key Length (5)
30546869 73                               AAD Key-Value Pair 1, Key ("0This")
```

```
0002                                    AAD Key-Value Pair 1, Value Length (2)
6973                                    AAD Key-Value Pair 1, Value ("is")
0003                                    AAD Key-Value Pair 2, Key Length (3)
31616e                                  AAD Key-Value Pair 2, Key ("1an")
000a                                    AAD Key-Value Pair 2, Value Length (10)
656e6372 79707469 6f6e                  AAD Key-Value Pair 2, Value ("encryption")
0008                                    AAD Key-Value Pair 3, Key Length (8)
32636f6e 74657874                       AAD Key-Value Pair 3, Key ("2context")
0007                                    AAD Key-Value Pair 3, Value Length (7)
6578616d 706c65                         AAD Key-Value Pair 3, Value ("example")
0015                                    AAD Key-Value Pair 4, Key Length (21)
6177732d 63727970 746f2d70 75626c69     AAD Key-Value Pair 4, Key ("aws-crypto-
public-key")
632d6b65 79
0044                                    AAD Key-Value Pair 4, Value Length (68)
41746733 72703845 41345161 36706669     AAD Key-Value Pair 4, Value
 ("QXRnM3JwOEVBNFFhNnBmaTk3MUlTNTk3NHpOMnlZWE5vSmtwRHFPc0dIYkVaVDRqME5OMlFkRStmbTFVY01WdThnPT0=
39373149 53353937 347a4e32 7959584e
6f4a6b70 44714f73 47486245 5a54346a
304e4e32 5164452b 666d3155 634d5675
38673d3d
0001                                    Encrypted Data Key Count (1)
0007                                    Encrypted Data Key 1, Key Provider ID Length
 (7)
6177732d 6b6d73                         Encrypted Data Key 1, Key Provider ID ("aws-
kms")
004b                                    Encrypted Data Key 1, Key Provider
 Information Length (75)
61726e3a 6177733a 6b6d733a 75732d77      Encrypted Data Key 1, Key
 Provider Information ("arn:aws:kms:us-west-2:658956600833:key/b3537ef1-
d8dc-4780-9f5a-55776cbb2f7f")
6573742d 323a3635 38393536 36303038
33333a6b 65792f62 33353337 6566312d
64386463 2d343738 302d3966 35612d35
35373736 63626232 663766
00a7                                    Encrypted Data Key 1, Encrypted Data Key
 Length (167)
01010100 7840f38c 275e3109 7416c107     Encrypted Data Key 1, Encrypted Data Key
29515057 1964ada3 ef1c21e9 4c8ba0bd
bc9d0fb4 14000000 7e307c06 092a8648
86f70d01 0706a06f 306d0201 00306806
092a8648 86f70d01 0701301e 06096086
48016503 04012e30 11040c39 32d75294
06063803 f8460802 0110803b 2a46bc23
```

```
413196d2 903bf1d7 3ed98fc8 a94ac6ed
e00ee216 74ec1349 12777577 7fa052a5
ba62e9e4 f2ac8df6 bcb1758f 2ce0fb21
cc9ee5c9 7203bb
02                                      Content Type (2, framed data)
00001000                                Frame Length (4096)
05cd035b 29d5499d 4587570b 87502afe     Algorithm Suite Data (key commitment)
634f7b2c c3df2aa9 88a10105 4a2c7687
76cb339f 2536741f 59a1c202 4f2594ab     Authentication Tag
+------+
| Body |
+------+
ffffffff                                Final Frame, Sequence Number End
00000001                                Final Frame, Sequence Number (1)
00000000 00000000 00000001              Final Frame, IV
00000009                                Final Frame, Encrypted Content Length (9)
fa6e39c6 02927399 3e                    Final Frame, Encrypted Content
f683a564 405d68db eeb0656c d57c9eb0     Final Frame, Authentication Tag
+--------+
| Footer |
+--------+
0067                                    Signature Length (103)
30650230 2a1647ad 98867925 c1712e8f     Signature
ade70b3f 2a2bc3b8 50eb91ef 56cfdd18
967d91d8 42d92baf 357bba48 f636c7a0
869cade2 023100aa ae12d08f 8a0afe85
e5054803 110c9ed8 11b2e08a c4a052a9
074217ea 3b01b660 534ac921 bf091d12
3657e2b0 9368bd
```

Données non encadrées (format de message version 1)

L'exemple suivant illustre le format de message pour des données non encadrées.

> **Note**
>
> Dans la mesure du possible, utilisez des données encadrées. Il AWS Encryption SDK prend en charge les données non encadrées uniquement pour une utilisation traditionnelle. Certaines implémentations linguistiques du AWS Encryption SDK peuvent toujours générer

du texte chiffré non encadré. Toutes les implémentations linguistiques prises en charge peuvent déchiffrer le texte chiffré encadré et non cadré.

```
+--------+
| Header |
+--------+
01                                      Version (1.0)
80                                      Type (128, customer authenticated encrypted
 data)
0378                                    Algorithm ID (see Référence relative aux
 algorithmes du kit)
B8929B01 753D4A45 C0217F39 404F70FF     Message ID (random 128-bit value)
008E                                    AAD Length (142)
0004                                    AAD Key-Value Pair Count (4)
0005                                    AAD Key-Value Pair 1, Key Length (5)
30746869 73                             AAD Key-Value Pair 1, Key ("0This")
0002                                    AAD Key-Value Pair 1, Value Length (2)
6973                                    AAD Key-Value Pair 1, Value ("is")
0003                                    AAD Key-Value Pair 2, Key Length (3)
31616E                                  AAD Key-Value Pair 2, Key ("1an")
000A                                    AAD Key-Value Pair 2, Value Length (10)
656E6372 79774690 6F6E                  AAD Key-Value Pair 2, Value ("encryption")
0008                                    AAD Key-Value Pair 3, Key Length (8)
32636F6E 74657874                       AAD Key-Value Pair 3, Key ("2context")
0007                                    AAD Key-Value Pair 3, Value Length (7)
6578616D 706C65                         AAD Key-Value Pair 3, Value ("example")
0015                                    AAD Key-Value Pair 4, Key Length (21)
6177732D 63727970 746F2D70 75626C69     AAD Key-Value Pair 4, Key ("aws-crypto-
public-key")
632D6B65 79
0044                                    AAD Key-Value Pair 4, Value Length (68)
41734738 67473949 6E4C5075 3136594B     AAD Key-Value Pair 4, Value
 ("AsG8gG9InLPu16YKlqXTOD+nykG8YqHAhqecj8aXfD2e5B4gtVE73dZkyClA+rAMOQ==")
6C715854 4F442B6E 796B4738 59714841
68716563 6A386158 66443265 35423467
74564537 33645A6B 79436C41 2B72414D
4F513D3D
0002                                    Encrypted Data Key Count (2)
0007                                    Encrypted Data Key 1, Key Provider ID Length
 (7)
6177732D 6B6D73                         Encrypted Data Key 1, Key Provider ID ("aws-
kms")
```

```
004B                                    Encrypted Data Key 1, Key Provider
 Information Length (75)
61726E3A 6177733A 6B6D733A 75732D77      Encrypted Data Key 1, Key Provider
 Information ("arn:aws:kms:us-west-2:111122223333:key/715c0818-5825-4245-
a755-138a6d9a11e6")
6573742D 323A3131 31313232 32323333
33333A6B 65792F37 31356330 3831382D
35383235 2D343234 352D6137 35352D31
33386136 64396131 316536
00A7                                    Encrypted Data Key 1, Encrypted Data Key
 Length (167)
01010200 7857A1C1 F7370545 4ECA7C83      Encrypted Data Key 1, Encrypted Data Key
956C4702 23DCE8D7 16C59679 973E3CED
02A4EF29 7F000000 7E307C06 092A8648
86F70D01 0706A06F 306D0201 00306806
092A8648 86F70D01 0701301E 06096086
48016503 04012E30 11040C28 4116449A
0F2A0383 659EF802 0110803B B23A8133
3A33605C 48840656 C38BCB1F 9CCE7369
E9A33EBE 33F46461 0591FECA 947262F3
418E1151 21311A75 E575ECC5 61A286E0
3E2DEBD5 CB005D
0007                                    Encrypted Data Key 2, Key Provider ID Length
 (7)
6177732D 6B6D73                         Encrypted Data Key 2, Key Provider ID ("aws-
kms")
004E                                    Encrypted Data Key 2, Key Provider
 Information Length (78)
61726E3A 6177733A 6B6D733A 63612D63      Encrypted Data Key 2, Key Provider
 Information ("arn:aws:kms:ca-central-1:111122223333:key/9b13ca4b-afcc-46a8-aa47-
be3435b423ff")
656E7472 616C2D31 3A313131 31323232
32333333 333A6B65 792F3962 31336361
34622D61 6663632D 34366138 2D616134
372D6265 33343335 62343233 6666
00A7                                    Encrypted Data Key 2, Encrypted Data Key
 Length (167)
01010200 78FAFFFB D6DE06AF AC72F79B      Encrypted Data Key 2, Encrypted Data Key
0E57BD87 3F60F4E6 FD196144 5A002C94
AF787150 69000000 7E307C06 092A8648
86F70D01 0706A06F 306D0201 00306806
092A8648 86F70D01 0701301E 06096086
48016503 04012E30 11040CB2 A820D0CC
76616EF2 A6B30D02 0110803B 8073D0F1
```

```
FDD01BD9 B0979082 099FDBFC F7B13548
3CC686D7 F3CF7C7A CCC52639 122A1495
71F18A46 80E2C43F A34C0E58 11D05114
2A363C2A E11397
01                                      Content Type (1, nonframed data)
00000000                                Reserved
0C                                      IV Length (12)
00000000                                Frame Length (0, nonframed data)
734C1BBE 032F7025 84CDA9D0              IV
2C82BB23 4CBF4AAB 8F5C6002 622E886C     Authentication Tag
+------+
| Body |
+------+
D39DD3E5 915E0201 77A4AB11              IV
00000000 0000028E                       Encrypted Content Length (654)
E8B6F955 B5F22FE4 FD890224 4E1D5155     Encrypted Content
5871BA4C 93F78436 1085E4F8 D61ECE28
59455BD8 D76479DF C28D2E0B BDB3D5D3
E4159DFE C8A944B6 685643FC EA24122B
6766ECD5 E3F54653 DF205D30 0081D2D8
55FCDA5B 9F5318BC F4265B06 2FE7C741
C7D75BCC 10F05EA5 0E2F2F40 47A60344
ECE10AA7 559AF633 9DE2C21B 12AC8087
95FE9C58 C65329D1 377C4CD7 EA103EC1
31E4F48A 9B1CC047 EE5A0719 704211E5
B48A2068 8060DF60 B492A737 21B0DB21
C9B21A10 371E6179 78FAFB0B BAAEC3F4
9D86E334 701E1442 EA5DA288 64485077
54C0C231 AD43571A B9071925 609A4E59
B8178484 7EB73A4F AAE46B26 F5B374B8
12B0000C 8429F504 936B2492 AAF47E94
A5BA804F 7F190927 5D2DF651 B59D4C2F
A15D0551 DAEBA4AF 2060D0D5 CB1DA4E6
5E2034DB 4D19E7CD EEA6CF7E 549C86AC
46B2C979 AB84EE12 202FD6DF E7E3C09F
C2394012 AF20A97E 369BCBDA 62459D3E
C6FFB914 FEFD4DE5 88F5AFE1 98488557
1BABBAE4 BE55325E 4FB7E602 C1C04BEE
F3CB6B86 71666C06 6BF74E1B 0F881F31
B731839B CF711F6A 84CA95F5 958D3B44
E3862DF6 338E02B5 C345CFF8 A31D54F3
6920AA76 0BF8E903 552C5A04 917CCD11
D4E5DF5C 491EE86B 20C33FE1 5D21F0AD
6932E67C C64B3A26 B8988B25 CFA33E2B
```

```
63490741 3AB79D60 D8AEFBE9 2F48E25A
978A019C FE49EE0A 0E96BF0D D6074DDB
66DFF333 0E10226F 0A1B219C BE54E4C2
2C15100C 6A2AA3F1 88251874 FDC94F6B
9247EF61 3E7B7E0D 29F3AD89 FA14A29C
76E08E9B 9ADCDF8C C886D4FD A69F6CB4
E24FDE26 3044C856 BF08F051 1ADAD329
C4A46A1E B5AB72FE 096041F1 F3F3571B
2EAFD9CB B9EB8B83 AE05885A 8F2D2793
1E3305D9 0C9E2294 E8AD7E3B 8E4DEC96
6276C5F1 A3B7E51E 422D365D E4C0259C
50715406 822D1682 80B0F2E5 5C94
65B2E942 24BEEA6E A513F918 CCEC1DE3      Authentication Tag
+--------+
| Footer |
+--------+
0067                                     Signature Length (103)
30650230 7229DDF5 B86A5B64 54E4D627      Signature
CBE194F1 1CC0F8CF D27B7F8B F50658C0
BE84B355 3CED1721 A0BE2A1B 8E3F449E
1BEB8281 023100B2 0CB323EF 58A4ACE3
1559963B 889F72C3 B15D1700 5FB26E61
331F3614 BC407CEE B86A66FA CBF74D9E
34CB7E4B 363A38
```

Référence Données authentifiées supplémentaires (données AAD) du corps pour le kit AWS Encryption SDK

Les informations de cette page constituent une référence pour le développement de votre propre bibliothèque de chiffrement compatible avec le kit AWS Encryption SDK. Si vous ne créez pas votre propre bibliothèque de chiffrement compatible, vous n'aurez probablement pas besoin de ces informations.

Pour utiliser le AWS Encryption SDK dans l'un des langages de programmation pris en charge, voirLangages de programmation.

Pour la spécification qui définit les éléments d'une AWS Encryption SDK implémentation appropriée, voir la AWS Encryption SDK spécification dans GitHub.

Vous devez fournir des données authentifiées supplémentaires (données AAD) à l'algorithme AES GCM pour chaque opération de chiffrement. Cela s'applique aux données du corps encadrées et non encadrées. Pour plus d'informations sur AAD et son mode d'utilisation en Galois/Counter Mode (GCM), consultez Recommendations pour les Modes d'opération des blocs de chiffrement : Galois/Counter Mode (GCM) et GMAC.

Le tableau suivant décrit les champs formant les données AAD du corps. Les octets sont ajoutés dans l'ordre indiqué.

Structure des données AAD du corps

Champ	Longueur, en octets
ID de message	16
Contenu AAD du corps	Variable. Consultez Contenu AAD du corps dans la liste suivante.
Numéro de séquence	4
Longueur du contenu	8

ID de message

Même valeur ID de message que celle définie dans l'en-tête du message.

Contenu AAD du corps

Valeur codée en UTF-8 déterminée par le type de données du corps utilisé.

Pour les données non encadrées, utilisez la valeur `AWSKMSEncryptionClient Single Block`.

Pour les cadres réguliers dans les données encadrées, utilisez la valeur `AWSKMSEncryptionClient Frame`.

Pour le cadre final dans les données encadrées, utilisez la valeur `AWSKMSEncryptionClient Final Frame`.

Numéro de séquence

Valeur de 4 octets interprétée comme un entier 32 bits non signé.

Pour les données encadrées, il s'agit du numéro de séquence du cadre.

Pour les données non encadrées, utilisez la valeur 1, codée en tant que 4 octets `00 00 00 01` au format hexadécimal.

Longueur du contenu

Longueur, en octets, des données en texte brut fournies à l'algorithme pour le chiffrement. Il s'agit d'une valeur de 8 octets interprétée comme un entier 64 bits non signé.

AWS Encryption SDK référence aux algorithmes

Les informations de cette page constituent une référence pour le développement de votre propre bibliothèque de chiffrement compatible avec le kit AWS Encryption SDK. Si vous ne créez pas votre propre bibliothèque de chiffrement compatible, vous n'aurez probablement pas besoin de ces informations.

Pour utiliser le AWS Encryption SDK dans l'un des langages de programmation pris en charge, voirLangages de programmation.

Pour la spécification qui définit les éléments d'une AWS Encryption SDK implémentation approprié e, voir la AWS Encryption SDK spécification dans GitHub.

Si vous créez votre propre bibliothèque capable de lire et d'écrire des textes chiffrés compatibles avec le AWS Encryption SDK, vous devez comprendre comment elle AWS Encryption SDK implémente les suites d'algorithmes prises en charge pour chiffrer les données brutes.

AWS Encryption SDK prend en charge les suites d'algorithmes suivantes. Toutes les suites d'algorithmes AES-GCM possèdent un vecteur d'initialisation de 12 octets et une balise d'authentification AES-GCM de 16 octets. La suite d'algorithmes par défaut varie en fonction de la AWS Encryption SDK version et de la politique d'engagement clé sélectionnée. Pour plus de détails, voir Politique d'engagement et suite d'algorithmes.

AWS Encryption SDK Suites d'algorithmes

ID de l'algorithme	Version du format du message	Algorithme de chiffrement	Longueur de la clé de données (bits)	Algorithme de dérivation de clé	Algorithme de signature	Algorithme d'engagement clé	Longueur des données de la suite d'algorithmes (octets)
05 78	0x02	AES-GCM	256	HKDF avec SHA-512	ECDSA avec P-384 et SHA-384	HKDF avec SHA-512	32 (engagement clé)
04 78	0x02	AES-GCM	256	HKDF avec SHA-512	Aucun	HKDF avec SHA-512	32 (engagement clé)
03 78	0x01	AES-GCM	256	HKDF avec SHA-384	ECDSA avec P-384 et SHA-384	Aucun	N/A
03 46	0x01	AES-GCM	192	HKDF avec SHA-384	ECDSA avec P-384 et SHA-384	Aucun	N/A
02 14	0x01	AES-GCM	128	HKDF avec SHA-256	ECDSA avec P-256 et SHA-256	Aucun	N/A
01 78	0x01	AES-GCM	256	HKDF avec SHA-256	Aucun	Aucun	N/A

ID de l'algorithme	Version du format du message	Algorithme de chiffrement	Longueur de la clé de données (bits)	Algorithme de dérivation de clé	Algorithme de signature	Algorithme d'engagement clé	Longueur des données de la suite d'algorithmes (octets)
`01 46`	0x01	AES-GCM	192	HKDF avec SHA-256	Aucun	Aucun	N/A
`01 14`	0x01	AES-GCM	128	HKDF avec SHA-256	Aucun	Aucun	N/A
`00 78`	0x01	AES-GCM	256	Aucun	Aucun	Aucun	N/A
`00 46`	0x01	AES-GCM	192	Aucun	Aucun	Aucun	N/A
`00 14`	0x01	AES-GCM	128	Aucun	Aucun	Aucun	N/A

ID de l'algorithme

Valeur hexadécimale de 2 octets qui identifie de manière unique une implémentation d'algorithme. Cette valeur est enregistrée dans l'en-tête du message chiffré.

Version du format du message

Version du format du message. Les suites d'algorithmes avec engagement clé utilisent le format de message version 2 (0x02). Les suites d'algorithmes sans engagement de clé utilisent le format de message version 1 (0x01).

Longueur des données de la suite d'algorithmes

Longueur en octets des données spécifiques à la suite d'algorithmes. Ce champ n'est pris en charge que dans le format de message version 2 (0x02). Dans le format de message version 2 (0x02), ces données apparaissent dans le `Algorithm suite data` champ de l'en-tête du message. Les suites d'algorithmes qui prennent en charge l'engagement des clés utilisent 32 octets pour la chaîne d'engagement des clés. Pour plus d'informations, consultez la section Algorithme d'engagement clé dans cette liste.

Longueur de la clé de données

Longueur de la clé de données en bits. Il AWS Encryption SDK prend en charge les clés 256 bits, 192 bits et 128 bits. La clé de données est générée par un trousseau de clés ou une clé principale.

Dans certaines implémentations, cette 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. Pour plus d'informations, voir Algorithme de dérivation des clés dans cette liste.

Algorithme de chiffrement

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

Algorithme d'engagement clé

Algorithme utilisé pour calculer la chaîne d'engagement clé. La sortie est stockée dans le `Algorithm suite data` champ de l'en-tête du message et est utilisée pour valider la clé de données pour l'engagement clé.

Pour une explication technique de l'ajout d'un engagement clé à une suite d'algorithmes, voir Key Committing AEADs dans Cryptology ePrint Archive.

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. AWS Encryption SDK Utilise le HKDF défini dans la RFC 5869.

Suites d'algorithmes sans engagement de clé (ID d'algorithme `01xx` —`03xx`)

- La fonction de hachage utilisée est SHA-384 ou SHA-256, selon la suite d'algorithmes.

- Pour l'étape d'extraction :
  - Aucune valeur salt n'est utilisée. Selon la RFC, le sel est défini sur une chaîne de zéros. La longueur de la chaîne est égale à la longueur de la sortie de la fonction de hachage, qui est de 48 octets pour SHA-384 et de 32 octets pour SHA-256.
  - Le matériel de saisie est la clé de données fournie par le trousseau de clés ou le fournisseur de la clé principale.
- Pour l'étape de développement :
  - La clé pseudo aléatoire en entrée est la sortie de l'étape d'extraction.
  - Les informations d'entrée sont une concaténation de l'ID de l'algorithme et de l'ID du message (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.

Suites d'algorithmes avec engagement clé (identifiant d'algorithme `04xx` et`05xx`)

- La fonction de hachage utilisée est SHA-512.
- Pour l'étape d'extraction :
  - Le sel est une valeur aléatoire cryptographique de 256 bits. Dans le format de message version 2 (0x02), cette valeur est stockée dans le `MessageID` champ.
  - Le matériel de saisie initial est la clé de données fournie par le porte-clés ou le fournisseur de la clé principale.
- 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.

Version du format du message

Version du format de message utilisé avec la suite d'algorithmes. Pour plus de détails, consultez Référence Format des messages du kit.

Algorithme de signature

Algorithme de signature utilisé pour générer une signature numérique sur l'en-tête et le corps du texte chiffré. AWS Encryption SDK utilise l'algorithme de signature numérique à courbe elliptique (ECDSA) avec les spécificités suivantes :

- La courbe elliptique utilisée est soit la courbe P-384, soit la courbe P-256, comme spécifié par l'ID de l'algorithme. Ces courbes sont définies dans le document Digital Signature Standard (DSS) (FIPS PUB 186-4).
- La fonction de hachage utilisée est la fonction SHA-384 (avec la courbe P-384) ou SHA-256 (avec la courbe P-256).

AWS Encryption SDK référence du vecteur d'initialisation

Les informations de cette page constituent une référence pour le développement de votre propre bibliothèque de chiffrement compatible avec le kit AWS Encryption SDK. Si vous ne créez pas votre propre bibliothèque de chiffrement compatible, vous n'aurez probablement pas besoin de ces informations.

Pour utiliser le AWS Encryption SDK dans l'un des langages de programmation pris en charge, voirLangages de programmation.

Pour la spécification qui définit les éléments d'une AWS Encryption SDK implémentation approprié e, voir la AWS Encryption SDK spécification dans GitHub.

Le AWS Encryption SDK fournit les vecteurs d'initialisation (IV) requis par toutes les suites d'algorithmes prises en charge. Le kit SDK utilise des numéros de séquence de cadre pour construire un vecteur d'initialisation afin qu'il ne soit pas possible que deux cadres du même message puissent avoir le même vecteur d'initialisation.

Chaque vecteur d'initialisation de 96 bits (12 octets) est construit à partir de deux tableaux d'octets de poids fort concaténés dans l'ordre suivant :

- 64 bits : 0 (réservé pour une utilisation ultérieure)
- 32 bits : numéro de séquence du cadre. Pour la balise d'authentification de l'en-tête, cette valeur est uniquement constituée de zéros.

Avant que ne soit disponible la mise en cache des clés de données, le kit AWS Encryption SDK utilisait toujours une nouvelle clé de données pour chiffrer chaque message, et générait tous les vecteurs d'initialisation de façon aléatoire. Les vecteurs d'initialisation générés de façon aléatoire étaient sûrs dans le cadre du chiffrement, car les clés de données n'étaient jamais été réutilisées. Lorsque le kit SDK a lancé la mise en cache des clés de données, qui réutilise intentionnellement les clés de données, nous avons modifié la façon dont le kit SDK génère les vecteurs d'initialisation.

Le fait d'utiliser des vecteurs d'initialisation déterministes qui ne peuvent pas se répéter au sein d'un message augmente considérablement le nombre d'appels pouvant être exécutés en toute sécurité dans le cadre d'une seule clé de données. En outre, les clés de données qui sont mises en cache utilisent toujours une suite d'algorithmes avec une fonction de dérivation de clés. L'utilisation d'un IV déterministe avec une fonction de dérivation de clé pseudo-aléatoire pour dériver des clés de chiffrement à partir d'une clé de données permet de chiffrer 2^32 AWS Encryption SDK messages sans dépasser les limites cryptographiques.

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.

Valeur	Longueur en octets	Interprété comme
"aws-kms-hierarchy"	17	Encodé en UTF-8
L'identifiant de la clé de branche	Variable	Encodé en UTF-8
La version de la clé de branche	16	Encodé en UTF-8
Contexte de chiffrement	Variable	Paires de valeurs clés codées en UTF-8

Historique du document pour le guide du AWS Encryption SDK développeur

Cette rubrique décrit les mises à jour importantes apportées au Guide du développeur AWS Encryption SDK .

Rubriques

Mises à jour récentes

Le tableau suivant décrit les modifications importantes apportées à cette documentation depuis novembre 2017. En plus des principales modifications répertoriées ici, 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
Disponibilité générale	Ajout de documentation pour le porte-clés AWS KMS ECDH et le porte-clés ECDH brut.	17 juin 2024
Kit SDK de chiffrement AWS pour Java version 3.x	Intègre Kit SDK de chiffrement AWS pour Java la bibliothèque des fournisseurs de matériaux. Ajoute le support pour les trousseaux de clés et le contexte de chiffrement requis CMM.	6 décembre 2023
AWS Encryption SDK pour .NET version 4.x	Ajoute la prise en charge du AWS KMS jeu de clés hiérarchique, du contexte de	12 octobre 2023

	chiffrement requis CMM et des jeux de clés RSA asymétriq ues. AWS KMS	
Disponibilité générale	Présentation du support AWS Encryption SDK pour .NET.	17 mai 2022
Modification de la documenta tion	Remplacez le AWS Key Management Service terme clé principale du client (CMK) par AWS KMS keyclé KMS.	30 août 2021
Disponibilité générale	Ajout du support pour AWS Key Management Service. (AWS KMS) Clés multirégi onales. Les clés multirégi onales 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
Disponibilité générale	Ajout et mise à jour de la documentation sur le processus amélioré de déchiffrement des messages.	11 mai 2021
Disponibilité générale	Documentation ajoutée et mise à jour pour la version de disponibilité générale de AWS Encryption CLI version 1.8. x pour remplacer la version 1.7 de la CLI de AWS chiffreme nt. x, et AWS Encryption CLI 2.1. x pour remplacer AWS Encryption CLI 2.0. x.	27 octobre 2020

Disponibilité générale	Documentation ajoutée et mise à jour pour la mise à disposition générale des AWS Encryption SDK versions 1.7. x et 2.0. x, y compris un guide des meilleures pratiques, un guide de migration, des concepts mis à jour, des sujets de langage de programmation mis à jour, une référence de suites d'algorithmes mise à jour, une référence de format de message mise à jour et un nouvel exemple de format de message.	24 septembre 2020
Disponibilité générale	Ajout et mise à jour de la documentation pour la version de disponibilité générale du Kit SDK de chiffrement AWS pour JavaScript.	1 octobre 2019
Version préliminaire	Ajout et mise à jour de la documentation de la version bêta publique du Kit SDK de chiffrement AWS pour JavaScript.	21 juin 2019
Disponibilité générale	Ajout et mise à jour de la documentation pour la version de disponibilité générale du Kit SDK de chiffrement AWS pour C.	16 mai 2019

Version préliminaire	Documentation ajoutée de la version préliminaire du Kit SDK de chiffrement AWS pour C.	le 5 février 2019
Nouvelle version	Ajout de la documentation de l'interface de ligne de commande pour le kit AWS Encryption SDK.	le 20 novembre 2017

Mises à jour antérieures

Le tableau suivant décrit les modifications importantes apportées au Guide du AWS Encryption SDK développeur avant novembre 2017.

Modification	Description	Date
Nouvelle version	Ajout du chapitre Mise en cache des clés de données concernant la nouvelle fonctionnalité. Ajout de la rubrique the section called “Référence Vecteur d'initialisation du kit” qui explique que le kit SDK est passé de la génération de vecteurs d'initialisation aléatoires à la construction de vecteurs d'initialisation déterministes. Ajout d'une the section called “Concepts” rubrique expliquant les concepts, notamment	31 juillet 2017

Modification	Description	Date
	le nouveau gestionnaire de matériel cryptographique.	
Mettre à jour	Extension de la documentation Référence Format des messages du kit avec une nouvelle section AWS Encryption SDK référence. Ajout d'une section sur le AWS Encryption SDK Suites d'algorithmes prises en charge.	21 mars 2017
Nouvelle version	Le supporte AWS Encryption SDK désormais le langage Python de programmation, en plus deJava.	21 mars 2017
Première version	Publication initiale de la AWS Encryption SDK et de cette documentation.	22 mars 2016

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.