



Guia do Desenvolvedor

AWS Encryption SDK



AWS Encryption SDK: Guia do Desenvolvedor

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

As marcas comerciais e imagens comerciais da Amazon não podem ser usadas no contexto de nenhum produto ou serviço que não seja da Amazon, nem de qualquer maneira que possa gerar confusão entre os clientes ou que deprecie ou desprestige a Amazon. Todas as outras marcas comerciais que não pertencem à Amazon pertencem a seus respectivos proprietários, que podem ou não ser afiliados, patrocinados pela Amazon ou ter conexão com ela.

Table of Contents

O que é o AWS Encryption SDK?	1
Desenvolvido em repositórios de código aberto	2
Compatibilidade com bibliotecas e serviços de criptografia	3
Suporte e manutenção	4
Saiba mais	5
Enviar comentários	6
Conceitos	6
Criptografia de envelope	7
Chave de dados	9
Chave de encapsulamento	10
Tokens de autenticação e provedores de chaves mestras	11
Contexto de criptografia	12
Mensagem criptografada	13
Pacote de algoritmos	14
Gerenciador de material de criptografia	14
Criptografia simétrica e assimétrica	15
Confirmação de chave	16
Política de compromisso	17
Assinaturas digitais	19
Saiba como o SDK funciona	20
Como o AWS Encryption SDK criptografa os dados	20
Como o AWS Encryption SDK descriptografa e criptografa uma mensagem	21
Pacotes de algoritmos compatíveis	22
Recomendado: AES-GCM com derivação de chave, assinatura e confirmação de chave	22
Outros pacotes de algoritmos compatíveis	23
Interagindo com o AWS KMS	25
Práticas recomendadas	27
Como configurar o SDK	32
Seleção de uma linguagem de programação	32
Seleção de chaves de encapsulamento	32
Usar AWS KMS keys multirregionais	34
Escolher um pacote de algoritmo	56
Limitar as chaves de dados criptografadas	65
Criação de um filtro de descoberta	69

Como definir uma política de compromisso	71
Trabalhar com streaming de dados	72
Armazenamento em cache de chaves de dados	72
Usar tokens de autenticação	73
Como os tokens de autenticação funcionam	74
Compatibilidade dos tokens de autenticação	75
Requisitos variados para tokens de autenticação de criptografia	76
Tokens de autenticação e provedores de chaves mestras compatíveis	76
Como escolher um token de autenticação	77
AWS KMS chaveiros	78
AWS KMS Chaveiros hierárquicos	98
AWS KMS chaveiros ECDH	124
Tokens de autenticação AES Raw	129
Tokens de autenticação brutos do RSA	133
Chaveiros ECDH brutos	139
Multitokens de autenticação	145
Linguagens de programação	150
C	150
Instalação	151
Uso do C SDK	152
Exemplos	157
.NET	165
Instalar e compilar	167
Depuração	167
Tokens de autenticação do AWS KMS	168
CMM do contexto de criptografia necessário	171
Exemplos	173
Java	182
Pré-requisitos	182
Instalação	184
Tokens de autenticação do AWS KMS	185
CMM do contexto de criptografia necessário	188
Exemplos	190
JavaScript	203
Compatibilidade	204
Instalação	206

Modules	207
Exemplos	210
Python	217
Pré-requisitos	217
Instalação	217
Exemplos	219
Interface da linha de comando	230
Instalar a CLI do	231
Como usar a CLI	235
Exemplos	249
Referência da sintaxe e de parâmetros	274
Versões	288
Armazenamento em cache de chaves de dados	292
Como usar o armazenamento em cache de chaves de dados	293
Usando o cache de chaves de dados: S tep-by-step	294
Armazenamento em cache de chaves de dados de exemplo: criptografar uma string	302
Definir limites de segurança do cache	318
Detalhes do armazenamento em cache de chaves de dados	320
Como o armazenamento em cache de chaves de dados funciona	320
Criar um cache de material de criptografia	324
Criar um gerenciador de material de criptografia de armazenamento em cache	325
O que é uma entrada de chave de dados em cache?	326
Contexto de criptografia: como selecionar entradas do cache	327
Meu aplicativo está usando chaves de dados armazenadas em cache?	327
Exemplo de armazenamento em cache de chaves de dados	328
Resultados do cache local	329
Código de exemplo	330
Modelo AWS CloudFormation	342
Versões do AWS Encryption SDK	357
C	358
C# /.NET	359
Interface de linha de comando (CLI)	359
Java	362
JavaScript	364
Python	365
Detalhes da versão	367

Versões anteriores à 1.7.x	367
Versão 1.7.x	368
Versão 2.0x	371
Versão 2.2x	372
Versão 2.3x	373
Como migrar seu AWS Encryption SDK	375
Como migrar e implantar	377
Etapa 1: atualize a aplicação para a versão 1.x mais recente	377
Etapa 2: atualize a aplicação para a versão mais recente	378
Como atualizar provedores de chaves mestras do AWS KMS	380
Migração para o modo estrito	381
Migrar para o modo de descoberta	385
Atualizar tokens de autenticação do AWS KMS	388
Como definir sua política de compromisso	391
Como definir sua política de compromisso	392
Solução de problemas de migração para as versões mais recentes	400
Objetos descontinuados ou removidos	401
Conflito de configuração: política de compromisso e pacote de algoritmos	401
Conflito de configuração: política de compromisso e texto cifrado	402
Falha na validação do confirmação de chave	403
Outras falhas de criptografia	403
Outras falhas de decriptografia	403
Considerações sobre reversão	404
Perguntas frequentes	405
Referência	411
Referência do formato de mensagens	411
Estrutura do cabeçalho	412
Estrutura do corpo	421
Estrutura do rodapé	426
Exemplos de formatos de mensagens	427
Dados emoldurados (formato de mensagem versão 1)	427
Dados emoldurados (formato de mensagem versão 2)	431
Dados não emoldurados (formato de mensagem versão 1)	433
Referência de AAD de corpo	437
Referência de algoritmos	439
Referência do vetor de inicialização	444

AWS KMS Detalhes técnicos do chaveiro hierárquico	445
Histórico do documento	446
Atualizações recentes	446
Atualizações anteriores	449
.....	cdli

O que é o AWS Encryption SDK?

O AWS Encryption SDK é uma biblioteca de criptografia do lado do cliente projetada para facilitar a criptografia e a descriptografia de dados para todos usando os padrões e as melhores práticas do setor. Ele permite que você se concentre na funcionalidade principal do aplicativo, em vez de sobre como melhor criptografar e descriptografar os dados. O AWS Encryption SDK é fornecido gratuitamente sob a licença do Apache 2.0.

O AWS Encryption SDK responde a perguntas como as seguintes para você:

- Qual algoritmo de criptografia devo usar?
- Como, ou em qual modo, devo usar esse algoritmo?
- Como faço para gerar a chave de criptografia?
- Como faço para proteger a chave de criptografia e onde devo armazená-la?
- Como posso tornar meus dados criptografados portáteis?
- Como faço para garantir que o destinatário pretendido possa ler meus dados criptografados?
- Como posso garantir que meus dados criptografados não sejam modificados entre o momento em que são gravados e o momento em que são lidos?
- Como faço para usar as chaves de dados que o AWS KMS retorna?

Com o AWS Encryption SDK, você define um [provedor de chaves mestras](#) (Java ou Python) ou um [token de autenticação](#) (C, C#.NET e JavaScript) que determine quais chaves de encapsulamento usar para proteger seus dados. Você criptografa e descriptografa os dados usando os métodos diretos fornecidos pelo AWS Encryption SDK. O AWS Encryption SDK cuida do resto.

Sem o AWS Encryption SDK, você pode despendar mais esforço na criação de uma solução de criptografia do que na funcionalidade principal do aplicativo. O AWS Encryption SDK responde a essas perguntas, fornecendo as funcionalidades a seguir.

Uma implementação padrão que segue as melhores práticas de criptografia

Por padrão, o AWS Encryption SDK gera uma chave de dados exclusiva para cada objeto de dados que criptografa. Isso segue a melhor prática de criptografia de usar chaves de dados exclusivas para cada operação de criptografia.

O AWS Encryption SDK criptografa os dados usando um algoritmo de chave simétrica seguro e autenticado. Para obter mais informações, consulte [the section called “Pacotes de algoritmos compatíveis”](#).

Uma estrutura para proteção de chaves de dados com chaves de encapsulamento

O AWS Encryption SDK protege as chaves de dados que criptografam seus dados criptografando-as sob uma ou mais chaves de encapsulamento. Ao fornecer uma estrutura para criptografar chaves de dados com mais de uma chave de encapsulamento, o AWS Encryption SDK ajuda a tornar seus dados criptografados portáteis.

Por exemplo, criptografe dados com uma entrada de AWS KMS key no AWS KMS e uma chave do seu HSM on-premises. É possível usar qualquer uma das duas chaves de encapsulamento para descriptografar os dados, caso alguma não esteja disponível ou o chamador não tenha permissão para usar as duas chaves.

Uma mensagem formatada que armazena chaves de dados criptografadas com os dados criptografados

O AWS Encryption SDK armazena os dados criptografados e a chave de dados criptografada juntos em uma [mensagem criptografada](#) que usa um formato de dados definido. Isso significa que você não precisa controlar ou proteger as chaves de dados que criptografam seus dados pois o AWS Encryption SDK faz isso para você.

Algumas implementações de linguagem do AWS Encryption SDK exigem um AWS SDK, mas o AWS Encryption SDK não requer uma Conta da AWS e não depende de nenhum serviço da AWS. Você só necessitará de uma Conta da AWS se optar por usar uma [AWS KMS keys](#) para proteger seus dados.

Desenvolvido em repositórios de código aberto

O SDK de criptografia de banco de dados do AWS Encryption SDK é desenvolvido em repositórios de código aberto no GitHub. É possível usar esses repositórios para visualizar o código, ler e enviar problemas e encontrar informações específicas para sua implementação de linguagem.

- AWS Encryption SDK for C — [aws-encryption-sdk-c](#)
- AWS Encryption SDK para .NET — diretório [aws-encryption-sdk-net](#) do repositório `aws-encryption-sdk-dafny`.
- CLI de criptografia da AWS — [aws-encryption-sdk-cli](#)

- AWS Encryption SDK for Java — [aws-encryption-sdk-java](#)
- AWS Encryption SDK para JavaScript — [aws-encryption-sdk-javascript](#)
- AWS Encryption SDK for Python — [aws-encryption-sdk-python](#)

Compatibilidade com bibliotecas e serviços de criptografia

O AWS Encryption SDK é compatível com várias [linguagens de programação](#). As implementações de linguagem são interoperáveis. É possível criptografar com uma implementação de linguagem e descriptografar com outra. A interoperabilidade pode estar sujeita às restrições de linguagem. Em caso afirmativo, essas restrições estarão descritas no tópico sobre a implementação de linguagem. Além disso, ao criptografar e descriptografar, é necessário usar tokens de autenticação compatíveis ou chaves mestras e provedores de chaves mestras. Para obter mais detalhes, consulte [the section called “Compatibilidade dos tokens de autenticação”](#).

No entanto, o AWS Encryption SDK não pode interoperar com outras bibliotecas. Como cada biblioteca retorna dados criptografados em um formato diferente, você não pode criptografar com uma biblioteca e descriptografar com outra.

DynamoDB Encryption Client e criptografia do lado do cliente do Amazon S3

O AWS Encryption SDK não pode descriptografar dados criptografados pelo [DynamoDB Encryption Client](#) ou pela [criptografia do lado do cliente do Amazon S3](#). Essas bibliotecas não podem descriptografar a [mensagem criptografada](#) retornada pelo AWS Encryption SDK.

AWS Key Management Service (AWS KMS)

O AWS Encryption SDK pode usar [AWS KMS keys](#) e [chaves de dados](#) para proteger seus dados, incluindo chaves do KMS multirregionais. Por exemplo, você pode configurar o AWS Encryption SDK para criptografar os dados sob uma ou mais AWS KMS keys em sua Conta da AWS. No entanto, você deve usar o AWS Encryption SDK para descriptografar esses dados.

O AWS Encryption SDK não pode descriptografar o texto cifrado retornado pelas operações [Encrypt](#) ou [ReEncrypt](#) do AWS KMS. Da mesma forma, a operação [Decrypt](#) do AWS KMS não pode descriptografar a [mensagem criptografada](#) que o AWS Encryption SDK retorna.

O AWS Encryption SDK oferece suporte somente para [chaves de criptografia simétricas do KMS](#). Não é possível usar uma [chave assimétrica do KMS](#) para criptografia ou assinatura no AWS Encryption SDK. O AWS Encryption SDK gera suas próprias chaves de assinatura ECDSA para [pacotes de algoritmos](#) que assinam mensagens.

Para obter ajuda para decidir qual biblioteca ou serviço usar, consulte [Como escolher uma ferramenta ou serviço de criptografia](#) em Serviços e ferramentas de criptografia da AWS.

Suporte e manutenção

O AWS Encryption SDK usa a mesma [política de manutenção](#) que o SDK da AWS e as ferramentas usam, incluindo suas fases de versionamento e ciclo de vida. Como [prática recomendada](#), você deve usar a versão mais recente do AWS Encryption SDK para sua linguagem de programação e atualizá-la à medida que novas versões forem lançadas. Quando uma versão exige alterações significativas, como a atualização de versões do AWS Encryption SDK anteriores à 1.7.x para as versões 2.0.x e posteriores, fornecemos [instruções detalhadas](#) para ajudar você.

Cada implementação de linguagem de programação do AWS Encryption SDK é desenvolvida em um repositório separado de código aberto do GitHub. É provável que o ciclo de vida e a fase do suporte de cada versão variem entre os repositórios. Por exemplo, uma determinada versão do AWS Encryption SDK pode estar na fase de disponibilidade geral (suporte completo) em determinada linguagem de programação, mas estar na fase de fim do suporte em uma linguagem de programação diferente. Recomendamos que você use uma versão totalmente compatível sempre que possível e evite versões que já não sejam compatíveis.

Para encontrar a fase do ciclo de vida das versões do AWS Encryption SDK para sua linguagem de programação, consulte o arquivo `SUPPORT_POLICY.rst` em cada repositório do AWS Encryption SDK.

- AWS Encryption SDK for C: [SUPPORT_POLICY.rst](#)
- AWS Encryption SDK para .NET: [Support_policy.rst](#)
- CLI de criptografia do AWS: [Support_policy.rst](#)
- AWS Encryption SDK for Java: [SUPPORT_POLICY.rst](#)
- AWS Encryption SDK para JavaScript: [SUPPORT_POLICY.rst](#)
- AWS Encryption SDK for Python: [SUPPORT_POLICY.rst](#)

Para obter mais informações, consulte [Versões do AWS Encryption SDK](#) e a [Política de manutenção de SDKs e ferramentas da AWS](#) no Guia de referência de SDKs e ferramentas da AWS.

Saiba mais

Se estiver procurando mais informações sobre o AWS Encryption SDK e a criptografia do lado do cliente, tente essas fontes.

- Para obter ajuda com os termos e conceitos usados neste SDK, consulte [Conceitos no AWS Encryption SDK](#).
- Para obter as diretrizes de práticas recomendadas, consulte [Melhores práticas para a AWS Encryption SDK](#).
- Para obter informações sobre como o SDK funciona, consulte [Saiba como o SDK funciona](#).
- Para obter exemplos que mostram como configurar opções no AWS Encryption SDK, consulte [Configurar o AWS Encryption SDK](#).
- Para obter informações técnicas, consulte a [Referência](#).
- Para obter a especificação técnica do AWS Encryption SDK, consulte [Especificação do AWS Encryption SDK](#) no GitHub.
- Para obter respostas às suas perguntas sobre como usar o AWS Encryption SDK, leia e poste no [Fórum de discussão sobre ferramentas de criptografia da AWS](#).

Para obter informações sobre as implementações do AWS Encryption SDK em diferentes linguagens de programação.

- C: consulte [AWS Encryption SDK for C](#), a documentação do AWS Encryption SDK [C](#) e o repositório [aws-encryption-sdk-c](#) no GitHub.
- C#/.NET: consulte [AWS Encryption SDK para .NET](#) e o diretório [aws-encryption-sdk-net](#) do repositório [aws-encryption-sdk-dafny](#) no GitHub.
- Interface de linha de comando: consulte [Interface da linha de comando do AWS Encryption SDK](#), [Leia os documentos da](#) CLI de criptografia da AWS e o repositório [aws-encryption-sdk-cli](#) no GitHub.
- Java: consulte [AWS Encryption SDK for Java](#), o AWS Encryption SDK [Javadoc](#) e o repositório [aws-encryption-sdk-java](#) no GitHub.
- JavaScript: consulte [the section called “JavaScript”](#) e o repositório [aws-encryption-sdk-javascript](#) no GitHub.
- Python: consulte [AWS Encryption SDK for Python](#), a AWS Encryption SDK documentação do Python [do](#) , e o repositório do [aws-encryption-sdk-python](#) no GitHub.

Enviar comentários

Os seus comentários são bem-vindos. Se você tiver uma pergunta ou comentário, ou um problema a relatar, use os seguintes recursos.

- Se você descobrir uma potencial vulnerabilidade de segurança no AWS Encryption SDK, [notifique a segurança da AWS](#). Não crie um problema público no GitHub.
- Para fornecer comentários sobre o AWS Encryption SDK, registre um problema no repositório do GitHub para a linguagem de programação que você está usando.
- Para fornecer comentários sobre esta documentação, use os links Feedback nesta página. Você também pode registrar um problema ou contribuir para o [aws-encryption-sdk-docs](#), o repositório de código aberto desta documentação no GitHub.

Conceitos no AWS Encryption SDK

Esta seção apresenta os conceitos usados no AWS Encryption SDK e fornece um glossário e uma referência. Ele foi projetado para ajudar você a entender como o AWS Encryption SDK funciona e os termos que usamos para descrevê-lo.

Precisa de ajuda?

- Saiba como o AWS Encryption SDK usa a [criptografia envelopada](#) para proteger seus dados.
- Saiba mais sobre os elementos da criptografia envelopada: as [chaves de dados](#) que protegem seus registros e as [chaves de encapsulamento](#) que protegem suas chaves de dados.
- Saiba mais sobre os [tokens de autenticação](#) e os [provedores de chaves mestras](#) que determinam quais chaves de encapsulamento você usa.
- Saiba mais sobre o [contexto de criptografia](#) que adiciona integridade ao seu processo de criptografia. É opcional, mas é uma prática recomendada que incentivamos.
- Saiba mais sobre a [mensagem criptografada](#) que os métodos de criptografia retornam.
- Então você está pronto para usar o AWS Encryption SDK em sua [linguagem de programação](#) preferida.

Tópicos

- [Criptografia de envelope](#)
- [Chave de dados](#)

- [Chave de encapsulamento](#)
- [Tokens de autenticação e provedores de chaves mestras](#)
- [Contexto de criptografia](#)
- [Mensagem criptografada](#)
- [Pacote de algoritmos](#)
- [Gerenciador de material de criptografia](#)
- [Criptografia simétrica e assimétrica](#)
- [Confirmação de chave](#)
- [Política de compromisso](#)
- [Assinaturas digitais](#)

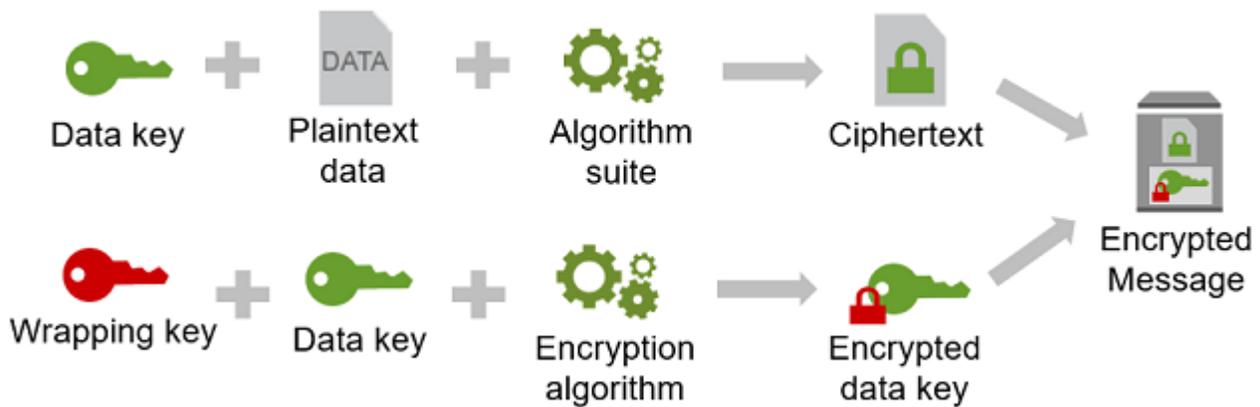
Criptografia de envelope

A segurança dos dados criptografados depende em parte da proteção da chave de dados que pode descriptografá-los. Uma prática recomendada aceita para proteger a chave de dados é criptografá-la. Para fazer isso, você precisa de outra chave de criptografia, conhecida como chave de criptografia de chave ou [chave de encapsulamento](#). Essa prática de uso de uma chave de encapsulamento para criptografar chaves de dados é conhecida como criptografia envelopada.

Proteção de chaves de dados

O AWS Encryption SDK criptografa cada mensagem com uma chave de dados exclusiva. Em seguida, ele criptografa cada chave de dados sob a chave de encapsulamento especificada. Ele armazena as chaves de dados criptografadas junto com os dados criptografados na mensagem criptografada que as operações de criptografia retornam.

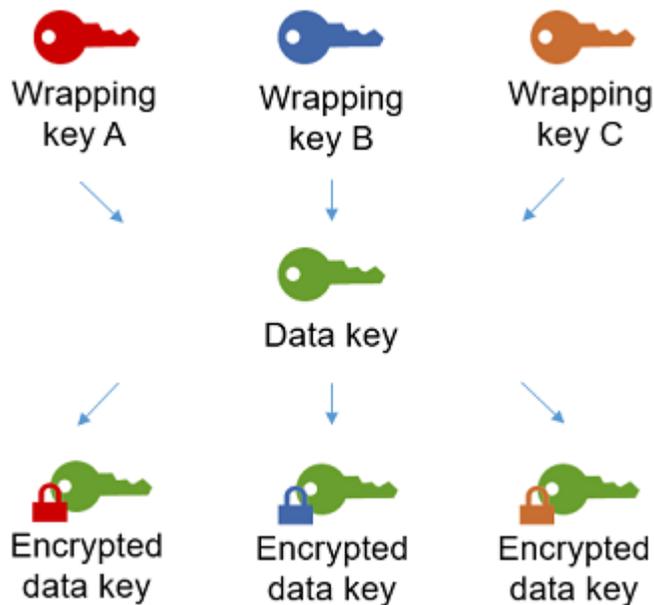
Para especificar sua chave de encapsulamento, use um [token de autenticação](#) ou um [provedor de chave mestra](#).



Criptografar os mesmos dados com várias chaves de encapsulamento

É possível criptografar a chave de dados sob várias chaves de encapsulamento. Talvez você queira fornecer chaves de encapsulamento distintas para usuários diferentes ou chaves de encapsulamento de tipos variados ou em locais diferentes. Cada uma das chaves de encapsulamento criptografa a mesma chave de dados. O AWS Encryption SDK armazena todas as chaves de dados criptografadas junto com os dados criptografados na mensagem criptografada.

Para descriptografar os dados, você precisa fornecer pelo menos uma chave de encapsulamento que possa descriptografar uma das chaves de dados criptografadas.



Combinação de pontos fortes de vários algoritmos

Para criptografar seus dados, por padrão, o AWS Encryption SDK usa um [conjunto de algoritmos](#) com criptografia simétrica AES-GCM, uma função de derivação de chave (HKDF) baseada em HMAC e assinatura ECDSA. Para criptografar a chave de dados, você pode especificar um [algoritmo de criptografia simétrico ou assimétrico](#) apropriado à sua chave de encapsulamento.

Em geral, os algoritmos de criptografia de chaves simétricas são mais rápidos e produzem textos cifrados menores que a criptografia de chave pública ou assimétrica. No entanto, os algoritmos de chave pública fornecem separação inerente de funções e gerenciamento de chaves mais fácil. Para combinar as forças de cada um, você pode criptografar dados brutos com criptografia de chave simétrica e, em seguida, criptografar a chave de dados com criptografia de chave pública.

Chave de dados

A chave de dados é uma chave de criptografia que o AWS Encryption SDK usa para criptografar os dados. Cada chave de dados é uma matriz de bytes que cumpre os requisitos para chaves criptográficas. A menos que você esteja usando [armazenamento em cache de chaves de dados](#), o AWS Encryption SDK usará uma chave de dados exclusiva para criptografar cada mensagem.

Você não precisa especificar, gerar, implementar, estender, proteger ou usar chaves de dados. O AWS Encryption SDK faz esse trabalho para você quando você chama as operações de criptografia e descriptografia.

Para proteger suas chaves de dados, o AWS Encryption SDK as criptografa sob uma ou mais chaves de criptografia de chaves, conhecidas como [chaves de encapsulamento](#) ou chaves mestras. Depois que o AWS Encryption SDK usa suas chaves de dados em texto simples para criptografar dados, ele as remove da memória assim que possível. Depois, ele armazena as chaves de dados criptografadas junto com os dados criptografados na [mensagem criptografada](#) que as operações de criptografia retornam. Para obter detalhes, consulte [the section called “Saiba como o SDK funciona”](#).

Tip

No AWS Encryption SDK, diferenciamos chaves de dados de chaves de criptografia de dados. Vários dos [pacotes de algoritmos](#), incluindo o pacote padrão, usam uma [função de derivação de chaves](#) que impede que a chave de dados atinja seus limites de criptografia. A função de derivação de chaves usa a chave de dados como entrada e retorna uma chave de criptografia de dados que é realmente usada para criptografar os dados. Por esse motivo,

sempre dizemos que os dados são criptografados "sob" uma chave de dados em vez de "pela" chave de dados.

Cada chave de dados criptografada inclui metadados, incluindo o identificador da chave de encapsulamento que a criptografou. Esses metadados facilitam a identificação de chaves de encapsulamento válidas durante a descriptografia pelo AWS Encryption SDK.

Chave de encapsulamento

Uma chave de encapsulamento é uma chave de criptografia que o AWS Encryption SDK usa para criptografar a [chave de dados](#) que criptografa seus registros. Cada chave de dados em texto simples pode ser criptografada sob uma ou mais chaves mestras. Você determina quais chaves de encapsulamento serão usadas para proteger seus dados ao configurar um [token de autenticação](#) ou um [provedor de chave mestra](#).

Note

A chave de encapsulamento refere-se às chaves em um token de autenticação ou provedor de chave mestra. A chave mestra geralmente está associada à classe `MasterKey` que você instancia ao usar um provedor de chave mestra.

O AWS Encryption SDK é compatível com várias ou chaves de encapsulamento usadas comumente, como a [AWS KMS keys](#) simétricas do AWS Key Management Service (AWS KMS) (incluindo [chaves KMS multirregionais](#)), chaves brutas do AES-GCM (Advanced Encryption Standard/Galois Counter Mode) e chaves RSA brutas. Você também pode estender ou implementar suas próprias chaves de encapsulamento.

Quando você usa a criptografia envelopada, precisa proteger suas chaves de encapsulamento contra acesso não autorizado. É possível fazer isso de uma das seguintes maneiras:

- Use um serviço web projetado para essa finalidade, como o [AWS Key Management Service \(AWS KMS\)](#).
- Use um [hardware security module \(HSM - módulo de segurança de hardware\)](#), como os oferecidos pelo [AWS CloudHSM](#).
- Use outras ferramentas e serviços de gerenciamento de chaves.

Se você não tiver um sistema de gerenciamento de chaves, recomendamos o AWS KMS. O AWS Encryption SDK se integra com o AWS KMS para ajudar a proteger e usar suas chaves de encapsulamento. No entanto, o AWS Encryption SDK não requer o AWS nem qualquer outro serviço da AWS.

Tokens de autenticação e provedores de chaves mestras

Para especificar as chaves de agrupamento que você usa para criptografia e descriptografia, use um chaveiro (C, C# / .NET e JavaScript) ou um provedor de chave mestra (Java, Python, CLI). Você pode usar os tokens de autenticação e os provedores de chaves mestras que o AWS Encryption SDK fornece ou criar suas próprias implementações. O AWS Encryption SDK fornece tokens de autenticação e provedores de chaves mestras compatíveis entre si, sujeitos a restrições de linguagem. Para obter detalhes, consulte [Compatibilidade dos tokens de autenticação](#).

Um token de autenticação gera, criptografa e descriptografa chaves de dados. Ao definir um token de autenticação, você pode especificar as [chaves de encapsulamento](#) que criptografam suas chaves de dados. A maioria dos tokens de autenticação especificam pelo menos uma chave de encapsulamento ou um serviço que fornece e protege chaves de encapsulamento. Você também pode definir um token de autenticação sem chaves de encapsulamento ou um token de autenticação mais complexo com opções de configuração adicionais. Para obter ajuda para escolher e usar os tokens de autenticação definidos pelo AWS Encryption SDK, consulte [Usar tokens de autenticação](#). Os chaveiros são compatíveis com C, C# / .NET e versão 3. JavaScript x do AWS Encryption SDK for Java.

Um provedor de chave mestra é uma alternativa a um token de autenticação. O provedor de chave mestra retorna as chaves de encapsulamento (ou chaves mestras) que você especificar. Cada chave mestra é associada a um provedor de chaves mestras, mas um provedor de chaves mestras normalmente fornece várias chaves mestras. Os provedores de chaves mestras são compatíveis com Java, Python e com a CLI de criptografia da AWS.

Você deve especificar um token de autenticação (ou provedor de chave mestra) para criptografia. Você pode especificar o mesmo token de autenticação (ou provedor de chave mestra), ou um diferente, para descriptografia. Ao criptografar, o AWS Encryption SDK usa todas as chaves de encapsulamento que você especifica para criptografar a chave de dados. Ao descriptografar, o AWS Encryption SDK usa somente as chaves de encapsulamento que você especificar para descriptografar uma chave de dados criptografada. Especificar chaves de encapsulamento para decodificação é opcional, mas é uma [prática recomendada. do AWS Encryption SDK](#).

Para obter detalhes sobre como especificar chaves de encapsulamento, consulte [Seleção de chaves de encapsulamento](#)

Contexto de criptografia

Para melhorar a segurança de suas operações de criptografia, inclua um [contexto de criptografia](#) em todas as solicitações para criptografar dados. O uso de um contexto de criptografia é opcional, mas é uma melhor prática de criptografia que recomendamos.

Um contexto de criptografia é um conjunto de pares de chave-valor que contêm dados autenticados adicionais arbitrários e não secretos. O contexto de criptografia pode conter todos os dados que você escolher, mas geralmente consiste em dados que são úteis para registro em log e rastreamento, como dados sobre o tipo de arquivo, a finalidade ou a propriedade. Quando você criptografa dados, o contexto de criptografia é associado de maneira criptográfica aos dados criptografados de forma que o mesmo contexto de criptografia seja necessário para descriptografar os dados. O AWS Encryption SDK inclui o contexto de criptografia em texto simples no cabeçalho da [mensagem criptografada](#) retornada por ele.

O contexto de criptografia usado pelo AWS Encryption SDK consiste no contexto de criptografia especificado e um par de chaves públicas adicionado pelo [gerenciador de materiais criptográficos](#) (CMM). Especificamente, sempre que você usar um [algoritmo de criptografia com assinatura](#), o CMM adicionará um par de nome/valor ao contexto de criptografia consistindo em um nome reservado, `aws-crypto-public-key`, e um valor representando a chave de verificação pública. O nome `aws-crypto-public-key` no contexto de criptografia é reservado pelo AWS Encryption SDK e não pode ser usado como um nome em nenhum outro par no contexto de criptografia. Para obter detalhes, consulte [AAD](#) na Referência do formato de mensagens.

O exemplo de contexto de criptografia a seguir consiste nos dois pares de contexto de criptografia especificados na solicitação e no par de chaves públicas adicionado pelo CMM.

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

Para descriptografar os dados, você passa a mensagem criptografada. Como o AWS Encryption SDK pode extrair o contexto de criptografia do cabeçalho da mensagem criptografada, não é necessário fornecer o contexto separadamente. No entanto, o contexto de criptografia pode ajudar a confirmar se você está descriptografando a mensagem criptografada correta.

- Na [interface de linha de comando do AWS Encryption SDK \(CLI\)](#), se você fornecer um contexto de criptografia em um comando de descriptografia, a CLI verificará se os valores estão presentes no contexto da mensagem criptografada antes de retornar os dados em texto simples.
- Em outras implementações de linguagens de programação, a resposta de descriptografia inclui o contexto de criptografia e os dados em texto simples. A função de descriptografia em seu aplicativo sempre deve verificar se o contexto de criptografia na resposta de descriptografia inclui o contexto de criptografia na solicitação de criptografia (ou um subconjunto) antes de retornar os dados em texto simples.

Note

Com a [versão 4. x do AWS Encryption SDK para .NET](#) e [versão 3. x do AWS Encryption SDK for Java](#), você pode exigir um contexto de criptografia em todas as solicitações de criptografia com o contexto de criptografia necessário CMM.

Ao escolher um contexto de criptografia, lembre-se de que ele não é um segredo. O contexto de criptografia é exibido em texto não criptografado no cabeçalho da [mensagem criptografada](#) retornada pelo AWS Encryption SDK. Se você estiver usando o AWS Key Management Service, o contexto de criptografia também poderá ser exibido em texto não criptografado em registros e logs de auditoria, como o AWS CloudTrail.

Para obter exemplos de envio e verificação de um contexto de criptografia no seu código, consulte os exemplos da [linguagem de programação](#) de sua preferência.

Mensagem criptografada

Quando você criptografa dados com o AWS Encryption SDK, ele retorna uma mensagem criptografada.

Uma mensagem criptografada é uma [estrutura de dados formatados](#) portátil que inclui os dados criptografados junto com as cópias criptografadas das chaves de dados, o ID do algoritmo e, opcionalmente, um [contexto de criptografia](#) e [uma assinatura digital](#). As operações de criptografia no AWS Encryption SDK retornam uma mensagem criptografada, e as operações de descriptografia usam uma mensagem criptografada como entrada.

A combinação de dados criptografados e de suas chaves de dados criptografadas simplifica a operação e elimina a necessidade de armazenar e gerenciar chaves de dados criptografadas independentemente dos dados que elas criptografam.

Para obter informações técnicas sobre a mensagem criptografada, consulte [Formato da mensagem criptografada](#).

Pacote de algoritmos

O AWS Encryption SDK usa um pacote de algoritmos para criptografar e assinar os dados na [mensagem criptografada](#) que as operações de criptografia e descriptografia retornam. O AWS Encryption SDK é compatível com vários [pacotes de algoritmos](#). Todos os conjuntos compatíveis usam o Advanced Encryption Standard (AES) como o algoritmo principal e o combinam com outros algoritmos e valores.

O AWS Encryption SDK estabelece um pacote de algoritmos recomendado como o padrão para todas as operações de criptografia. O padrão pode ser alterado conforme os padrões e as práticas recomendadas são aprimoradas. Você pode especificar um pacote de algoritmos alternativo em solicitações de criptografia de dados ou ao criar um [gerenciador de materiais criptográficos \(CMM\)](#), mas, a menos que um alternativo seja necessário para sua situação, é melhor usar o padrão. O padrão atual é AES-GCM com uma [função de derivação de extract-and-expand chave \(HKDF\) baseada em HMAC](#), [compromisso de chave](#), uma assinatura de [algoritmo de assinatura digital de curva elíptica \(ECDSA\)](#) e uma chave de criptografia de 256 bits.

Se sua aplicação exigir alta performance e os usuários que criptografam dados e aqueles que os descriptografam forem igualmente confiáveis, considere especificar um pacote de algoritmos sem uma assinatura digital. No entanto, recomendamos fortemente um pacote de algoritmos que inclua confirmação de chave e uma função de derivação de chave. Os pacotes de algoritmos que não têm esses atributos são compatíveis apenas para compatibilidade com versões anteriores.

Gerenciador de material de criptografia

O gerenciador de material de criptografia (CMM) monta o material criptográfico usado para criptografar e descriptografar dados. O material criptográfico inclui texto não criptografado e chaves de dados criptografadas e uma chave de assinatura de mensagem opcional. Você nunca interage diretamente com o CMM. Os métodos de criptografia e descriptografia o processam para você.

Você pode usar o CMM padrão ou o [CMM de armazenamento em cache](#) fornecido pelo AWS Encryption SDK ou escrever um CMM personalizado. E é possível especificar explicitamente um

CMM, mas isso não é obrigatório. Quando você especifica um token de autenticação ou um provedor de chaves mestras, o AWS Encryption SDK cria um CMM padrão para você. O CMM padrão obtém o material de criptografia ou de descryptografia do token de autenticação ou do provedor de chave mestra que você especificar. Isso pode envolver uma chamada a um serviço criptográfico, como o [AWS Key Management Service](#) (AWS KMS).

Como o CMM atua como uma ligação entre o AWS Encryption SDK e um token de autenticação (ou um provedor de chaves mestras), ele é um ponto ideal para personalização e extensão, como suporte para a aplicação de políticas e o armazenamento em cache. O AWS Encryption SDK fornece um CMM de armazenamento em cache para oferecer suporte ao [armazenamento em cache de chaves de dados](#).

Criptografia simétrica e assimétrica

A criptografia simétrica usa a mesma chave para criptografar e descryptografar dados.

A criptografia assimétrica usa um par de chaves de dados matematicamente relacionado. Uma chave no par criptografa os dados; somente a outra chave no par pode descryptografá-los. Para obter detalhes, consulte [Algoritmos criptográficos](#) no Guia de ferramentas e serviços criptográficos da AWS.

O AWS Encryption SDK usa a [criptografia envelopada](#). Ele criptografa os dados com uma chave de dados simétrica. Ele criptografa a chave de dados simétrica com uma ou mais chaves de encapsulamento simétricas ou assimétricas. Ele retorna uma [mensagem criptografada](#) que inclui os dados criptografados e pelo menos uma cópia criptografada da chave de dados.

Criptografar dados (criptografia simétrica)

Para criptografar os dados, o AWS Encryption SDK usa uma [chave de dados](#) simétrica e um [pacote de algoritmos](#) que inclui um algoritmo de criptografia simétrica. Para descryptografar os dados, o AWS Encryption SDK usa a mesma chave de dados e o mesmo pacote de algoritmos.

Criptografar chave de dados (criptografia simétrica ou assimétrica)

O [token de autenticação](#) ou o [provedor de chave mestra](#) que você fornece para uma operação de criptografia e descryptografia determina como a chave de dados simétrica será criptografada e descryptografada. Você pode escolher um token de autenticação ou provedor de chave mestra que use criptografia simétrica, como um token de autenticação do AWS KMS, ou um que use criptografia assimétrica, como um token de autenticação bruto do RSA ou JceMasterKey.

Confirmação de chave

O AWS Encryption SDK oferece suporte à confirmação de chaves (às vezes conhecido como robustez), uma propriedade de segurança que garante que cada texto cifrado possa ser descryptografado somente em um único texto simples. Para fazer isso, a confirmação de chave garante que somente a chave de dados que criptografou sua mensagem seja usada para descryptografá-la. Criptografar e descryptografar com o confirmação de chave é uma [prática recomendada do AWS Encryption SDK](#).

A maioria das cifras simétricas modernas (incluindo AES) criptografa um texto simples com uma única chave secreta, como a [chave de dados exclusiva](#) que o AWS Encryption SDK usa para criptografar cada mensagem de texto simples. Descryptografar esses dados com a mesma chave de dados retorna um texto sem formatação idêntico ao original. A decodificação com uma chave diferente geralmente falhará. No entanto, é possível decifrar um texto cifrado com duas chaves diferentes. Em casos raros, é possível encontrar uma chave que possa descryptografar alguns bytes do o texto cifrado em um texto simples diferente, mas ainda inteligível.

O AWS Encryption SDK sempre criptografa cada mensagem de texto simples em uma chave de dados exclusiva. Ele pode criptografar essa chave de dados em várias chaves de encapsulamento, mas as chaves de encapsulamento (ou chaves mestras) sempre criptografam a mesma chave de dados. No entanto, uma [mensagem criptografada](#) sofisticada e criada manualmente pode, na verdade, conter chaves de dados diferentes, cada uma criptografada por uma chave de encapsulamento diferente. Por exemplo, se um usuário descryptografar a mensagem criptografada, ela retornará 0x0 (falso), enquanto outro usuário descryptografando a mesma mensagem criptografada obterá 0x1 (verdadeiro).

Para evitar esse cenário, o AWS Encryption SDK é compatível com confirmação de chave ao criptografar e descryptografar. Quando o AWS Encryption SDK criptografa uma mensagem com confirmação de chave, ele vincula criptograficamente a chave de dados exclusiva que produziu o texto cifrado ao string de compromisso de chave, um identificador de chave de dados não secreto. Em seguida, ele armazena o string de compromisso chave nos metadados da mensagem criptografada. Ao descryptografar uma mensagem com confirmação de chave, o AWS Encryption SDK verifica se a chave de dados é a única chave para essa mensagem criptografada. Se a verificação da chave de dados falhar, a operação de descryptografia falhará.

O suporte para confirmação de chaves foi apresentado na versão 1.7. x, que pode descryptografar mensagens com confirmação de chave, mas não pode criptografar com confirmação de chave. Você pode usar essa versão para implantar totalmente a capacidade de descryptografar texto cifrado

com confirmação de chave. A versão 2.0.x inclui suporte total para os compromissos de chave. Por padrão, ela criptografa e descriptografa somente com confirmação de chave. Essa é uma configuração ideal para aplicações que não precisam descriptografar texto cifrado criptografado por versões anteriores do AWS Encryption SDK.

Embora criptografar e descriptografar com confirmação de chave seja uma prática recomendada, deixamos que você decida quando ela será usada e ajustamos o ritmo em que você a adota. A partir da versão 1.7.x, o AWS Encryption SDK suporta uma [política de compromisso](#) que define o [pacote de algoritmos padrão](#) e limita os pacotes de algoritmos que podem ser usados. Essa política determina se seus dados são criptografados e descriptografados com confirmação de chave.

A confirmação de chave resulta em uma [mensagem criptografada um pouco maior \(+ 30 bytes\)](#) e que leva mais tempo para ser processada. Se sua aplicação for muito sensível ao tamanho ou à performance, você poderá optar por não aceitar a confirmação de chave. Mas faça isso somente se for necessário.

Para obter mais informações sobre a migração para as versões 1.7.x e 2.0.x, incluindo seus atributos de confirmação de chave, consulte [Como migrar seu AWS Encryption SDK](#). Para obter informações técnicas sobre confirmação de chave, consulte [the section called “Referência de algoritmos”](#) e [the section called “Referência do formato de mensagens”](#).

Política de compromisso

Uma política de compromisso é uma definição de configuração que determina se a aplicação criptografa e descriptografa com [confirmação de chave](#). Criptografar e descriptografar com o confirmação de chave é uma [prática recomendada do AWS Encryption SDK](#).

A política de compromisso tem três valores.

Note

Talvez seja necessário rolar horizontalmente ou verticalmente para ver a tabela inteira.

Valores da política de compromisso

Valor	Criptografa com confirmação de chave	Criptografa sem confirmação de chave	Descriptografa com confirmação de chave	Descriptografa sem confirmação de chave
ForbidEncryptAllowDecrypt				
RequireEncryptAllowDecrypt				
RequireEncryptRequireDecrypt				

A configuração da política de compromisso foi introduzida na versão 1.7.x. do AWS Encryption SDK. Ela é válida em todas as [linguagens de programação](#) suportadas.

- O `ForbidEncryptAllowDecrypt` descriptografa com ou sem confirmação de chave, mas não criptografa com confirmação de chave. Este é o único valor válido para a política de compromisso na versão 1.7. x, e ele é usado para todas as operações de criptografia e descriptografia. Ele foi projetado para preparar todos os hosts que executam sua aplicação para descriptografar com confirmação de chave antes mesmo de encontrarem um texto cifrado criptografado com confirmação de chave.
- O `RequireEncryptAllowDecrypt` sempre criptografa com confirmação de chave. Ele pode descriptografar textos cifrados criptografados com ou sem confirmação de chave. Este valor, introduzido na versão 2.0.x, permite que você comece a criptografar com confirmação de chave, mas ainda descriptografe textos cifrados herdados sem confirmação de chave.
- `RequireEncryptRequireDecrypt`: criptografa e descriptografa somente com confirmação de chave. Esse valor é o padrão para a versão 2.0x. Use-o quando tiver certeza de que todos os seus textos cifrados estão criptografados com a confirmação de chave.

A configuração da política de compromisso determina quais pacotes de algoritmos você pode usar. A partir da versão 1.7.x, o AWS Encryption SDK suporta [pacotes de algoritmos](#) para confirmação de chave; com e sem assinatura. Se você especificar um pacote de algoritmos que entre em conflito com sua política de compromisso, o AWS Encryption SDK retornará um erro.

Para obter ajuda para definir sua política de compromisso, consulte [Como definir sua política de compromisso](#).

Assinaturas digitais

Para garantir a integridade de uma mensagem digital à medida que ela passa entre os sistemas, você pode aplicar uma assinatura digital à mensagem. As assinaturas digitais são sempre assimétricas. Use a chave privada para criar a assinatura e anexá-la à mensagem original. O destinatário usa uma chave pública para verificar se a mensagem não foi modificada desde que você a assinou.

O AWS Encryption SDK criptografa seus dados usando um algoritmo de criptografia autenticado, o AES-GCM, e o processo de descryptografia verifica a integridade e a autenticidade de uma mensagem criptografada sem usar uma assinatura digital. Mas como o AES-GCM usa chaves simétricas, qualquer pessoa que possa descryptografar a chave de dados usada para descryptografar o texto cifrado também pode criar manualmente um novo texto cifrado, causando uma possível preocupação de segurança. Por exemplo, se você usar uma chave do AWS KMS como chave de encapsulamento, isso significa que é possível para um usuário com permissões do KMS Decrypt criar textos cifrados criptografados sem chamar o KMS Encrypt.

Para evitar esse problema, o AWS Encryption SDK suporta a adição de uma assinatura do Algoritmo de Assinatura Digital de Curva Elíptica (ECDSA) ao final das mensagens criptografadas. Quando um pacote de algoritmos de assinatura é usado, o AWS Encryption SDK gera uma chave privada temporária e um par de chaves públicas para cada mensagem criptografada. O AWS Encryption SDK armazena a chave pública no contexto de criptografia da chave de dados e descarta a chave privada, e ninguém pode criar outra assinatura que verifique com a chave pública. Como o algoritmo vincula a chave pública à chave de dados criptografada como dados autenticados adicionais no cabeçalho da mensagem, um usuário que só pode descryptografar mensagens não pode alterar a chave pública.

A verificação de assinatura adiciona um custo significativo de performance à descryptografia. Se os usuários que criptografam dados e os usuários que decifram os dados forem igualmente confiáveis, considere usar um pacote de algoritmos que não inclua assinatura.

Como o AWS Encryption SDK funciona

Os fluxos de trabalho desta seção explicam como o AWS Encryption SDK criptografa os dados e descriptografa [mensagens criptografadas](#). Esses fluxos de trabalho descrevem o processo básico usando os atributos padrão. Para obter detalhes sobre como definir e usar componentes personalizados, consulte o GitHub repositório de cada [implementação de linguagem](#) compatível.

O AWS Encryption SDK usa a criptografia envelopada para proteger seus dados. Cada mensagem é criptografada em uma chave de dados exclusiva. Em seguida, a chave de dados é criptografada pelas chaves de encapsulamento que você especificar. Para descriptografar a mensagem criptografada, o AWS Encryption SDK usa as chaves de encapsulamento que você especifica para descriptografar pelo menos uma chave de dados criptografada. Em seguida, ele pode descriptografar o texto cifrado e retornar uma mensagem de texto simples.

Precisa de ajuda com a terminologia que usamos no AWS Encryption SDK? Consulte [the section called “Conceitos”](#).

Como o AWS Encryption SDK criptografa os dados

O AWS Encryption SDK fornece métodos que criptografam strings, matrizes de bytes e fluxos de bytes. Para obter exemplos de código, consulte o tópico Exemplos em cada seção de [Linguagens de programação](#).

1. Crie um [token de autenticação](#) (ou um [provedor de chave mestra](#)) que especifique as chaves de agrupamento que protegem seus dados.
2. Transmita o chaveiro e os dados do texto simples para um método de criptografia. Recomendamos transmitir um [contexto de criptografia](#) opcional, não secreto.
3. O método de criptografia solicita materiais de criptografia ao token de autenticação. O chaveiro retorna chaves de criptografia de dados exclusivas para a mensagem: uma chave de dados em texto simples e uma cópia dessa chave de dados criptografada por cada uma das chaves de encapsulamento especificadas.
4. O método de criptografia usa a chave de dados de texto não criptografado para criptografar os dados e, em seguida, descarta a chave de dados de texto não criptografado. Se você fornecer um contexto de criptografia (uma [prática recomendada](#) do AWS Encryption SDK), o método de criptografia também vinculará de forma criptográfica o contexto de criptografia aos dados criptografados.

5. O método de criptografia retorna uma [mensagem criptografada](#) que contém os dados criptografados, as chaves de dados criptografadas e outros metadados, incluindo o contexto de criptografia, se você o usou.

Como o AWS Encryption SDK descriptografa e criptografa uma mensagem

O AWS Encryption SDK fornece métodos que decifram a [mensagem criptografada](#) e retornam texto simples. Para obter exemplos de código, consulte o tópico Exemplos em cada seção de [Linguagens de programação](#).

O [token de autenticação](#) (ou o [provedor de chave mestra](#)) que descriptografará a mensagem criptografada deve ser compatível com aquele usado para criptografar a mensagem. Uma das chaves de encapsulamento dele deve ser capaz descriptografar uma chave de dados criptografada na mensagem criptografada. Para obter informações sobre compatibilidade com tokens de autenticação e provedores de chaves mestra, consulte [the section called “Compatibilidade dos tokens de autenticação”](#).

1. Crie um token de autenticação ou provedor de chave mestra com chaves de encapsulamento que possam descriptografar seus dados. É possível usar o mesmo token de autenticação fornecido para o método de criptografia ou um token diferente.
2. Transmita a [mensagem criptografada](#) e o token de autenticação para um método de descriptografia.
3. O método de descriptografia solicita que o token de autenticação ou o provedor de chave mestra descriptografe uma das chaves de dados criptografadas na mensagem criptografada. Ele passa informações da mensagem criptografada, incluindo as chaves de dados criptografadas.
4. O token de autenticação usa suas chaves de empacotamento para descriptografar uma das chaves de dados criptografadas. A resposta incluirá a chave de dados em texto simples, se for bem-sucedida. Caso nenhuma das chaves de encapsulamento especificadas pelo token de autenticação ou provedor da chave mestra possa descriptografar uma chave de dados criptografada, a chamada de descriptografia falhará.
5. O método de descriptografia usa a chave de dados de texto simples para descriptografar os dados, descarta a chave de dados de texto simples e retorna os dados de texto simples.

Pacotes de algoritmos compatíveis no AWS Encryption SDK

Um pacote de algoritmos é uma coleção de algoritmos criptográficos e de valores relacionados. Os sistemas de criptografia usam a implementação do algoritmo para gerar a mensagem de texto cifrado.

O pacote de algoritmos do AWS Encryption SDK usa o algoritmo Advanced Encryption Standard (AES - padrão avançado de criptografia) no Galois/Counter Mode (GCM), conhecido como AES-GCM, para criptografar dados brutos. O AWS Encryption SDK é compatível com chaves de criptografia de 256, 192 e 128 bits. O tamanho do vetor de inicialização (IV) é sempre 12 bytes. O tamanho da tag de autenticação é sempre 16 bytes.

Por padrão, o AWS Encryption SDK usa um conjunto de algoritmos com AES-GCM com uma função de derivação de chave de extração e expansão baseada em HMAC ([HKDF](#)), assinatura e uma chave de criptografia de 256 bits. Se a [política de compromisso](#) exigir [confirmação de chave](#), o AWS Encryption SDK seleciona um pacote de algoritmos que também oferece suporte ao [confirmação de chave](#); caso contrário, seleciona um pacote de algoritmos com derivação e assinatura de chaves, mas não com [confirmação de chave](#).

Recomendado: AES-GCM com derivação de chave, assinatura e confirmação de chave

O AWS Encryption SDK recomenda um pacote de algoritmos que deriva uma chave de criptografia AES-GCM fornecendo uma chave de criptografia de dados de 256 bits para a função de derivação de chave de extração e expansão (HKDF) baseada em HMAC. O AWS Encryption SDK adiciona uma assinatura do Elliptic Curve Digital Signature Algorithm (ECDSA – Algoritmo de assinatura digital de curva elíptica). Para oferecer suporte ao [comprometimento da chave](#), esse pacote de algoritmos também deriva uma sequência de caracteres de comprometimento da chave, um identificador de chave de dados não secreto, que é armazenado nos metadados da mensagem criptografada. Essa sequência de comprometimento da chave também é derivada por meio do HKDF usando um procedimento semelhante à derivação da chave de criptografia de dados.

Pacote de algoritmos do AWS Encryption SDK

Algoritmo de criptografia	Tamanho da chave de criptografia de dados (em bits)	Algoritmo de derivação de chave	Algoritmo de assinatura	Confirmação de chave
AES-GCM	256	HKDF com SHA-384	ECDSA com P-384 e SHA-384	HKDF com SHA-512

A HKDF ajuda a evitar a reutilização acidental de uma chave de criptografia de dados e reduz o risco de sobreuso de chaves de dados.

Para assinatura, esse pacote de algoritmos usa ECDSA com um algoritmo de função hash criptográfica (SHA-384). O ECDSA é usado por padrão, mesmo quando ele não é especificado pela política da chave mestra subjacente. A [assinatura da mensagem](#) verifica se o remetente foi autorizado a criptografar mensagens e fornece o não repúdio. Isso é especialmente útil quando a política de autorização de uma chave mestra permite que um conjunto de usuários criptografe dados e outro conjunto de usuários descriptografe os dados.

Conjuntos de algoritmos com confirmação de chave garantem que cada texto cifrado seja descriptografado em apenas um texto simples. Eles fazem isso validando a chave de dados usada como entrada para o algoritmo de criptografia. Ao criptografar, esses conjuntos de algoritmos derivam um HMAC de comprometimento fundamental. Antes de descriptografar, eles validam que a chave de dados corresponde à sequência de conformação da chave. Caso contrário, a chamada de descriptografia falhará.

Outros pacotes de algoritmos compatíveis

O AWS Encryption SDK oferece suporte aos seguintes pacotes de algoritmos alternativos para fins de compatibilidade com versões anteriores. Em geral, não recomendamos esses pacotes de algoritmos. No entanto, reconhecemos que a assinatura pode prejudicar significativamente a performance, por isso oferecemos um pacote de confirmação de chaves com derivação de chaves para esses casos. Para aplicações que precisam fazer concessões de performance mais significativas, continuamos oferecendo pacotes que não possuem assinatura, confirmação de chaves e derivação de chaves.

AES-GCM sem confirmação de chave

Os conjuntos de algoritmos sem confirmação de chave não validam a chave de dados antes da criptografia. Como resultado, esses conjuntos de algoritmos podem criptografar um único texto cifrado em diferentes mensagens de texto simples. No entanto, como os pacotes de algoritmos com confirmação de chave produzem uma [mensagem criptografada um pouco maior \(+30 bytes\)](#) e demoram mais para serem processados, eles podem não ser a melhor opção para cada aplicação.

O AWS Encryption SDK oferece suporte a um pacote de algoritmos com derivação de chave, confirmação de chave e assinatura, e um com derivação de chave e confirmação de chave, mas sem assinatura. Não recomendamos usar um pacote de algoritmos sem confirmação de chave. Se necessário, recomendamos um pacote de algoritmos com derivação e confirmação de chaves, mas sem assinatura. No entanto, se o perfil de performance da aplicação for compatível com o uso de um pacote de algoritmos, usar um pacote de algoritmos com confirmação de chaves, derivação de chaves e assinatura é uma prática recomendada.

AES-GCM sem assinatura

Os conjuntos de algoritmos sem assinatura carecem da assinatura ECDSA, que fornece autenticidade e não repúdio. Use esse conjunto apenas quando os usuários que criptografam dados e os que os descriptografam são igualmente confiáveis.

Ao usar um pacote de algoritmos sem assinatura, recomendamos escolher um com derivação e confirmação de chave.

AES-GCM sem derivação de chaves

Pacotes de algoritmos sem derivação usam a criptografia de dados como a chave de criptografia do AES-GCM, em vez de usar uma função de derivação de chaves para derivar uma chave exclusiva. Desencorajamos o uso desse pacote para gerar texto cifrado, mas o AWS Encryption SDK oferece suporte a ele por motivos de compatibilidade.

Para obter mais informações sobre como esses pacotes são representados e usados na biblioteca, consulte [the section called “Referência de algoritmos”](#).

Uso da AWS Encryption SDK com o AWS KMS

Para usar o AWS Encryption SDK, você precisa configurar [tokens de autenticação](#) ou [provedores de chave mestra](#) com chaves de encapsulamento. Se você não tiver uma infraestrutura de chaves, recomendamos usar o [AWS Key Management Service \(AWS KMS\)](#). Muitos dos exemplos de código no AWS Encryption SDK exigem uma [AWS KMS key](#).

Para interagir com o AWS KMS, o AWS Encryption SDK exige o SDK da AWS para a linguagem de programação que você preferir. A biblioteca de cliente do AWS Encryption SDK funciona com os SDKs da AWS para oferecer suporte às chaves mestres armazenadas no AWS KMS.

Como preparar-se para usar o AWS Encryption SDK com o AWS KMS

1. Crie Conta da AWS. Para saber como, consulte [Como faço para criar e ativar uma nova conta do Amazon Web Services?](#) no Centro de conhecimento da AWS.
2. Crie uma chave de criptografia simétrica AWS KMS key. Para obter ajuda, consulte [Criação de chaves](#) no Guia do desenvolvedor AWS Key Management Service.

Tip

Para usar a AWS KMS key programaticamente, você precisará do ID de chave ou do nome de recurso da Amazon (ARN) da AWS KMS key. Para ajudar a descobrir o ID ou o ARN de uma AWS KMS key, consulte [Descobrir o ID de chave e o ARN](#) no Guia do desenvolvedor do AWS Key Management Service.

3. Gere um ID de chave de acesso e uma chave de acesso de segurança. Você pode usar o ID da chave de acesso e a chave de acesso secreta para um usuário do IAM ou o AWS Security Token Service para criar uma nova sessão com credenciais de segurança temporárias que incluam um ID de chave de acesso, chave de acesso secreta e token de sessão. Como prática recomendada de segurança, use credenciais temporárias em vez das credenciais de longo prazo associadas às contas de usuário do IAM ou contas de usuário (raiz) da AWS.

Para criar um usuário do IAM com uma chave de acesso, consulte [Criação de usuários do IAM](#) no Guia do usuário do IAM.

Para gerar mais informações sobre credenciais de segurança temporárias, consulte [Solicitação de credenciais de segurança temporárias](#) no Guia do usuário do IAM.

4. Defina suas credenciais da AWS usando as instruções no [AWS SDK for Java](#)[AWS SDK for JavaScript](#) [AWS SDK for Python \(Boto\)](#) ou [AWS SDK for C++](#) (para C) e o ID da chave de acesso e a chave de acesso secreta gerados na etapa 3. Se você gerou credenciais temporárias, também precisará especificar o token de sessão.

Este procedimento permite que os SDKs da AWS assinem solicitações à AWS para você. Os códigos de exemplo no AWS Encryption SDK que interagem com o AWS KMS pressupõem que você tenha concluído esta etapa.

5. Faça download e instale o AWS Encryption SDK. Para saber como, consulte as instruções de instalação da [linguagem de programação](#) que você deseja usar.

Melhores práticas para a AWS Encryption SDK

O AWS Encryption SDK foi projetado para facilitar a proteção de seus dados usando os padrões e as práticas recomendadas do setor. Embora muitas práticas recomendadas tenham sido selecionadas para você nos valores padrão, algumas delas são opcionais, mas recomendadas sempre que seja prático.

Use a versão mais recente

Ao começar a usar o AWS Encryption SDK, use a versão mais recente oferecida na [linguagem de programação](#) da sua preferência. Se estiver usando o AWS Encryption SDK, atualize para cada versão mais recente o mais rápido possível. Isso assegura que você esteja usando a configuração recomendada e aproveitando as novas propriedades de segurança para proteger seus dados. Para obter detalhes sobre as versões compatíveis, incluindo orientações para migração e implantação, consulte [Suporte e manutenção](#) e [Versões do AWS Encryption SDK](#).

Se uma nova versão descontinuar elementos em seu código, substitua-os assim que possível. Os avisos de descontinuação e os comentários de código geralmente recomendam uma boa alternativa.

Para tornar as atualizações significativas mais fáceis e menos propensas a erros, ocasionalmente fornecemos uma versão temporária ou transitória. Use essas versões e a documentação que as acompanha para garantir que você possa atualizar a aplicação sem interromper seu fluxo de trabalho de produção.

Use os valores padrão

O AWS Encryption SDK projeta as práticas recomendadas em seus valores padrão. Sempre que possível, use-os. Para casos em que aplicar o padrão seja pouco prático, fornecemos alternativas, como pacotes de algoritmos sem assinatura. Também oferecemos oportunidades de personalização para usuários avançados, como tokens de autenticação, fornecedores de chaves mestras e gerenciadores de material criptográfico (CMMs) personalizados. Use essas alternativas avançadas com cuidado e faça com que um engenheiro de segurança verifique suas escolhas sempre que possível.

Usar um contexto de criptografia

Para melhorar a segurança de suas operações de criptografia, inclua um [contexto de criptografia](#) com um valor significativo em todas as solicitações para criptografar dados. O uso de um contexto de criptografia é opcional, mas é uma melhor prática de criptografia que recomendamos.

Ele oferece dados autenticados adicionais (AAD) para criptografia autenticada no AWS Encryption SDK. Embora não seja secreto, o contexto de criptografia pode ajudar você a [proteger a integridade e a autenticidade](#) de seus dados criptografados.

No AWS Encryption SDK, você especifica um contexto de criptografia somente ao criptografar. Ao descriptografar, o AWS Encryption SDK usa o contexto de criptografia no cabeçalho da mensagem criptografada retornada pelo AWS Encryption SDK. Antes da aplicação retornar os dados de texto simples, verifique se o contexto de criptografia usado para criptografar a mensagem está incluído no contexto de criptografia usado ao descriptografar a mensagem. Para obter detalhes, consulte os exemplos na sua linguagem de programação.

Quando você usa a interface de linha de comando, o AWS Encryption SDK verifica o contexto de criptografia para você.

Proteja suas chaves de encapsulamento

O AWS Encryption SDK gera uma chave de dados exclusiva para criptografar cada mensagem de texto simples. Em seguida, ele criptografa a chave de dados com as chaves de encapsulamento fornecidas por você. Se suas chaves de encapsulamento forem perdidas ou excluídas, seus dados criptografados serão irre recuperáveis. Se suas chaves não estiverem protegidas, seus dados podem ficar vulneráveis.

Use chaves de encapsulamento protegidas por uma infraestrutura de chave segura, como o [AWS Key Management Service](#) (AWS KMS). Ao usar chaves AES ou RSA brutas, utilize uma fonte de randomização e armazenamento durável que atenda aos seus requisitos de segurança. A geração e o armazenamento de chaves de encapsulamento em um módulo de segurança de hardware (HSM) ou em um serviço que fornece HSMs, como o AWS CloudHSM, é uma prática recomendada.

Use os mecanismos de autorização da sua infraestrutura de chaves para limitar o acesso às chaves de encapsulamento somente aos usuários que exigem elas. Implemente princípios de práticas recomendadas, como privilégio mínimo. Ao usar as AWS KMS keys, use políticas de chave e políticas do IAM que implementem [os princípios das práticas recomendadas](#).

Especifique suas chaves de encapsulamento

A [especificação das chaves de encapsulamento](#) de forma explícita ao descriptografar e criptografar é sempre uma prática recomendada. Ao fazer isso, o AWS Encryption SDK usa somente as chaves especificadas por você. Essa prática garante que você use somente as chaves de criptografia que pretende utilizar. Para chaves de encapsulamento do AWS KMS,

ela também melhora a performance ao impedir que você inadvertidamente use chaves em uma Conta da AWS ou região diferente ou tente descriptografar com chaves que não tem permissão para usar.

Ao criptografar, os tokens de autenticação e os provedores de chaves mestras fornecidos pelo AWS Encryption SDK exigem que você especifique as chaves de encapsulamento. Eles utilizam somente as chaves de encapsulamento que você especificar, nenhuma outra. Também é necessário especificar chaves de encapsulamento ao criptografar e descriptografar com tokens de autenticação do AES brutos, tokens de autenticação do RSA brutos e JCEmasterKeys.

Entretanto, ao descriptografar com tokens de autenticação e provedores de chaves mestras do AWS KMS, você não precisa especificar chaves de encapsulamento. O AWS Encryption SDK pode obter o identificador da chave a partir dos metadados da chave de dados criptografada. No entanto, recomendamos que você especifique as chaves de encapsulamento, já que esta é uma prática recomendada.

Para apoiar essa prática recomendada ao trabalhar com chaves de encapsulamento do AWS KMS, recomendamos o seguinte:

- Use tokens de autenticação do AWS KMS que especifiquem as chaves de encapsulamento. Ao criptografar e descriptografar, esses tokens de autenticação usam somente as chaves de encapsulamento especificadas por você.
- Ao usar chaves mestras e provedores de chaves mestras do AWS KMS, use os construtores de modo estrito introduzidos na [versão 1.7.x](#) do AWS Encryption SDK. Eles criam provedores que criptografam e descriptografam somente com as chaves de encapsulamento que você especificar. Os construtores de provedores de chave mestra que sempre descriptografam com qualquer chave de encapsulamento foram descontinuados na versão 1.7.x e excluídos na versão 2.0.x.

Quando a especificação de chaves de encapsulamento do AWS KMS para descriptografar for impraticável, você pode usar provedores de descoberta. O AWS Encryption SDK em C e JavaScript é compatível [tokens de autenticação de descoberta do AWS KMS](#). Os provedores de chaves mestras com um modo de descoberta estão disponíveis para Java e Python nas versões 1.7.x e posteriores. Esses provedores de descoberta, usados somente para descriptografar com chaves de encapsulamento do AWS KMS, direcionam explicitamente o AWS Encryption SDK para usar qualquer chave de encapsulamento que tenha criptografado uma chave de dados.

Se você precisar usar um provedor de descoberta, use os atributos de filtro de descoberta para limitar as chaves de encapsulamento que eles usam. Por exemplo, o [token de autenticação](#)

[de descoberta regional do AWS KMS](#) usa somente as chaves de encapsulamento em uma determinada Região da AWS. Você também pode configurar tokens de autenticação do AWS KMS e [provedores de chaves mestras do AWS KMS](#) para usar somente as [chaves de encapsulamento](#) em Contas da AWS específicas. Além disso, como sempre, use políticas de chave e políticas do IAM para controlar o acesso às suas chaves de encapsulamento do AWS KMS.

Use assinaturas digitais

É uma prática recomendada usar um pacote de algoritmos com assinatura. [Assinaturas digitais](#) confirmam que o remetente da mensagem foi autorizado a enviá-la e protegem a integridade da mensagem. Todas as versões do AWS Encryption SDK usam pacotes de algoritmos com assinatura por padrão.

Se seus requisitos de segurança não incluírem assinaturas digitais, você pode selecionar um pacote de algoritmos sem assinaturas digitais. No entanto, recomendamos o uso de assinaturas digitais, especialmente quando um grupo de usuários criptografa dados e um grupo diferente de usuários descriptografa esses dados.

Use o confirmação de chave

É uma prática recomendada usar o atributo de segurança de confirmação de chave. Ao verificar a identidade da [chave de dados](#) exclusiva que criptografou seus dados, a [confirmação de chave](#) impede que você descriptografe qualquer texto cifrado que possa resultar em mais de uma mensagem de texto simples.

[O AWS Encryption SDK fornece suporte completo para criptografia e descriptografia com confirmação de chave a partir da versão 2.0.x..](#) Por padrão, todas as suas mensagens são criptografadas e descriptografadas com confirmação de chave. A [versão 1.7.x](#) do AWS Encryption SDK pode descriptografar textos cifrados com confirmação de chave. Ela foi projetada para ajudar os usuários de versões anteriores a implantar a versão 2.0.x com sucesso.

O suporte para confirmação de chaves inclui [novos pacotes de algoritmos](#) e um [novo formato de mensagem](#) que produz um texto cifrado apenas 30 bytes maior do que um texto cifrado sem confirmação de chave. O design minimiza seu impacto na performance para que a maioria dos usuários possa aproveitar os benefícios da confirmação de chave. Se sua aplicação for muito sensível ao tamanho e à performance, você poderá decidir usar a configuração da [política de compromisso](#) para desabilitar o compromisso da chave ou permitir que o AWS Encryption SDK descriptografe mensagens sem compromisso, mas somente faça isso se realmente for necessário.

Limite o número de chaves de dados criptografadas

É uma prática recomendada [limitar o número de chaves de dados criptografadas](#) nas mensagens que você descriptografa, especialmente mensagens de fontes não confiáveis. Descriptografar uma mensagem com várias chaves de dados criptografadas que você não pode descriptografar pode gerar atrasos prolongados, aumentar despesas, limitar a capacidade da aplicação e de outros que compartilham sua conta e potencialmente esgotar sua infraestrutura de chaves. Uma mensagem criptografada pode conter até 65.535 ($2^{16} - 1$) chaves de dados criptografadas. Para obter mais detalhes, consulte [Limitar as chaves de dados criptografadas](#).

Para obter mais informações sobre os atributos de segurança do AWS Encryption SDK subjacentes a essas práticas recomendadas, consulte [Criptografia do lado do cliente: aprimorada: IDs de chave explícitos e confirmação de chave](#) no Blog de segurança da AWS.

Configurar o AWS Encryption SDK

O AWS Encryption SDK foi projetado para ser fácil de usar. Embora AWS Encryption SDK tenha várias opções de configuração, os valores padrão são cuidadosamente escolhidos para serem práticos e seguros para a maioria das aplicações. No entanto, talvez seja necessário ajustar sua configuração para melhorar a performance ou incluir um atributo personalizado em seu design.

Ao configurar sua implementação, analise as [prática recomendadas do AWS Encryption SDK](#) e as implemente o máximo possível.

Tópicos

- [Seleção de uma linguagem de programação](#)
- [Seleção de chaves de encapsulamento](#)
- [Usar AWS KMS keys multirregionais](#)
- [Escolher um pacote de algoritmo](#)
- [Limitar as chaves de dados criptografadas](#)
- [Criação de um filtro de descoberta](#)
- [Como definir uma política de compromisso](#)
- [Trabalhar com streaming de dados](#)
- [Armazenamento em cache de chaves de dados](#)

Seleção de uma linguagem de programação

O AWS Encryption SDK está disponível em várias [linguagens de programação](#). As implementações de linguagem são projetadas para serem totalmente interoperáveis e oferecer os mesmos atributos, embora possam ser implementadas de maneiras diferentes. Normalmente, você usa a biblioteca compatível com sua aplicação. No entanto, pode selecionar uma linguagem de programação para uma implementação específica. Por exemplo, se você preferir trabalhar com [tokens de autenticação](#), pode escolher o AWS Encryption SDK for C ou o AWS Encryption SDK para JavaScript.

Seleção de chaves de encapsulamento

O AWS Encryption SDK gera uma chave de dados simétrica exclusiva para criptografar cada mensagem. A menos que você esteja usando o [armazenamento em cache de chaves de dados](#),

you do not need to configure, manage or use the data keys. The AWS Encryption SDK does this for you.

However, you must select one or more wrapping keys to encrypt each data key. The AWS Encryption SDK is compatible with symmetric AES keys and asymmetric RSA keys in different sizes. It is also compatible with symmetric AWS KMS keys from the [AWS Key Management Service](#) (AWS KMS). You are responsible for the security and durability of your wrapping keys, so we recommend that you use a key encryption module or a key service, such as AWS KMS.

To specify your wrapping keys for encryption and decryption, use an authentication token (C and JavaScript) or a master key provider (Java, Python, CLI of AWS CLI). It is possible to specify one wrapping key or several keys of the same type or of different types. If you use several wrapping keys to encrypt a data key, each wrapping key will encrypt a copy of the data key. The encrypted data keys (one per wrapping key) are stored with the encrypted data in the encrypted message that the AWS Encryption SDK returns. To decrypt the data, the AWS Encryption SDK must first use one of your wrapping keys to decrypt a data key.

To specify an AWS KMS key in an authentication token or master key provider, use a key identifier compatible with AWS KMS. For details about key identifiers for an AWS KMS key, consult [Key Identifiers](#) in the AWS Key Management Service Developer Guide.

- When encrypting with the AWS Encryption SDK for Java, the AWS Encryption SDK for JavaScript, the AWS Encryption SDK for Python, or with the AWS CLI, you can use any valid key identifier (key ID, key ARN, key alias name, or key alias ARN) for a KMS key. When encrypting with the AWS Encryption SDK for C, you can only use a key ID or a key ARN.

If you specify an alias name or an alias ARN for a KMS key when encrypting, the AWS Encryption SDK will save the ARN of the key currently associated with that alias; it will not save the alias name. Changes to the alias do not affect the KMS key used to decrypt your data keys.

- Ao descriptografar no modo estrito (onde você especifica chaves de encapsulamento específicas), você deve usar um ARN de chave para identificar as AWS KMS keys. Esse requisito aplica-se a todas as implementações de linguagem do AWS Encryption SDK.

Quando você criptografa com um token de autenticação do AWS KMS, o AWS Encryption SDK armazena o ARN de chave da AWS KMS key nos metadados da chave de dados criptografada. Ao descriptografar no modo estrito, o AWS Encryption SDK verifica se o mesmo ARN de chave aparece no token de autenticação (ou no provedor da chave mestra) antes de tentar usar a chave de encapsulamento para descriptografar a chave de dados criptografada. Se você usar um identificador de chave diferente, o AWS Encryption SDK não reconhecerá nem usará o AWS KMS key, mesmo que os identificadores se refiram à mesma chave.

Para especificar uma [chave AES bruta](#) ou um [par de chaves RSA brutas](#) como chave de agrupamento em um token de autenticação, você deve especificar um namespace e um nome. Em um provedor de chave mestra, o `Provider ID` é o equivalente do namespace e o `Key ID` é o equivalente do nome. Ao descriptografar, você deve usar exatamente o mesmo namespace e nome para cada chave de encapsulamento bruta que você usou ao criptografar. Se você usar um namespace ou nome diferente, o AWS Encryption SDK não reconhecerá nem usará a chave de encapsulamento, mesmo que o material da chave seja o mesmo.

Usar AWS KMS keys multirregionais

Você pode usar chaves multirregionais AWS Key Management Service (AWS KMS) como chaves de encapsulamento no AWS Encryption SDK. Se você criptografar com uma chave multirregional em uma Região da AWS, poderá descriptografar usando uma chave multirregional relacionada em outra Região da AWS. O suporte para chaves multirregionais foi introduzido na versão 2.3.x do AWS Encryption SDK e na versão 3.0.x da CLI de criptografia da AWS.

As chaves do AWS KMS multirregionais são um conjunto de AWS KMS keys em diferentes Regiões da AWS que têm o mesmo ID de chave e material de chave. É possível usar essas chaves relacionadas como se fossem a mesma chave em regiões diferentes. As chaves multirregionais oferecem suporte a cenários comuns de recuperação de desastres e backup que exigem criptografia em uma região e descriptografia em uma região diferente sem fazer uma chamada entre regiões para AWS KMS. Para obter mais informações sobre chaves multirregionais, consulte [Usar chaves multirregionais](#) no Guia do Desenvolvedor do AWS Key Management Service.

Para oferecer suporte a chaves multirregionais, o AWS Encryption SDK inclui tokens de autenticação compatíveis com várias regiões e fornecedores de chaves mestras do AWS KMS. O novo símbolo de

reconhecimento de várias regiões em cada linguagem de programação é compatível com a chaves de região única e de várias regiões.

- Para chaves de região única, o símbolo de reconhecimento de várias regiões se comporta da mesma forma que o token de autenticação de região única do AWS KMS e o provedor da chave mestra. Ele tenta descriptografar o texto cifrado somente com a chave de região única que criptografou os dados.
- Para chaves multirregionais, o símbolo de reconhecimento de várias regiões tenta descriptografar o texto cifrado com a mesma chave multirregional que criptografou os dados ou com a chave multirregional relacionada na região especificada.

Nos tokens de autenticação com reconhecimento de várias regiões e nos provedores de chaves mestras que usam mais de uma chave KMS, você pode especificar várias chaves de região única e multirregionais. No entanto, é possível especificar somente uma chave de cada conjunto de chaves de várias regiões relacionadas. Se você especificar mais de um identificador de chave com o mesmo ID de chave, a chamada do construtor falhará.

Você também pode usar uma chave multirregional com tokens de autenticação e provedores de chaves mestras do AWS KMS padrões e de região única. No entanto, deve usar a mesma chave multirregional na mesma região para criptografar e descriptografar. Os tokens de autenticação de região única e os provedores de chaves mestras tentam descriptografar o texto cifrado somente com as chaves que criptografaram os dados.

Os exemplos a seguir mostram como criptografar e descriptografar dados usando chaves multirregionais e os novos de tokens de autenticação e provedores de chaves mestras compatíveis com várias regiões. Esses exemplos criptografam dados na região `us-east-1` e descriptografam os dados na região `us-west-2` usando chaves multirregionais relacionadas em cada região. Antes de executar esses exemplos, substitua o exemplo de ARN de chave multirregional por um valor válido da sua Conta da AWS.

C

Para criptografar com uma chave multirregional, use o método `Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder()` para instanciar o token de autenticação. Especifique uma chave multirregional.

Esse exemplo simples não inclui um [contexto de criptografia](#). Para obter um exemplo que usa um contexto de criptografia em C, consulte [Criptografar e descriptografar strings](#).

Para ver um exemplo completo, consulte [kms_multi_region_keys.cpp](#) no repositório do AWS Encryption SDK for C no 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

Para criptografar com uma chave multirregional na região Leste dos EUA (Norte da Virgínia) (us-east-1), instancie um objeto `CreateAwsKmsMrkKeyringInput` com um identificador de chave para a chave multirregional e um cliente AWS KMS para a região especificada. Em seguida, use o método `CreateAwsKmsMrkKeyring()` para criar o token de autenticação.

O método `CreateAwsKmsMrkKeyring()` cria um token de autenticação com exatamente uma chave multirregional. Para criptografar com várias chaves de encapsulamento, incluindo uma chave multirregional, use o método `CreateAwsKmsMrkMultiKeyring()`.

Para ver um exemplo completo, consulte [AwsKmsMrkKeyringExample.cs](#) no repositório do AWS Encryption SDK para .NET no 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

Este exemplo criptografa o arquivo `hello.txt` com uma chave multirregional na região `us-east-1`. Como o exemplo especifica um ARN de chave que tem um elemento de região, esse exemplo não usa o atributo `region` do parâmetro `--wrapping-keys`.

Quando o ID da chave de encapsulamento não especifica uma região, você pode usar o atributo `region` de `--wrapping-keys` para especificar a região, como `--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

Para criptografar com uma chave multirregional, instancie uma classe `AwsKmsMrkAwareMasterKeyProvider` e especifique uma chave multirregional.

Para ver um exemplo completo, consulte [BasicMultiRegionKeyEncryptionExample.java](#) no repositório do AWS Encryption SDK for Java no 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

Para criptografar com uma chave mulirregional, use o método `buildAwsKmsMrkAwareStrictMultiKeyringBrowser()` para criar o token de autenticação e especificar uma chave multirregional.

Para ver um exemplo completo, consulte [kms_multi_region_simple.ts](#) no repositório do AWS Encryption SDK para JavaScript no 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 AWS Encryption SDK para 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

Para criptografar com uma chave multirregional, use o método `buildAwsKmsMrkAwareStrictMultiKeyringNode()` para criar o token de autenticação e especificar uma chave multirregional.

Para ver um exemplo completo, consulte [kms_multi_region_simple.ts](#) no repositório do AWS Encryption SDK para JavaScript no 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

Para criptografar com uma chave AWS KMS multirregional, use o método `MRKAwareStrictAwsKmsMasterKeyProvider()` e especifique uma chave multirregional.

Para ver um exemplo completo, consulte [mrk_aware_kms_provider.py](#) no repositório do AWS Encryption SDK for Python no 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
)
```

Em seguida, mova seu texto cifrado para a região us-west-2. Não é necessário recriptografar o texto cifrado.

Para descriptografar o texto cifrado no modo estrito na região us-west-2, instancie o símbolo de reconhecimento de várias regiões com o ARN da chave multirregional relacionada na região us-west-2. Se você especificar o ARN da chave de uma chave multirregional relacionada em uma região diferente (incluindo us-east-1), o símbolo compatível com várias regiões fará uma chamada entre regiões para essa AWS KMS key.

Ao descriptografar no modo estrito, o símbolo com reconhecimento de várias regiões requer um ARN de chave. Ele aceita somente um ARN de chave de cada conjunto de chaves de várias regiões relacionadas.

Antes de executar esses exemplos, substitua o exemplo de ARN de chave multirregional por um valor válido da sua Conta da AWS.

C

Para descriptografar no modo estrito com uma chave multirregional, use o método `Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder()` para instanciar o token de autenticação. Especifique a chave multirregional relacionada na região local (us-west-2).

Para ver um exemplo completo, consulte [kms_multi_region_keys.cpp](#) no repositório do AWS Encryption SDK for C no 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

Para descriptografar no modo estrito com uma única chave multirregional, use os mesmos construtores e métodos usados para montar a entrada e criar o token de autenticação para criptografia. Instancie um objeto `CreateAwsKmsMrkKeyringInput` com o ARN de chave de uma chave multirregional relacionada e um cliente AWS KMS para a região Oeste dos EUA (Oregon) (`us-west-2`). Em seguida, use o método `CreateAwsKmsMrkKeyring()` para criar um token de autenticação multirregional com uma chave do KMS multirregional.

Para ver um exemplo completo, consulte [AwsKmsMrkKeyringExample.cs](#) no repositório do AWS Encryption SDK para .NET no 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

Para descriptografar com a chave multirregional relacionada na região us-west-2, use o atributo `key` do parâmetro `--wrapping-keys` para especificar o ARN da chave.

```
# 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

Para descriptografar no modo estrito, instancie uma `AwsKmsMrkAwareMasterKeyProvider` e especifique a chave multirregional na região local (us-west-2).

Para ver um exemplo completo, consulte [BasicMultiRegionKeyEncryptionExample.java](#) no repositório do AWS Encryption SDK for Java no 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

Para descriptografar em modo estrito, use o método `buildAwsKmsMrkAwareStrictMultiKeyringBrowser()` para criar o token de autenticação e especifique a chave multirregional na região local (us-west-2).

Para ver um exemplo completo, consulte [kms_multi_region_simple.ts](#) no repositório do AWS Encryption SDK para JavaScript no 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 AWS Encryption SDK para 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

Para descriptografar em modo estrito, use o método `buildAwsKmsMrkAwareStrictMultiKeyringNode()` para criar o token de autenticação e especifique a chave multirregional na região local (us-west-2).

Para ver um exemplo completo, consulte [kms_multi_region_simple.ts](#) no repositório do AWS Encryption SDK para JavaScript no 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

Para descriptografar no modo estrito, use o método `MRKAwareStrictAwsKmsMasterKeyProvider()` para criar o provedor de chave mestra. Especifique a chave multirregional relacionada na região local (us-west-2).

Para ver um exemplo completo, consulte [mrk_aware_kms_provider.py](#) no repositório do AWS Encryption SDK for Python no 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
)

```

Você também pode descriptografar no modo de descoberta com chaves AWS KMS multirregionais. Ao descriptografar no modo de descoberta, você não especifica nenhuma AWS KMS keys. (Para

obter informações sobre tokens de autenticação de descoberta do AWS KMS de região única, consulte [Usando um chaveiro AWS KMS Discovery](#).)

Se você criptografou com uma chave multirregional, o símbolo compatível com várias regiões no modo de descoberta tentará descriptografar usando uma chave multirregional relacionada na região local. Se não existir nenhuma, a chamada falhará. No modo de descoberta, o AWS Encryption SDK não tentará fazer uma chamada entre regiões para a chave multirregional usada para criptografia.

Note

Se você usar um símbolo com reconhecimento de várias regiões no modo de descoberta para criptografar dados, a operação de criptografia falhará.

O exemplo a seguir mostra como descriptografar com o símbolo de reconhecimento de várias regiões no modo de descoberta. Como você não especifica um AWS KMS key, o AWS Encryption SDK deve obter a região de uma fonte diferente. Quando possível, especifique explicitamente a região local. Caso contrário, o AWS Encryption SDK obtém a região local da região configurada no SDK da AWS para sua linguagem de programação.

Antes de executar esses exemplos, substitua o exemplo do ID de conta e do ARN de chave multirregional por valores válidos da sua Conta da AWS.

C

Para descriptografar no modo de descoberta com uma chave multirregional, use o método `Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder()` para criar o token de autenticação e o método `Aws::Cryptosdk::KmsKeyring::DiscoveryFilter::Builder()` para criar o filtro de descoberta. Para especificar a região local, defina uma `ClientConfiguration` e especifique-a no AWS KMS cliente.

Para ver um exemplo completo, consulte [kms_multi_region_keys.cpp](#) no repositório do AWS Encryption SDK for C no 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).Build();

/* 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

Para criar um token de autenticação de descoberta compatível com várias regiões no AWS Encryption SDK para .NET, instancie um objeto `CreateAwsKmsMrkDiscoveryKeyringInput`

que leve um cliente AWS KMS para uma determinada partição da Região da AWS e um filtro de descoberta opcional que limita as chaves do KMS a uma partição e conta específica da AWS. Em seguida, chame o método `CreateAwsKmsMrkDiscoveryKeyring()` do objeto `.` Para ver um exemplo completo, consulte [AwsKmsMrkDiscoveryKeyringExample.cs](#) no repositório do AWS Encryption SDK para .NET no GitHub.

Para criar um token de autenticação de descoberta compatível com várias regiões para mais de uma Região da AWS, use o método `CreateAwsKmsMrkDiscoveryMultiKeyring()` para criar um multitoken de autenticação ou use `CreateAwsKmsMrkDiscoveryKeyring()` para criar vários tokens de autenticação de descoberta compatíveis com várias regiões e, em seguida, use o método `CreateMultiKeyring()` para combiná-los em um multitoken de autenticação.

Para ver um exemplo, consulte [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

Para descriptografar no modo de descoberta, use o atributo `discovery` do parâmetro `--wrapping-keys`. Os atributos `discovery-account` e `discovery-partition` criam um filtro de descoberta que é opcional, mas recomendado.

Para especificar a região, esse comando inclui o atributo `region` do parâmetro `--wrapping-keys`.

```
# 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

Para especificar a região local, use o parâmetro `builder().withDiscoveryMrkRegion()`. Caso contrário, o AWS Encryption SDK obtém a região local da região configurada no [AWS SDK for Java](#).

Para ver um exemplo completo, consulte [DiscoveryMultiRegionDecryptionExample.java](#) no repositório do AWS Encryption SDK for Java no 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

Para descriptografar no modo de descoberta com uma chave multirregional simétrica, use o método `AwsKmsMrkAwareSymmetricDiscoveryKeyringBrowser()`.

Para ver um exemplo completo, consulte [kms_multi_region_discovery.ts](#) no repositório do AWS Encryption SDK para JavaScript no 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

Para descriptografar no modo de descoberta com uma chave multirregional simétrica, use o método `AwsKmsMrkAwareSymmetricDiscoveryKeyringNode()`.

Para ver um exemplo completo, consulte [kms_multi_region_discovery.ts](#) no repositório do AWS Encryption SDK para JavaScript no 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

Para descriptografar no modo de descoberta com uma chave multirregional, use o método `MRKAwareDiscoveryAwsKmsMasterKeyProvider()`.

Para ver um exemplo completo, consulte [mrk_aware_kms_provider.py](#) no repositório do AWS Encryption SDK for Python no 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
)
```

Escolher um pacote de algoritmo

O AWS Encryption SDK é compatível com vários [algoritmos de criptografia simétrica e assimétrica](#) para criptografar suas chaves de dados sob as chaves de encapsulamento que você especificar. [No entanto, quando ele usa essas chaves de dados para criptografar seus dados, é padrão do AWS Encryption SDK um pacote de algoritmos recomendado que usa o algoritmo AES-GCM com derivação de chave, assinaturas digitais e confirmação de chave.](#) Embora o pacote de algoritmos padrão seja adequado para a maioria das aplicações, você pode escolher um conjunto alternativo de algoritmos. Por exemplo, alguns modelos de confiança seriam satisfeitos com um pacote de algoritmos sem [assinaturas digitais](#). Para obter informações sobre os pacotes de algoritmos compatíveis com o AWS Encryption SDK, consulte [Pacotes de algoritmos compatíveis no AWS Encryption SDK](#).

Os exemplos a seguir mostram como selecionar um pacote de algoritmos alternativo ao criptografar. Esses exemplos selecionam um pacote de algoritmos recomendado AES-GCM com derivação e confirmação de chaves, mas sem assinaturas digitais. Ao criptografar com um pacote de algoritmos que não inclui assinaturas digitais, use o modo de descryptografia somente sem assinatura ao descryptografar. Esse modo, que falha se encontrar um texto cifrado assinado, é mais útil ao transmitir a decodificação.

C

Para especificar um pacote de algoritmos alternativo no AWS Encryption SDK for C, você deve criar um CMM explicitamente. Em seguida, use o `aws_cryptosdk_default_cmm_set_alg_id` com o CMM e o pacote de algoritmos selecionado.

```
/* 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;
}

```

Ao descriptografar dados que foram criptografados sem assinaturas digitais, use.

`AWS_CRYPTOSDK_DECRYPT_UNSIGNED` Isso faz com que a descriptografia falhe se encontrar um texto cifrado assinado.

```

/* 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

Para especificar um pacote de algoritmos alternativo no AWS Encryption SDK para .NET, especifique a propriedade `AlgorithmSuiteId` de um objeto [EncryptInput](#). O AWS Encryption SDK para .NET inclui [constantes](#) que você pode usar para identificar seu pacote de algoritmos preferido.

O AWS Encryption SDK para .NET não tem um método para detectar texto cifrado assinado durante a transmissão de criptografia porque essa biblioteca não é compatível com dados de streaming.

```

// 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

Ao criptografar o arquivo `hello.txt`, este exemplo usa o parâmetro `--algorithm` para especificar um pacote de algoritmos sem assinaturas digitais.

```
# 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
```

Ao descriptografar, este exemplo usa o parâmetro `--decrypt-unsigned`. Esse parâmetro é recomendado para garantir que você esteja descriptografando texto cifrado não assinado, especialmente com a CLI, que está sempre transmitindo entrada e saída.

```
# 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

Para especificar um pacote de algoritmos alternativo, use o método `AwsCrypto.builder().withEncryptionAlgorithm()`. Este exemplo usa um pacote de algoritmos alternativo sem assinaturas digitais.

```
// 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();
```

Ao transmitir dados para descriptografia, use o método `createUnsignedMessageDecryptingStream()` para garantir que todo texto cifrado que você está descriptografando não esteja assinado.

```
// 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

Para especificar um pacote de algoritmos alternativo, use o parâmetro `suiteId` com um valor `enum AlgorithmSuiteIdentifier`.

```
// 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, })
```

Ao descriptografar, use o método padrão `decrypt`. O AWS Encryption SDK para JavaScript no navegador não tem um modo `decrypt-unsigned` porque o navegador não é compatível com 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

Para especificar um pacote de algoritmos alternativo, use o parâmetro `suiteId` com um valor `enum AlgorithmSuiteIdentifier`.

```
// 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, })
```

Ao descriptografar dados que foram criptografados sem assinaturas digitais, use `decryptUnsignedMessageStream`. Esse método falhará se encontrar texto cifrado assinado.

```
// 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

Para especificar um algoritmo de criptografia alternativo, use o parâmetro `algorithm` com um valor enum `Algorithm`.

```
# Specify an algorithm suite without signing

# Instantiate a client
client =
  aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT,
                                         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
)

```

Ao descriptografar mensagens que foram criptografadas sem assinaturas digitais, use o modo de `decrypt-unsigned streaming`, especialmente ao descriptografar durante o 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
```

Limitar as chaves de dados criptografadas

Você pode limitar o número de chaves de dados criptografadas em uma mensagem criptografada. Esse atributo de práticas recomendadas pode ajudar você a detectar um token de autenticação mal configurado ao criptografar ou um texto cifrado malicioso ao descriptografar. Isso também evita chamadas desnecessárias, caras e potencialmente exaustivas para sua infraestrutura principal. Limitar as chaves de dados criptografadas tem mais valor quando você está descriptografando mensagens de uma fonte não confiável.

Embora a maioria das mensagens criptografadas tenha uma chave de dados criptografada para cada chave de encapsulamento usada na criptografia, uma mensagem criptografada pode conter até 65.535 chaves de dados criptografadas. Um agente mal-intencionado pode criar uma mensagem criptografada com milhares de chaves de dados criptografadas, nenhuma delas capaz de ser descriptografada. Como resultado, o AWS Encryption SDK tentaria descriptografar cada chave de dados criptografada até esgotar as chaves de dados criptografadas na mensagem.

Para limitar as chaves de dados criptografadas, use o parâmetro `MaxEncryptedDataKeys`. Esse parâmetro está disponível para todas as linguagens de programação compatíveis a partir das versões 1.9.x e 2.2.x do AWS Encryption SDK. Ele é opcional e válido ao criptografar e descriptografar. Os exemplos a seguir descriptografam dados que foram criptografados sob três chaves de encapsulamento diferentes. O valor de `MaxEncryptedDataKeys` foi definido como 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

Para limitar as chaves de dados criptografadas no AWS Encryption SDK para .NET, instancie um cliente AWS Encryption SDK para .NET e defina seu parâmetro `MaxEncryptedDataKeys` opcional com o valor desejado. Em seguida, chame o método `Decrypt()` na instância do AWS Encryption SDK configurada.

```

// 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)
```

Criação de um filtro de descoberta

Ao descriptografar dados criptografados com chaves KMS, é uma prática recomendada descriptografar no modo estrito, ou seja, limitar as chaves de encapsulamento usadas somente às que você especificar. No entanto, se necessário, você também poderá descriptografar no modo de descoberta, onde você não especifica nenhuma chave de encapsulamento. Nesse modo, o AWS KMS pode descriptografar a chave de dados criptografada usando a chave KMS que a criptografou, independentemente de quem possui ou tem acesso a essa chave KMS.

Se você for descriptografar em modo de descoberta, recomendamos que sempre use um filtro de descoberta para limitar as chaves do KMS que podem ser utilizadas às chaves em uma Conta da AWS especificada e na [partição](#). O filtro de descoberta é opcional, mas é uma prática recomendada.

Use a tabela a seguir para determinar o valor da partição do seu filtro de descoberta.

região	Partition
Regiões da AWS	aws
Regiões da China	aws-cn
AWS GovCloud (US) Regions	aws-us-gov

Os exemplos nesta seção mostram como criar um filtro de descoberta. Antes de usar o código, substitua os valores de exemplo por valores válidos para a Conta da AWS e partição.

C

Para obter um exemplo completo, consulte: [kms_discovery.cpp](#) no AWS Encryption SDK for 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).Build
```

C# / .NET

Para obter um exemplo completo, consulte [DiscoveryFilterExample.cs](#) no AWS Encryption SDK para .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

Para obter um exemplo completo, consulte: [DiscoveryDecryptionExample.java](#) no AWS Encryption SDK for Java.

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

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

JavaScript (Node and Browser)

Para obter exemplos completos, consulte [kms_filtered_discovery.ts \(Node.js\)](#) e [kms_multi_region_discovery.ts](#) (Navegador) no AWS Encryption SDK para JavaScript.

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

Python

Para obter um exemplo completo, consulte [discovery_kms_provider.py](#) no AWS Encryption SDK for 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",
)
```

Como definir uma política de compromisso

Uma [política de compromisso](#) é uma definição de configuração que determina se sua aplicação criptografa e descriptografa com [confirmação de chave](#). Criptografar e descriptografar com o [confirmar de chave](#) é uma [prática recomendada do AWS Encryption SDK](#).

Definir e ajustar sua política de compromisso é uma etapa fundamental na [migração](#) das versões 1.7.x e anteriores do AWS Encryption SDK às versões 2.0x posteriores. Essa progressão é explicada em detalhes no [tópico de migração](#).

O valor padrão da política de compromisso nas versões mais recentes do AWS Encryption SDK (a partir da versão 2.0.x), `RequireEncryptRequireDecrypt`, é ideal para a maioria das situações. No entanto, se você precisar descriptografar um texto cifrado que foi criptografado sem confirmação de chave, talvez seja necessário alterar sua política de compromisso para `RequireEncryptAllowDecrypt`. Para obter exemplos de como definir uma política de compromisso em cada linguagem de programação, consulte [Como definir sua política de compromisso](#).

Trabalhar com streaming de dados

Ao transmitir dados para descryptografia, lembre-se de que o AWS Encryption SDK retorna texto simples descryptografado após a conclusão das verificações de integridade, mas antes da verificação da assinatura digital. Para garantir que você não retorne ou use texto simples até que a assinatura seja verificada, recomendamos que você armazene o texto simples transmitido até que todo o processo de descryptografia seja concluído.

Esse problema surge somente quando você está transmitindo texto cifrado para decodificação e somente quando você está usando um pacote de algoritmos, como [opacote de algoritmos padrão](#), que inclui [assinaturas digitais](#).

Para facilitar o armazenamento em buffer, algumas implementações de linguagem do AWS Encryption SDK, como AWS Encryption SDK para JavaScript no Node.js, incluem um atributo de buffer como parte do método decrypt. A CLI de criptografia da AWS, que sempre transmite entrada e saída, introduziu um parâmetro `--buffer` nas versões 1.9.x e 2.2.x.. Em outras implementações de linguagem, você pode usar os atributos de buffer existentes. (O AWS Encryption SDK para .NET não é compatível com streaming.)

Se você estiver usando um pacote de algoritmos sem assinaturas digitais, certifique-se de usar o atributo `decrypt-unsigned` em cada implementação de linguagem. Esse atributo descryptografa o texto cifrado, mas falhará se encontrar um texto cifrado assinado. Para obter mais detalhes, consulte [Escolher um pacote de algoritmo](#).

Armazenamento em cache de chaves de dados

Em geral, a reutilização de chaves de dados não é incentivada, mas o AWS Encryption SDK oferece uma opção de [armazenamento em cache de chaves de dados](#) que fornece reutilização limitada de chaves de dados. O armazenamento em cache de chaves de dados pode melhorar a performance de algumas aplicações e reduzir as chamadas para sua infraestrutura de chaves. Antes de usar o armazenamento em cache de chaves de dados em produção, ajuste os [limites de segurança](#) e teste, para garantir que os benefícios superem as desvantagens da reutilização de chaves de dados.

Usar tokens de autenticação

O AWS Encryption SDK for C, o AWS Encryption SDK para JavaScript, o e o AWS Encryption SDK for Java para.NET usam chaveiros AWS Encryption SDK para realizar a criptografia de [envelopes](#). Tokens de autenticação geram, criptografam e descriptografam chaves de dados. Os tokens de autenticação determinam a origem das chaves de dados exclusivas que protegem cada mensagem, bem como as [chaves de encapsulamento](#) que criptografam essa chave de dados. Você especifica um token de autenticação ao criptografar e especifica o mesmo ou outro token de autenticação ao descriptografar. Você pode usar os tokens de autenticação fornecidos pelo SDK ou elaborar seus próprios tokens de autenticação personalizados compatíveis.

É possível usar cada token individualmente ou combiná-los em um [multitoken de autenticação](#). Embora a maioria dos tokens de autenticação possa gerar, criptografar e descriptografar chaves de dados, você pode criar um que execute apenas uma operação, por exemplo, um token que gere apenas chaves de dados, e usá-lo em combinação com outros.

Recomendamos que você use um chaveiro que proteja suas chaves de agrupamento e execute operações criptográficas dentro de um limite seguro, como o AWS KMS chaveiro, que usa AWS KMS keys that never leave () sem criptografia. [AWS Key Management Service](#) AWS KMS Também é possível escrever um token de autenticação que use chaves de encapsulamento armazenadas em seus módulos de segurança de hardware (HSMs) ou sejam protegidas por outros serviços de chave mestra. Para obter detalhes, consulte o tópico [Interface do token de autenticação](#) na Especificação do AWS Encryption SDK .

Os chaveiros desempenham o papel de [chaves mestras](#) e [provedores de chaves mestras](#) no AWS Encryption SDK for Java, AWS Encryption SDK for Python, e na CLI de AWS criptografia. Se você usar diferentes implementações de linguagem do AWS Encryption SDK para criptografar e descriptografar seus dados, certifique-se de usar tokens de autenticação e provedores de chaves mestras compatíveis. Para obter detalhes, consulte [Compatibilidade dos tokens de autenticação](#).

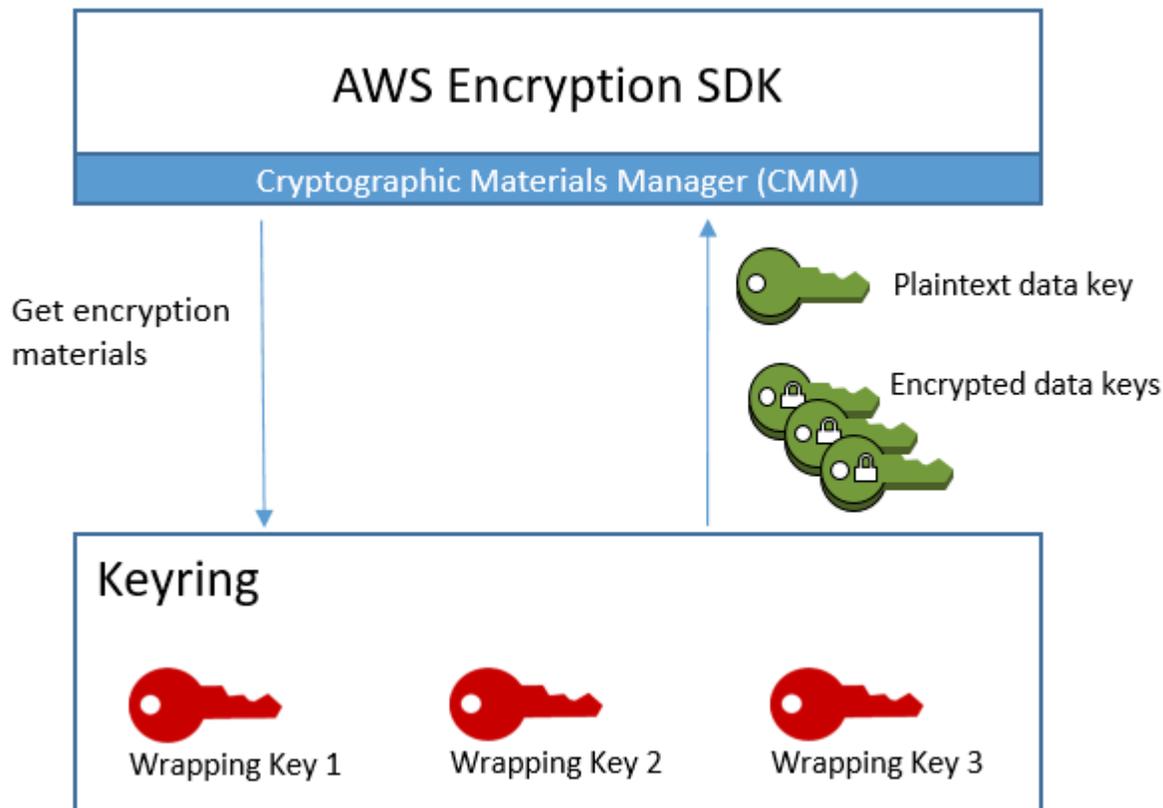
Este tópico explica como usar o recurso de chaveiro do AWS Encryption SDK e como escolher um chaveiro. Para exemplos de criação e uso de chaveiros, consulte o [C](#) e os [JavaScript](#) tópicos.

Tópicos

- [Como os tokens de autenticação funcionam](#)
- [Compatibilidade dos tokens de autenticação](#)
- [Como escolher um token de autenticação](#)

Como os tokens de autenticação funcionam

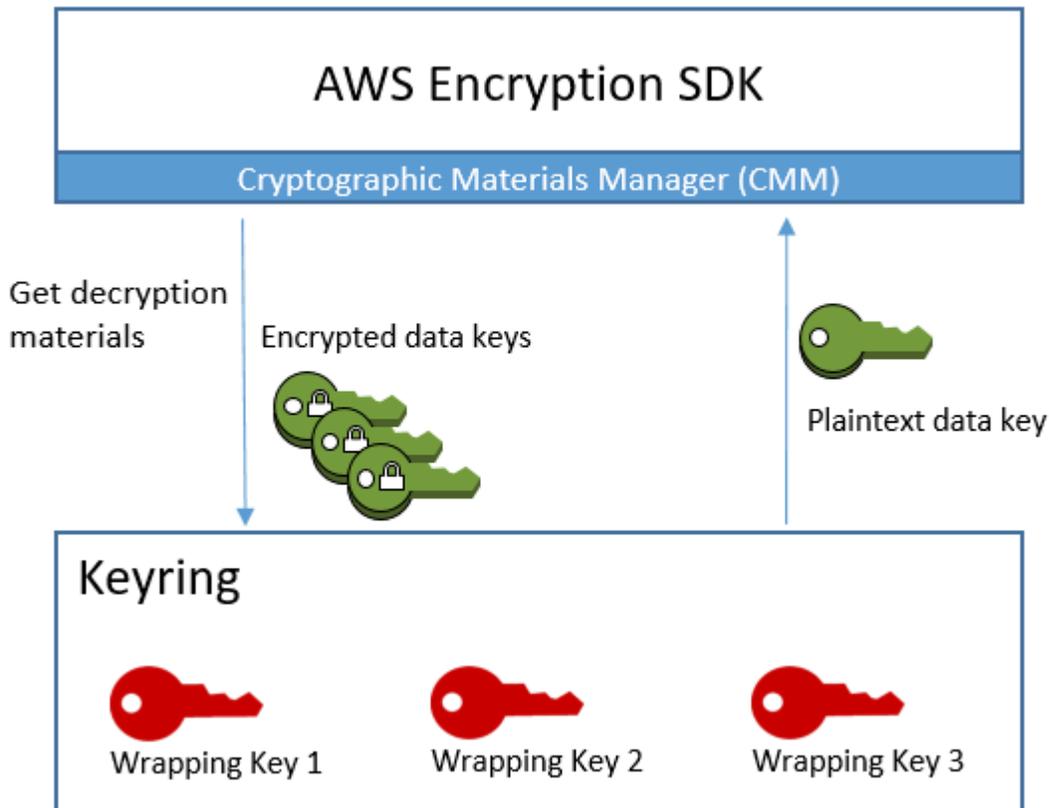
Quando você criptografa dados, AWS Encryption SDK solicita materiais de criptografia ao chaveiro. O token de autenticação retorna uma chave de dados de texto simples e uma cópia da chave de dados que é criptografada por cada uma das chaves de encapsulamento no token de autenticação. O AWS Encryption SDK usa a chave de texto simples para criptografar os dados e, em seguida, destrói a chave de dados de texto sem formatação. Em seguida, AWS Encryption SDK retorna uma [mensagem criptografada](#) que inclui as chaves de dados criptografadas e os dados criptografados.



Ao descriptografar dados, você pode usar o mesmo token de autenticação usado para criptografar os dados ou um token diferente. Para descriptografar os dados, um token de autenticação de descriptografia deve incluir (ou ter acesso a) pelo menos uma chave de encapsulamento no token de autenticação de criptografia.

Ele AWS Encryption SDK passa as chaves de dados criptografadas da mensagem criptografada para o chaveiro e solicita que o chaveiro decifre qualquer uma delas. O token de autenticação usa suas chaves de empacotamento para descriptografar uma das chaves de dados criptografadas

e retorna uma chave de dados de texto simples. O AWS Encryption SDK usa a chave de dados de texto simples para descriptografar os dados. Se nenhuma das chaves de empacotamento no token de autenticação puder descriptografar qualquer uma das chaves de dados criptografadas, a operação de descriptografia falhará.



Você pode usar um único token de autenticação ou também combinar tokens de autenticação do mesmo ou outro tipo em um [multitoken de autenticação](#). Quando você criptografa dados, o multitoken de autenticação retorna uma cópia da chave de dados criptografada por todas as chaves de empacotamento em todos os tokens de autenticação que compreendem o multitoken de autenticação. É possível descriptografar os dados usando um token de autenticação com qualquer uma das chaves de encapsulamento no multitoken de autenticação.

Compatibilidade dos tokens de autenticação

Embora as diferentes implementações de linguagem do AWS Encryption SDK tenham algumas diferenças arquitetônicas, elas são totalmente compatíveis, sujeitas às restrições de linguagem. Você pode criptografar seus dados usando uma implementação de linguagem e descriptografá-los em qualquer outra implementação de linguagem. No entanto, é necessário usar as mesmas chaves de

encapsulamento, ou correspondentes, para criptografar e descriptografar suas chaves de dados. Para obter informações sobre restrições de linguagem, consulte o tópico sobre cada implementação de linguagem, como [the section called “Compatibilidade”](#) no AWS Encryption SDK para JavaScript tópico.

Requisitos variados para tokens de autenticação de criptografia

Em implementações de AWS Encryption SDK linguagem diferentes da AWS Encryption SDK for C, todas as chaves agrupadas em um chaveiro de criptografia (ou vários chaveiros) ou provedor de chave mestra devem ser capazes de criptografar a chave de dados. Se alguma chave de encapsulamento falhar na criptografia, o método de criptografia falhará. Como resultado, o chamador deve ter as [permissões necessárias](#) para todas as chaves no token de autenticação. Se você usar um token de autenticação para criptografar dados, sozinho ou em um token de autenticação múltiplo, a operação de criptografia falhará.

A exceção é a AWS Encryption SDK for C, em que a operação de criptografia ignora um chaveiro de descoberta padrão, mas falha se você especificar um chaveiro de descoberta de várias regiões, sozinho ou em um chaveiro com várias chaves.

Tokens de autenticação e provedores de chaves mestras compatíveis

A tabela a seguir mostra quais chaves mestras e fornecedores de chaves mestras são compatíveis com os chaveiros fornecidos pela empresa AWS Encryption SDK . Qualquer pequena incompatibilidade devido às restrições de linguagem é explicada no tópico sobre a implementação de linguagem.

Token de autenticação:	Provedor de chaves mestras:
AWS KMS chaveiro	KMS MasterKey (Java) KMS MasterKeyProvider (Java) KMS MasterKey (Python) KMS MasterKeyProvider (Python)

 **Note**
O AWS Encryption SDK for Python e AWS Encryption SDK for Java não inclui uma chave mestra ou um provedor de

Token de autenticação:	Provedor de chaves mestras: chave mestra que seja equivalente ao chaveiro AWS KMS regional Discovery .
AWS KMS Chaveiro hierárquico	Disponível apenas com a versão 4. x do AWS Encryption SDK para .NET e versão 3. x do AWS Encryption SDK for Java.
AWS KMS Chaveiro ECDH	Disponível apenas com a versão 3. x do AWS Encryption SDK for Java.
Token de autenticação bruto do AES	Quando são usados com chaves de criptografia simétrica: JceMasterKey (Java) RawMasterKey (Python)
Token de autenticação bruto do RSA	Quando são usados com chaves de criptografia assimétrica: JceMasterKey (Java) RawMasterKey (Python) <div data-bbox="516 1045 1507 1455" style="border: 1px solid #add8e6; border-radius: 15px; padding: 10px;"> <p> Note</p> <p>O token de autenticação RSA bruto não oferece suporte a chaves do KMS assimétricas. Se você quiser usar chaves RSA KMS assimétricas, versão 4. x do AWS Encryption SDK for.NET suporta AWS KMS chaveiros que usam criptografia simétrica (SYMMETRIC_DEFAULT) ou RSA assimétrica. AWS KMS keys</p> </div>
Chaveiro ECDH bruto	Disponível apenas com a versão 3. x do AWS Encryption SDK for Java.

Como escolher um token de autenticação

O token de autenticação determina as chaves de encapsulamento que protegem as chaves de dados e, em última análise, os dados. Use as chaves de encapsulamento mais seguras e práticas para

sua tarefa. Sempre que possível, use chaves de encapsulamento protegidas por um módulo de segurança de hardware ou por uma infraestrutura de gerenciamento de chaves, como chaves KMS no [AWS Key Management Service](#) (AWS KMS) ou chaves de criptografia do [AWS CloudHSM](#).

AWS Encryption SDK Ele fornece vários chaveiros e configurações de chaveiros em várias linguagens de programação, e você pode criar seus próprios chaveiros personalizados. Você também pode criar um [multitoken de autenticação](#) que inclua um ou mais tokens de autenticação do mesmo tipo ou de um tipo diferente.

Tópicos

- [AWS KMS chaveiros](#)
- [AWS KMS Chaveiros hierárquicos](#)
- [AWS KMS chaveiros ECDH](#)
- [Tokens de autenticação AES Raw](#)
- [Tokens de autenticação brutos do RSA](#)
- [Chaveiros ECDH brutos](#)
- [Multitokens de autenticação](#)

AWS KMS chaveiros

Um AWS KMS chaveiro usa criptografia simétrica [AWS KMS keys](#) para gerar, criptografar e descriptografar chaves de dados. AWS Key Management Service (AWS KMS) protege suas chaves KMS e executa operações criptográficas dentro do limite do FIPS. Recomendamos que você use um AWS KMS chaveiro ou um chaveiro com propriedades de segurança semelhantes, sempre que possível.

Você pode usar uma chave AWS KMS multirregional em um AWS KMS chaveiro ou provedor de chave mestra a partir da [versão 2.3.x](#) do AWS Encryption SDK e versão 3.0.x da CLI AWS de criptografia. Para obter detalhes e exemplos de como usar do novo símbolo compatível com várias regiões, consulte [Usar AWS KMS keys multirregionais](#). Para obter mais informações sobre chaves multirregionais, consulte [Usar chaves multirregionais](#) no Guia do Desenvolvedor do AWS Key Management Service .

Note

Versão 4. x do AWS Encryption SDK para .NET e versão 3. x das AWS Encryption SDK for Java são as únicas implementações de linguagem de programação que suportam AWS KMS chaveiros que usam RSA assimétrico. AWS KMS keys

Se você tentar incluir uma chave do KMS assimétrica em um token de autenticação de criptografia que esteja em outra implementação de linguagem, a chamada de criptografia falhará. Se você incluí-la em um token de autenticação de descryptografia, ele será ignorado. Todas as menções aos chaveiros KMS AWS Encryption SDK se referem aos chaveiros.

AWS KMS

AWS KMS os chaveiros podem incluir dois tipos de chaves de embrulho:

- Chave geradora: gera uma chave de dados em texto simples e a criptografa. Um token de autenticação que criptografa dados deve ter uma chave geradora.
 - Chaves adicionais: criptografa a chave de dados em texto simples gerada pela chave do gerador.
- AWS KMS os chaveiros podem ter zero ou mais chaves adicionais.

Ao criptografar, o AWS KMS chaveiro que você usa deve ter uma chave geradora. Para a descryptografia, a chave geradora é opcional e a distinção entre chaves geradoras e chaves adicionais é ignorada.

Quando um chaveiro de AWS KMS criptografia tem apenas uma AWS KMS chave, essa chave é usada para gerar e criptografar a chave de dados.

Como todos os chaveiros, os AWS KMS chaveiros podem ser usados de forma independente ou em um [chaveiro múltiplo com outros chaveiros](#) do mesmo tipo ou de um tipo diferente.

Tópicos

- [Permissões necessárias para tokens de autenticação do AWS KMS](#)
- [Identificação AWS KMS keys em um AWS KMS chaveiro](#)
- [Criação de um AWS KMS chaveiro para criptografia](#)
- [Criando um AWS KMS chaveiro para decodificação](#)
- [Usando um chaveiro AWS KMS Discovery](#)
- [Usando um chaveiro de descoberta AWS KMS regional](#)

Permissões necessárias para tokens de autenticação do AWS KMS

O AWS Encryption SDK não requer um Conta da AWS e não depende de nenhum AWS service (Serviço da AWS). No entanto, para usar um AWS KMS chaveiro, você precisa de uma Conta da AWS e das seguintes permissões mínimas AWS KMS keys em seu chaveiro.

- Para criptografar com um AWS KMS chaveiro, você precisa da GenerateDataKey permissão [kms:](#) na chave do gerador. Você precisa da permissão [KMS:Encrypt](#) em todas as chaves adicionais no chaveiro. AWS KMS
- Para descriptografar com um AWS KMS chaveiro, você precisa da permissão [kms:Decrypt](#) em pelo menos uma chave no chaveiro. AWS KMS
- Para criptografar com um chaveiro múltiplo composto por AWS KMS chaveiros, você precisa da GenerateDataKey permissão [kms:](#) na chave do gerador no chaveiro do gerador. Você precisa da permissão [kms:Encrypt](#) em todas as outras chaves em todos os outros tokens de autenticação do AWS KMS .

Para obter informações detalhadas sobre permissões para AWS KMS keys, consulte [Autenticação e controle de acesso](#) no Guia do AWS Key Management Service desenvolvedor.

Identificação AWS KMS keys em um AWS KMS chaveiro

Um AWS KMS chaveiro pode incluir um ou mais AWS KMS keys. Para especificar um AWS KMS key em um AWS KMS chaveiro, use um identificador de AWS KMS chave compatível. Os identificadores de chave que você pode usar para identificar um AWS KMS key em um chaveiro variam de acordo com a operação e a implementação da linguagem. Para obter detalhes sobre os identificadores de chave de uma AWS KMS key, consulte [Identificadores de chave](#) no Guia do Desenvolvedor do AWS Key Management Service .

Como prática recomendada, use o identificador de chave mais específico que seja prático para sua tarefa.

- Em um chaveiro de criptografia para o AWS Encryption SDK for C, você pode usar um [ARN de chave](#) ou um [alias ARN](#) para identificar chaves KMS. Em todas as outras implementações de linguagem, você pode usar um [ID de chave](#), [ARN de chave](#), [nome de alias](#) ou [ARN de alias](#) para criptografar dados.
- Em um token de autenticação de descriptografia, você deve usar um ARN de chave para identificar AWS KMS keys. Esse requisito aplica-se a todas as implementações de linguagem do AWS Encryption SDK. Para obter detalhes, consulte [Seleção de chaves de encapsulamento](#).

- Em um token de autenticação usado para criptografia e descriptografia, você deve usar um ARN de chave para identificar AWS KMS keys. Esse requisito aplica-se a todas as implementações de linguagem do AWS Encryption SDK.

Se você especificar um nome de alias ou um ARN de alias para uma chave do KMS em um token de autenticação de criptografia, a operação de criptografia salvará o ARN de chave atualmente associado ao alias nos metadados da chave de dados criptografada. Isso não salva o alias. As alterações no alias não afetam a chave do KMS usada para descriptografar suas chaves de dados criptografadas.

Criação de um AWS KMS chaveiro para criptografia

Você pode configurar cada AWS KMS chaveiro com um único AWS KMS key ou vários AWS KMS keys no mesmo ou em diferentes Contas da AWS e. Regiões da AWS As AWS KMS keys devem ser chaves de criptografia simétrica (SYMMETRIC_DEFAULT). Também é possível usar uma [chave do KMS multirregional](#) de criptografia simétrica. Como ocorre com todos os tokens de autenticação, é possível usar um ou mais tokens de autenticação do AWS KMS em um [multitoken de autenticação](#).

Ao criar um AWS KMS chaveiro para criptografar dados, você deve especificar uma chave geradora, AWS KMS key que é usada para gerar uma chave de dados em texto simples e criptografá-la. A chave de dados não tem relação matemática com a chave KMS. Em seguida, se quiser, você pode especificar outras AWS KMS keys que criptografem a mesma chave de dados de texto sem formatação.

Para descriptografar a mensagem criptografada protegida por esse chaveiro, o chaveiro que você usa deve incluir pelo menos um dos AWS KMS keys definidos no chaveiro, ou não. AWS KMS keys (Um AWS KMS chaveiro sem AWS KMS keys é conhecido como [chaveiro AWS KMS Discovery](#).)

Em implementações de AWS Encryption SDK linguagem diferentes da AWS Encryption SDK for C, todas as chaves agrupadas em um chaveiro de criptografia ou em vários chaveiros devem ser capazes de criptografar a chave de dados. Se alguma chave de encapsulamento falhar na criptografia, o método de criptografia falhará. Como resultado, o chamador deve ter as [permissões necessárias](#) para todas as chaves no token de autenticação. Se você usar um token de autenticação para criptografar dados, sozinho ou em um token de autenticação múltiplo, a operação de criptografia falhará. A exceção é a AWS Encryption SDK for C, em que a operação de criptografia ignora um chaveiro de descoberta padrão, mas falha se você especificar um chaveiro de descoberta de várias regiões, sozinho ou em um chaveiro com várias chaves.

Os exemplos a seguir criam um AWS KMS chaveiro com uma chave geradora e uma chave adicional. Esses exemplos usam [ARNs de chave](#) para identificar as chaves do KMS. Essa é uma prática recomendada para AWS KMS chaveiros usados para criptografia e um requisito para AWS KMS chaveiros usados para decodificação. Para obter detalhes, consulte [Identificação AWS KMS keys em um AWS KMS chaveiro](#).

C

Para identificar um AWS KMS key em um chaveiro de criptografia no AWS Encryption SDK for C, especifique o ARN da [chave ou o ARN](#) do [alias](#). Em um token de autenticação de descryptografia, é necessário usar um ARN de chave. Para obter detalhes, consulte [Identificação AWS KMS keys em um AWS KMS chaveiro](#).

Para obter um exemplo completo, consulte [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

Para criar um AWS KMS chaveiro com uma ou várias AWS KMS chaves no para.NET, crie um chaveiro múltiplo. AWS Encryption SDK O AWS Encryption SDK for.NET inclui um chaveiro múltiplo apenas para AWS KMS chaves.

[Ao especificar um AWS KMS key para um chaveiro de criptografia no AWS Encryption SDK para.NET, você pode usar qualquer identificador de chave válido: um ID de chave, ARN de chave, nome de alias ou ARN de alias.](#) Para obter ajuda para identificar o AWS KMS keys em um AWS KMS chaveiro, consulte [Identificação AWS KMS keys em um AWS KMS chaveiro](#).

O exemplo a seguir usa a versão 4. x do AWS Encryption SDK para que o.NET crie um AWS KMS chaveiro com uma chave geradora e chaves adicionais. Para ver um exemplo completo, consulte [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

[Ao especificar um AWS KMS key para um chaveiro de criptografia no AWS Encryption SDK para JavaScript, você pode usar qualquer identificador de chave válido: um ID de chave, ARN da chave, nome do alias ou ARN do alias.](#) Para obter ajuda para identificar o AWS KMS keys em um AWS KMS chaveiro, consulte [Identificação AWS KMS keys em um AWS KMS chaveiro](#).

Para ver um exemplo completo, consulte [kms_simple.ts](#) no repositório em. AWS Encryption SDK para 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

[Ao especificar um AWS KMS key para um chaveiro de criptografia no AWS Encryption SDK para JavaScript, você pode usar qualquer identificador de chave válido: um ID de chave, ARN da](#)

[chave, nome do alias ou ARN do alias](#). Para obter ajuda para identificar o AWS KMS keys em um AWS KMS chaveiro, consulte [Identificação AWS KMS keys em um AWS KMS chaveiro](#).

Para ver um exemplo completo, consulte [kms_simple.ts](#) no repositório em AWS Encryption SDK para 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

Para criar um AWS KMS chaveiro com uma ou várias AWS KMS chaves no AWS Encryption SDK for Java, crie um chaveiro múltiplo. AWS Encryption SDK for Java Inclui um chaveiro múltiplo apenas para AWS KMS chaves.

[Ao especificar um AWS KMS key para um chaveiro de criptografia no AWS Encryption SDK for Java, você pode usar qualquer identificador de chave válido: um ID de chave, ARN da chave, nome do alias ou ARN do alias](#). Para obter ajuda para identificar o AWS KMS keys em um AWS KMS chaveiro, consulte [Identificação AWS KMS keys em um AWS KMS chaveiro](#).

Para ver um exemplo completo, consulte [BasicEncryptionKeyringExample.java](#) no AWS Encryption SDK for Java repositório em 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);
```

Criando um AWS KMS chaveiro para decodificação

Você também especifica um AWS KMS chaveiro ao descriptografar a [mensagem criptografada que ele retorna](#). AWS Encryption SDK Se o chaveiro de descriptografia especificar AWS KMS keys, eles AWS Encryption SDK usarão somente essas chaves de encapsulamento para descriptografar as chaves de dados criptografadas na mensagem criptografada. (Você também pode usar um [chaveiro de AWS KMS descoberta](#), que não especifica nenhum AWS KMS keys.)

Ao descriptografar, ele AWS Encryption SDK pesquisa no AWS KMS chaveiro por uma AWS KMS key que possa descriptografar uma das chaves de dados criptografadas. Especificamente, o AWS Encryption SDK usa o seguinte padrão para cada chave de dados criptografada em uma mensagem criptografada.

- O AWS Encryption SDK obtém o ARN da chave AWS KMS key que criptografou a chave de dados a partir dos metadados da mensagem criptografada.
- O AWS Encryption SDK pesquisa no chaveiro de decodificação por um AWS KMS key com um ARN de chave correspondente.
- Se encontrar um ARN AWS KMS key com uma chave correspondente no chaveiro, ele AWS Encryption SDK solicitará o uso da chave KMS AWS KMS para descriptografar a chave de dados criptografada.
- Caso contrário, ele passará para a próxima chave de dados criptografada, se houver.

Ele AWS Encryption SDK nunca tenta descriptografar uma chave de dados criptografada, a menos que o ARN da chave AWS KMS key que criptografou essa chave de dados esteja incluído no chaveiro de decodificação. Se o chaveiro de decodificação não incluir os ARNs de nenhuma das chaves de dados AWS KMS keys criptografadas, a chamada de decodificação AWS Encryption SDK falhará sem nunca ligar. AWS KMS

A partir da [versão 1.7. x](#), [ao descriptografar uma chave de dados criptografada, AWS Encryption SDK sempre passa o ARN da chave AWS KMS key para o KeyId parâmetro da operação Decrypt.](#)

[AWS KMS](#) Identificar o AWS KMS key ao descriptografar é uma prática AWS KMS recomendada que garante que você descriptografe a chave de dados criptografada com a chave de empacotamento que você pretende usar.

Uma chamada de decodificação com um AWS KMS chaveiro é bem-sucedida quando pelo menos uma no chaveiro de decodificação pode descriptografar uma das chaves de dados criptografadas AWS KMS key na mensagem criptografada. Além disso, o chamador deve ter a permissão `kms:Decrypt` para essa AWS KMS key. Esse comportamento permite que você Regiões da AWS criptografe dados AWS KMS keys em vários em diferentes contas, mas forneça um chaveiro de descriptografia mais limitado, adaptado a uma conta, região, usuário, grupo ou função específica.

Ao especificar um AWS KMS key em um chaveiro de descriptografia, você deve usar o ARN da chave. Caso contrário, AWS KMS key o não será reconhecido. Para ajudar a encontrar o ARN de chave, consulte [Descobrir o ID de chave e o ARN](#) no Guia do desenvolvedor do AWS Key Management Service .

Note

Se você reutilizar um token de autenticação de criptografia para descriptografar, verifique se as AWS KMS keys no token de autenticação são identificadas por seus ARNs de chave.

Por exemplo, o AWS KMS chaveiro a seguir inclui somente a chave adicional que foi usada no chaveiro de criptografia. No entanto, em vez de se referir à chave adicional por meio de seu alias, `alias/exampleAlias`, o exemplo usa o ARN de chave da chave adicional, conforme exigido pelas chamadas de descriptografia.

Você pode usar esse token de autenticação para descriptografar uma mensagem que foi criptografada sob a chave geradora e a chave adicional, desde que você tenha permissão para usar a chave adicional para descriptografar dados.

C

```
const char * additional_key = "arn:aws:kms:us-  
west-2:111122223333:key/0987dcb-a-09fe-87dc-65ba-ab0987654321"  
  
struct aws_cryptosdk_keyring *kms_decrypt_keyring =  
    Aws::Cryptosdk::KmsKeyring::Builder().Build(additional_key);
```

C# / .NET

Como esse chaveiro de decodificação inclui somente uma AWS KMS chave, o exemplo usa o `CreateAwsKmsKeyring()` método com uma instância de seu objeto. `CreateAwsKmsKeyringInput` Para criar um AWS KMS chaveiro com uma AWS KMS chave, você pode usar um chaveiro de uma ou várias teclas. Para obter detalhes, consulte [Criptografia de dados no AWS Encryption SDK para .NET](#). O exemplo a seguir usa a versão 4. x do AWS Encryption SDK para o.NET para criar um AWS KMS chaveiro para decodificação.

```
// 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

Como esse chaveiro de decodificação inclui somente uma AWS KMS chave, o exemplo usa o `CreateAwsKmsKeyring()` método com uma instância de seu objeto. `CreateAwsKmsKeyringInput` Para criar um AWS KMS chaveiro com uma AWS KMS chave, você pode usar um chaveiro de uma ou várias teclas.

```
// 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);
```

Você também pode usar um AWS KMS chaveiro que especifica uma chave geradora para descryptografia, como a seguinte. Ao descryptografar, o AWS Encryption SDK ignora a distinção entre chaves geradoras e chaves adicionais. Ele pode usar qualquer um dos especificados AWS KMS keys para descryptografar uma chave de dados criptografada. A chamada para é AWS KMS bem-sucedida somente quando o chamador tem permissão para usá-la para AWS KMS key descryptografar dados.

C

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

C# / .NET

O exemplo a seguir usa a versão 4.x do AWS Encryption SDK para .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);
```

Ao contrário de um chaveiro de criptografia que usa todos os itens especificados AWS KMS keys, você pode descriptografar uma mensagem criptografada usando um chaveiro de decodificação que inclui AWS KMS keys aqueles que não estão relacionados à mensagem criptografada e AWS KMS keys que o chamador não tem permissão para usar. Se ocorrer uma falha em uma chamada de descriptografia do AWS KMS, por exemplo, quando o chamador não tem a permissão necessária, o AWS Encryption SDK simplesmente pula para a próxima chave de dados criptografada.

Usando um chaveiro AWS KMS Discovery

Ao descriptografar, é uma [prática recomendada](#) especificar as chaves de encapsulamento que podem ser usadas. AWS Encryption SDK Para seguir essa prática recomendada, use um chaveiro de AWS KMS decodificação que limite as chaves de AWS KMS encapsulamento às que você especificar. No entanto, você também pode criar um chaveiro de AWS KMS descoberta, ou seja, um AWS KMS chaveiro que não especifique nenhuma chave de agrupamento.

AWS Encryption SDK Fornece um chaveiro de AWS KMS descoberta padrão e um chaveiro de descoberta para chaves AWS KMS multirregionais. Para obter informações sobre o uso de chaves multirregionais com o AWS Encryption SDK, consulte [Usar AWS KMS keys multirregionais](#).

Como não especifica nenhuma chave de encapsulamento, um token de autenticação de descoberta não pode criptografar dados. Se você usar um token de autenticação para criptografar dados, sozinho ou em um token de autenticação múltiplo, a operação de criptografia falhará. A exceção é a AWS Encryption SDK for C, em que a operação de criptografia ignora um chaveiro de descoberta padrão, mas falha se você especificar um chaveiro de descoberta de várias regiões, sozinho ou em um chaveiro com várias chaves.

Ao descriptografar, um chaveiro de descoberta permite que você solicite AWS Encryption SDK AWS KMS a decodificação de qualquer chave de dados criptografada usando AWS KMS key aquela que a criptografou, independentemente de quem a possui ou tem acesso a ela. AWS KMS key A chamada

será bem-sucedida somente quando o chamador tiver a permissão `kms:Decrypt` na AWS KMS key.

Important

Se você incluir um chaveiro de AWS KMS descoberta em um chaveiro de decodificação [múltipla, o chaveiro](#) de descoberta substituirá todas as restrições de chave KMS especificadas por outros chaveiros no chaveiro múltiplo. O token de autenticação múltiplo se comporta como o token de autenticação menos restritivo. Um chaveiro de AWS KMS descoberta não tem efeito na criptografia quando usado sozinho ou em um chaveiro múltiplo.

AWS Encryption SDK Ele fornece um chaveiro AWS KMS Discovery para sua conveniência. No entanto, recomendamos que você use um token de autenticação mais limitado sempre que possível pelas razões a seguir.

- **Autenticidade** — Um chaveiro de AWS KMS descoberta pode usar qualquer chave usada para criptografar uma chave de dados na mensagem criptografada, apenas para AWS KMS key que o chamador tenha permissão para usá-la para descriptografar. AWS KMS key Essa pode não ser a AWS KMS key que o chamador pretende usar. Por exemplo, uma das chaves de dados criptografadas pode ter sido criptografada de forma menos segura AWS KMS key que qualquer pessoa possa usar.
- **Latência e desempenho** — Um chaveiro de AWS KMS descoberta pode ser visivelmente mais lento do que outros chaveiros porque AWS Encryption SDK tenta descriptografar todas as chaves de dados criptografadas, incluindo aquelas criptografadas AWS KMS keys em outras regiões, Contas da AWS e AWS KMS keys que o chamador não tem permissão para usar para descriptografia.

[Se você usa um chaveiro de descoberta, recomendamos que você use um filtro de descoberta para limitar as chaves KMS que podem ser usadas para aquelas em partições Contas da AWS e partições especificadas.](#) Os filtros de descoberta são compatíveis com as versões 1.7.x e posteriores do AWS Encryption SDK. Para obter ajuda para encontrar seu ID de conta e partição, consulte [Seus identificadores da Conta da AWS](#) e [Formato de ARN](#) no. Referência geral da AWS.

O código a seguir instancia um chaveiro de AWS KMS descoberta com um filtro de descoberta que limita as chaves KMS que AWS Encryption SDK podem ser usadas às da aws partição e da conta de exemplo 111122223333.

Antes de usar esse código, substitua os valores de exemplo Conta da AWS e de partição por valores válidos para sua partição Conta da AWS e. Se as chaves do KMS estiverem em regiões da China, use o valor de partição `aws-cn`. Se suas chaves KMS estiverem inseridas AWS GovCloud (US) Regions, use o valor da `aws-us-gov` partição. Para todos os outros Regiões da AWS, use o valor da `aws` partição.

C

Para obter um exemplo completo, consulte: [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

O exemplo a seguir usa a versão 4.x do AWS Encryption SDK para .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

Em JavaScript, você deve especificar explicitamente a propriedade de descoberta.

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

JavaScript Node.js

Em JavaScript, você deve especificar explicitamente a propriedade de descoberta.

```
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);
```

Usando um chaveiro de descoberta AWS KMS regional

Um token de autenticação de descoberta regional AWS KMS é um token de autenticação que não especifica os ARNs das chaves do KMS. Em vez disso, ele permite que o AWS Encryption SDK decodifique usando somente as chaves KMS em particular. Regiões da AWS

Ao descriptografar com um chaveiro de descoberta AWS KMS regional, o AWS Encryption SDK descriptografa qualquer chave de dados criptografada que tenha sido criptografada sob um no especificado. AWS KMS key Região da AWS Para ter sucesso, o chamador deve ter `kms:Decrypt` permissão em pelo menos um dos AWS KMS keys itens especificados Região da AWS que criptografou uma chave de dados.

Como outros token de autenticação de descoberta, o token de autenticação de descoberta regional não afeta a criptografia. Ele funciona somente ao descriptografar mensagens criptografadas. Se você usar um token de autenticação de descoberta regional em um multitoken de autenticação usado para criptografar e descriptografar, ele só será efetivo durante a descriptografia. Se você usar um token de autenticação de descoberta multirregional para criptografar dados, sozinho ou em um token de autenticação com vários tokens de autenticação, a operação de criptografia falhará.

Important

Se você incluir um chaveiro de descoberta AWS KMS regional em um chaveiro de descriptografia [múltiplo, o chaveiro](#) de descoberta regional substituirá todas as restrições de chave KMS especificadas por outros chaveiros no chaveiro múltiplo. O token de autenticação múltiplo se comporta como o token de autenticação menos restritivo. Um chaveiro de AWS KMS descoberta não tem efeito na criptografia quando usado sozinho ou em um chaveiro múltiplo.

O chaveiro de descoberta regional nas AWS Encryption SDK for C tentativas de descriptografar somente com chaves KMS na região especificada. Ao usar um chaveiro de descoberta no AWS Encryption SDK para JavaScript e AWS Encryption SDK para.NET, você configura a região no AWS KMS cliente. Essas AWS Encryption SDK implementações não filtram as chaves KMS por região, mas AWS KMS falharão em uma solicitação de descriptografia de chaves KMS fora da região especificada.

Se você usa um chaveiro de descoberta, recomendamos que você use um filtro de descoberta para limitar as chaves KMS usadas na descriptografia àquelas em partições e partições especificadas.

Contas da AWS Os filtros de descoberta são compatíveis com as versões 1.7.x e posteriores do AWS Encryption SDK.

Por exemplo, o código a seguir cria um chaveiro de descoberta AWS KMS regional com um filtro de descoberta. Esse chaveiro limita as duas AWS Encryption SDK chaves KMS na conta 111122223333 na região Oeste dos EUA (Oregon) (us-west-2).

C

Para exibir esse token de autenticação e o método `create_kms_client` em um exemplo funcional, consulte [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

O AWS Encryption SDK for.NET não tem um chaveiro de descoberta regional dedicado. Porém, você pode usar várias técnicas para limitar as chaves KMS usadas ao descriptografar para uma região específica.

A maneira mais eficiente de limitar as regiões em um token de autenticação de descoberta é usar um token de autenticação de descoberta compatível com várias regiões, mesmo que você tenha criptografado os dados usando somente chaves de uma única região. Quando encontra chaves de uma única região, o token de autenticação compatível com várias regiões não usa nenhum atributo multirregional.

O token de autenticação retornado pelo método `CreateAwsKmsMrkDiscoveryKeyring()` filtra as chaves do por região antes de chamar o AWS KMS. Ele envia uma solicitação de descriptografia AWS KMS somente quando a chave de dados criptografada foi criptografada por uma chave KMS na região especificada pelo `Region` parâmetro no objeto `CreateAwsKmsMrkDiscoveryKeyringInput`

Os exemplos a seguir usam a versão 4. x do AWS Encryption SDK para o.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);
```

Você também pode limitar as chaves KMS a uma determinada Região da AWS especificando uma região na sua instância do AWS KMS cliente () [AmazonKeyManagementServiceClient](#). No entanto, essa configuração é menos eficiente e potencialmente mais cara do que usar um token de autenticação de descoberta compatível com várias regiões. Em vez de filtrar as chaves KMS por região antes da chamada AWS KMS, o AWS Encryption SDK for.NET chama AWS KMS cada chave de dados criptografada (até decifrar uma) e se baseia em limitar as chaves KMS que usa AWS KMS à região especificada.

O exemplo a seguir usa a versão 4.x do AWS Encryption SDK para .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

Para visualizar este token de autenticação e as `limitRegions` e a função, em um exemplo prático, consulte [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);
```

Ele AWS Encryption SDK para JavaScript também exporta uma `excludeRegions` função para o Node.js e o navegador. Essa função cria um chaveiro de descoberta AWS KMS regional que é omitido AWS KMS keys em regiões específicas. O exemplo a seguir cria um chaveiro de descoberta AWS KMS regional que pode ser usado AWS KMS keys na conta 111122223333 em todos, Região da AWS exceto no Leste dos EUA (Norte da Virgínia) (`us-east-1`).

O AWS Encryption SDK for C não tem um método análogo, mas você pode implementá-lo criando um personalizado [ClientSupplier](#).

Este exemplo mostra o código para 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 Chaveiros hierárquicos

Important

O AWS KMS chaveiro hierárquico só é compatível com a versão 4. x do AWS Encryption SDK para .NET e versão 3. x do AWS Encryption SDK for Java.

Com o AWS KMS chaveiro hierárquico, você pode proteger seus materiais criptográficos com uma chave KMS de criptografia simétrica sem ligar AWS KMS toda vez que criptografar ou descriptografar dados. É uma boa opção para aplicações que precisam minimizar as chamadas para o AWS KMS,

e para aplicações que podem reutilizar alguns materiais de criptografia sem violar os requisitos de segurança.

O chaveiro hierárquico é uma solução de armazenamento em cache de materiais criptográficos que reduz o número de AWS KMS chamadas usando chaves de ramificação AWS KMS protegidas persistentes em uma tabela do Amazon DynamoDB e, em seguida, armazenando localmente em cache materiais de chave de ramificação usados em operações de criptografia e descriptografia. A tabela do DynamoDB serve como o armazenamento de chaves de ramificação que gerencia e protege as chaves de ramificação. Ele armazena a chave de ramificação ativa e todas as versões anteriores da chave de ramificação. A chave de ramificação ativa é a versão mais recente da chave de ramificação. O token de autenticação hierárquico usa uma chave de dados exclusiva para criptografar cada mensagem e criptografa cada chave de dados com uma chave de encapsulamento exclusiva derivada da chave de ramificação ativa. O token de autenticação hierárquico depende da hierarquia estabelecida entre as chaves de ramificação ativas e suas chaves de agrupamento derivadas.

O token de autenticação hierárquico normalmente usa cada versão da chave de ramificação para atender a várias solicitações. Porém, você controla até que ponto as chaves de ramificação ativas são reutilizadas e determina com que frequência a chave de ramificação ativa é alternada. A versão ativa da chave de ramificação permanece ativa até que você [a alterne](#). As versões anteriores da chave de ramificação ativa não serão usadas para realizar operações de criptografia, mas ainda podem ser consultadas e usadas em operações de descriptografia.

Quando você instancia o token de autenticação hierárquico, ele cria um cache local. Você especifica um [limite de cache](#) que define o tempo máximo em que os materiais da chave de ramificação são armazenados no cache local antes de expirarem e serem despejados do cache. O chaveiro hierárquico faz uma AWS KMS chamada para descriptografar a chave de ramificação e montar os materiais da chave de ramificação na primeira vez em que a é especificado em uma `branch-key-id` operação. Em seguida, os materiais da chave de ramificação são armazenados no cache local e reutilizados para todas as operações de criptografia e descriptografia que especificam `branch-key-id` até que o limite do cache expire. Armazenar materiais de chave de ramificação no cache local reduz AWS KMS as chamadas. Por exemplo, considere um limite de cache de 15 minutos. Se você realizar 10.000 operações de criptografia dentro desse limite de cache, o [AWS KMS chaveiro tradicional](#) precisaria fazer 10.000 AWS KMS chamadas para satisfazer 10.000 operações de criptografia. Se você tiver um `ativobranch-key-id`, o chaveiro hierárquico só precisará fazer uma AWS KMS chamada para satisfazer 10.000 operações de criptografia.

O cache local consiste em duas partições, uma para operações de criptografia e outra para operações de descriptografia. A partição criptografada armazena os materiais da chave de ramificação reunidos a partir da chave de ramificação ativa e os reutiliza para todas as operações de criptografia até que o limite de cache expire. A partição de descriptografia armazena os materiais da chave de ramificação reunidos para outras versões de chave de ramificação identificadas nas operações de descriptografia. A partição de decodificação pode armazenar várias versões ativas de materiais de chave de ramificação ao mesmo tempo. Quando configurada para usar um fornecedor de ID de chave de ramificação para um ambiente multilocatário, a partição criptografada também pode armazenar várias versões de materiais de chave de ramificação ao mesmo tempo. Para ter mais informações, consulte [Usar o token de autenticação hierárquico em ambientes multilocatários](#).

Note

Todas as menções ao chaveiro hierárquico no AWS Encryption SDK referem-se ao chaveiro hierárquico. AWS KMS

Tópicos

- [Como funciona](#)
- [Pré-requisitos](#)
- [Criar um token de autenticação hierárquico](#)
- [Alternar a chave de ramificação ativa](#)
- [Usar o token de autenticação hierárquico em ambientes multilocatários](#)

Como funciona

As instruções a seguir descrevem como o token de autenticação hierárquico reúne materiais de criptografia e descriptografia e as diferentes chamadas que o token de autenticação faz para operações de criptografia e descriptografia. Para obter detalhes técnicos sobre a derivação da chave de empacotamento e os processos de criptografia da chave de dados em texto simples, consulte [Detalhes técnicos do token de autenticação hierárquico do AWS KMS](#).

Criptografar e assinar

O passo a passo a seguir descreve como o token de autenticação hierárquico reúne materiais de criptografia e obtém uma chave de empacotamento exclusiva.

1. O método de criptografia solicita materiais de criptografia ao token de autenticação hierárquico. O token de autenticação gera uma chave de dados em texto simples e, em seguida, verifica se há materiais de ramificação válidos no cache local para gerar a chave de empacotamento. Se houver materiais de chave de ramificação válidos, o token de autenticação prosseguirá para a Etapa 5.
2. Se não houver materiais de chave de ramificação válidos, o token de autenticação hierárquico consulta o armazenamento de chaves de ramificação em busca da chave de ramificação ativa.
 - a. O repositório de chaves de ramificação faz chamadas AWS KMS para descriptografar a chave de ramificação ativa e retorna a chave de ramificação ativa em texto simples. Os dados que identificam a chave de ramificação ativa são serializados para fornecer dados autenticados adicionais (AAD) na chamada de descriptografia para o AWS KMS.
 - b. O armazenamento da chave de ramificação retorna a chave de ramificação em texto simples e os dados que a identificam, como a versão da chave de ramificação.
3. O token de autenticação hierárquico reúne materiais de chave de ramificação (a chave de ramificação em texto simples e a versão da chave de ramificação) e armazena uma cópia deles no cache local.
4. O token de autenticação hierárquico deriva uma chave de empacotamento exclusiva da chave de ramificação de texto simples e um sal aleatório de 16 bytes. Ele usa a chave de encapsulamento derivada para criptografar uma cópia da chave de dados em texto simples.

O método de criptografia usa os materiais de criptografia para criptografar os dados. Para obter mais informações, consulte [Como o AWS Encryption SDK criptografa dados](#).

Descriptografar e verificar

O passo a passo a seguir descreve como o token de autenticação hierárquico reúne materiais de descriptografia e descriptografa a chave de dados criptografada.

1. O método de descriptografia identifica a chave de dados criptografada da mensagem criptografada e a transmite para o token de autenticação hierárquico.
2. O token de autenticação hierárquico desserializa os dados que identificam a chave de dados criptografada, incluindo a versão da chave de ramificação, o sal de 16 bytes e outras informações que descrevem como a chave de dados foi criptografada.

Para ter mais informações, consulte [AWS KMS Detalhes técnicos do chaveiro hierárquico](#).

3. O token de autenticação hierárquico verifica se há materiais de chave de ramificação válidos no cache local que correspondam à versão da chave de ramificação identificada na Etapa 2. Se houver materiais de chave de ramificação válidos, o token de autenticação prosseguirá para a Etapa 6 .
4. Se não houver materiais de chave de ramificação válidos, o token de autenticação hierárquico consulta o armazenamento da chave de ramificação em busca da chave de ramificação que corresponde à versão da chave de ramificação identificada na Etapa 2.
 - a. O repositório de chaves de ramificação faz chamadas AWS KMS para descriptografar a chave de ramificação e retorna a chave de ramificação ativa em texto simples. Os dados que identificam a chave de ramificação ativa são serializados para fornecer dados autenticados adicionais (AAD) na chamada de descriptografia para o AWS KMS.
 - b. O armazenamento da chave de ramificação retorna a chave de ramificação em texto simples e os dados que a identificam, como a versão da chave de ramificação.
5. O token de autenticação hierárquico reúne materiais de chave de ramificação (a chave de ramificação em texto simples e a versão da chave de ramificação) e armazena uma cópia deles no cache local.
6. O token de autenticação hierárquico usa os materiais de chave de ramificação montados e o sal de 16 bytes identificado na Etapa 2 para reproduzir a chave de empacotamento exclusiva que criptografou a chave de dados.
7. O token de autenticação hierárquico usa a chave de encapsulamento reproduzida para descriptografar a chave de dados e retorna a chave de dados em texto simples.

O método de decodificação usa os materiais de decodificação e a chave de dados de texto simples para descriptografar a mensagem criptografada. Para obter mais informações, consulte [Como o AWS Encryption SDK decodifica uma mensagem criptografada](#).

Pré-requisitos

O AWS Encryption SDK não requer um Conta da AWS e não depende de nenhum AWS service (Serviço da AWS). No entanto, o chaveiro hierárquico depende do Amazon AWS KMS DynamoDB.

[Para usar um chaveiro hierárquico, você precisa de uma criptografia AWS KMS key simétrica com as permissões kms:Decrypt](#). Também é possível usar uma [chave multirregional](#) de criptografia simétrica. Para obter informações detalhadas sobre permissões para AWS KMS keys, consulte [Autenticação e controle de acesso](#) no Guia do AWS Key Management Service desenvolvedor.

Antes de criar e usar um token de autenticação hierárquico, você deve criar seu repositório de chaves de ramificação e preenchê-lo com sua primeira chave de ramificação ativa.

Etapa 1: configurar um novo serviço de armazenamento de chaves

O serviço de armazenamento de chaves fornece várias operações de API, como `CreateKeyStore` e `CreateKey`, para ajudar você a montar os pré-requisitos do token de autenticação hierárquico e gerenciar seu armazenamento de chaves de ramificação.

O exemplo a seguir cria um serviço de armazenamento de chaves. Você deve especificar um nome de tabela do DynamoDB para atuar como o nome do seu armazenamento de chaves de ramificação, um nome lógico para o armazenamento de chaves de ramificação e o ARN da chave KMS que identifica a chave KMS que protegerá suas chaves de ramificação.

O nome do armazenamento lógico de chaves é vinculado criptograficamente a todos os dados armazenados na tabela para simplificar as operações de restauração do DynamoDB. O nome do armazenamento de chaves lógicas pode ser o mesmo nome da tabela do DynamoDB, mas isso não é necessário. É altamente recomendável especificar o nome da tabela do DynamoDB como o nome lógico da tabela ao definir a configuração do serviço de armazenamento de chave pela primeira vez. Você deve sempre especificar o mesmo nome de tabela lógica. Caso o nome do armazenamento de chaves da ramificação mude após a [restauração da tabela do DynamoDB a partir de um backup](#), o nome do armazenamento lógico de chaves será mapeado para o nome da tabela do DynamoDB que você especifica para garantir que o token de autenticação hierárquico ainda possa acessar o armazenamento de chaves da ramificação.

Note

O nome do armazenamento lógico de chaves está incluído no contexto de criptografia de todas as operações de API do serviço de armazenamento de chaves que chamam o AWS KMS. O contexto de criptografia não é secreto, seus valores, incluindo o nome lógico do armazenamento de chaves, aparecem em texto simples nos registros. 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();

```

Etapa 2: chamar o **CreateKeyStore** para criar um armazenamento de chaves de ramificação

A operação a seguir cria o armazenamento de chaves de ramificação que persistirá e protegerá suas chaves de ramificação.

C# / .NET

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

Java

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

A operação `CreateKeyStore` cria uma tabela do DynamoDB com o nome da tabela que você especificou na Etapa 1 e os valores obrigatórios a seguir.

	Chave de partição	Chave de classificação
Tabela base	branch-key-id	type

Note

Você pode criar manualmente a tabela do DynamoDB que serve como seu armazenamento de chaves de ramificação em vez de usar a operação.

CreateKeyStore Se você optar por criar manualmente o armazenamento de chaves de ramificação, deverá especificar os seguintes valores de string para as chaves de partição e classificação:

- Chave de partição: `branch-key-id`
- Chave de classificação: `type`

Etapa 3: chamar `CreateKey` para criar uma nova chave de ramificação ativa

A operação a seguir cria uma nova chave de ramificação ativa usando a chave KMS especificada na Etapa 1 e adiciona a chave de ramificação ativa à tabela do DynamoDB que você criou na Etapa 2.

Ao chamar `CreateKey`, você pode optar por especificar os valores opcionais a seguir.

- Identificador da chave de ramificação: define um personalizado `branch-key-id`.

Para criar um `branch-key-id` personalizado, você também deve incluir um contexto de criptografia adicional com o parâmetro `encryptionContext`.

- [Contexto de criptografia: define um conjunto opcional de pares chave-valor não secretos que fornecem dados autenticados adicionais \(AAD\) no contexto de criptografia incluído na chamada `kms: GenerateDataKeyWithoutPlaintext`](#)

Esse contexto de criptografia adicional é exibido com o prefixo `aws-crypto-ec:`.

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();
```

Primeiro, a operação `CreateKey` gera os valores a seguir.

- Um [Identificador Único Universal](#) (UUID) versão 4 para o `branch-key-id` (a menos que você tenha especificado um `branch-key-id` personalizado).
- Um UUID da versão 4 para a versão da chave de ramificação
- Um timestamp no [formato de data e hora ISO 8601](#) e em UTC (Tempo Universal Coordenado).

Em seguida, a `CreateKey` operação chama [kms: GenerateDataKeyWithoutPlaintext](#) usando a seguinte solicitação.

```
{
  "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"
}
```

Em seguida, a `CreateKey` operação chama [kms: ReEncrypt](#) para criar um registro ativo para a chave de ramificação atualizando o contexto de criptografia.

Por último, a `CreateKey` operação chama [ddb: TransactWriteItems](#) para escrever um novo item que persistirá com a chave de ramificação na tabela que você criou na Etapa 2. O item tem os seguintes atributos:

```
{
  "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"
}
```

Criar um token de autenticação hierárquico

Para inicializar o token de autenticação hierárquico, você deve fornecer os seguintes valores:

- Um nome de armazenamento de chave de ramificação

O nome da tabela do DynamoDB que você criou para servir como seu armazenamento de chaves de ramificação.

-

Um tempo de vida do cache (TTL)

A quantidade de tempo, em segundos, em que uma entrada de materiais de chave de ramificação no cache local pode ser usada antes de expirar. Esse valor deve ser maior que zero. Quando o limite de cache TTL expira, a entrada é removida do cache local.

- Um identificador de chave de ramificação

O `branch-key-id` que identifica a chave de ramificação ativa em seu armazenamento de chaves de ramificação.

Note

Para inicializar o token de autenticação hierárquico para uso multilocatário, você deve especificar um fornecedor de ID de chave de ramificação em vez de um `branch-key-`

id. Para ter mais informações, consulte [Usar o token de autenticação hierárquico em ambientes multilocatários](#).

- (Opcional) Um cache

Se você quiser personalizar o tipo de cache ou o número de entradas de materiais de chave de ramificação que podem ser armazenadas no cache local, especifique o tipo de cache e a capacidade de entrada ao inicializar o token de autenticação.

O tipo de cache define o modelo de segmentação. O chaveiro hierárquico fornece três tipos de cache que oferecem suporte a ambientes multilocatários: Padrão,,. MultiThreaded StormTracking

Se você não especificar um cache, o token de autenticação hierárquico usará automaticamente o tipo de cache Default e definirá a capacidade de entrada como 1000.

Default (Recommended)

Para a maioria dos usuários, o cache Default atende aos requisitos de segmentação. O cache Default foi projetado para oferecer suporte a ambientes com muitos threads. Quando uma entrada de materiais de chave de filial expira, o cache padrão impede que vários threads sejam chamados AWS KMS e o Amazon DynamoDB notifica um thread de que a entrada de materiais de chave de ramificação expirará com 10 segundos de antecedência. Isso garante que somente um thread envie uma solicitação AWS KMS para atualizar o cache.

Para inicializar o token de autenticação hierárquico com um cache Default, especifique o seguinte valor:

- Capacidade de entrada: limita o número de entradas de materiais de chave da ramificação que podem ser armazenadas no cache local.

C#/.NET

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

Java

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

```
.build())
```

O padrão e StormTracking os caches suportam o mesmo modelo de segmentação, mas você só precisa especificar a capacidade de entrada para inicializar o chaveiro hierárquico com o cache padrão. Para personalizações de cache mais granulares, use o cache. StormTracking

MultiThreaded

O MultiThreaded cache é seguro para uso em ambientes com vários processos, mas não fornece nenhuma funcionalidade para minimizar as chamadas do Amazon AWS KMS DynamoDB. Como resultado, quando uma entrada de materiais de chave de ramificação expirar, todos os tópicos serão notificados ao mesmo tempo. Isso pode resultar em várias chamadas do AWS KMS para atualizar o cache.

Para inicializar seu chaveiro hierárquico com um MultiThreaded cache, especifique os seguintes valores:

- Capacidade de entrada: limita o número de entradas de materiais de chave da ramificação que podem ser armazenadas no cache local.
- Tamanho de entrada de limpeza de tail: define o número de entradas a serem limpas se a capacidade de entrada for atingida.

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

O StormTracking cache foi projetado para suportar ambientes altamente multisegmentados. Quando uma entrada de materiais de chave de filial expira, o StormTracking cache impede que vários threads sejam chamados AWS KMS e o Amazon DynamoDB notifica um thread de que a entrada de materiais de chave de ramificação expirará com antecedência. Isso garante que somente um thread envie uma solicitação AWS KMS para atualizar o cache.

Para inicializar seu chaveiro hierárquico com um StormTracking cache, especifique os seguintes valores:

- Capacidade de entrada: limita o número de entradas de materiais de chave da ramificação que podem ser armazenadas no cache local.
- Tamanho de entrada de limpeza de tail: define o número de entradas de materiais de chave da ramificação a serem limpas por vez.

Valor padrão: 1 entrada

- Período de carência: define o número de segundos antes da expiração em que é feita uma tentativa de atualizar os materiais de chave da ramificação.

Valor padrão: 10 segundos

- Intervalo de carência: define o número de segundos entre as tentativas de atualizar os materiais de chave da ramificação.

Valor padrão: 1 segundo

- Fan out: define o número de tentativas simultâneas que podem ser feitas para atualizar os materiais de chave da ramificação.

Valor padrão: 20 tentativas

- Tempo de ativação (TTL) em trânsito: define o número de segundos até que uma tentativa de atualizar os materiais de chave de ramificação atinja o tempo limite. Sempre que o cache retorna `NoSuchEntry` em resposta a `GetCacheEntry`, essa chave de ramificação é considerada em trânsito até que a mesma chave seja gravada com uma entrada `PutCache`.

Valor padrão: 20 segundos

- Sleep: define o número de segundos que um thread deve ficar em repouso se `fanOut` for excedido.

Valor padrão: 20 milissegundos

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())
```

- (Opcional) Uma lista de Tokens de Concessão

Se você controlar o acesso à chave do KMS no token de autenticação hierárquico com [concessões](#), deverá fornecer todos os tokens de concessão necessários ao inicializar o token de autenticação.

O exemplo a seguir inicializa um chaveiro hierárquico com um limite de cache TLL de 600 segundos e uma capacidade de entrada de 1000.

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);
```

Alternar a chave de ramificação ativa

Só pode haver uma versão ativa para cada chave de ramificação por vez. O token de autenticação hierárquico normalmente usa cada versão ativa da chave de ramificação para atender a várias solicitações. Porém, você controla até que ponto as chaves de ramificação ativas são reutilizadas e determina com que frequência a chave de ramificação ativa é alternada.

As chaves de ramificação não são usadas para criptografar chaves de dados em texto simples. Eles são usados para derivar as chaves de empacotamento exclusivas que criptografam chaves de dados de texto simples. O [processo de derivação da chave de empacotamento](#) produz uma chave de empacotamento exclusiva de 32 bytes com 28 bytes de randomização. Isso significa que uma chave de ramificação pode derivar mais de 79 octilhões, ou 2^{96} , chaves de empacotamento exclusivas antes que ocorra o desgaste criptográfico. Apesar desse risco de exaustão muito baixo, talvez seja necessário alternar suas chaves de ramificações ativas devido a regras comerciais ou contratuais ou regulamentações governamentais.

A versão ativa da chave de ramificação permanece ativa até que você a altere. As versões anteriores da chave de ramificação ativa não serão usadas para realizar operações de criptografia e não podem ser usadas para derivar novas chaves de empacotamento. Porém, eles ainda podem ser consultados e fornecer chaves de empacotamento para descriptografar as chaves de dados que eles criptografaram enquanto estavam ativos.

Use a operação `VersionKey` do serviço de armazenamento de chaves para alternar sua chave de ramificação ativa. Quando você altera a chave de ramificação ativa, uma nova chave de ramificação é criada para substituir a versão anterior. O `branch-key-id` não muda quando você altera a chave de ramificação ativa. Você deve especificar o `branch-key-id` que identificará a chave de ramificação ativa atual quando você chamar `VersionKey`.

C# / .NET

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

Java

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

Usar o token de autenticação hierárquico em ambientes multilocatários

É possível usar a hierarquia de chaves estabelecida entre as chaves de ramificação ativas e as chaves de encapsulamento derivadas para oferecer suporte a ambientes multilocatários criando uma chave de ramificação para cada locatário em seu banco de dados. O token de autenticação

hierárquico criptografa todos os dados de um determinado locatário com sua chave de ramificação distinta. Assim você pode isolar os dados do locatário por chave de ramificação.

Cada locatário tem sua própria chave de ramificação, definida por uma `branch-key-id` exclusiva. Só pode haver uma versão ativa para cada `branch-key-id` por vez.

Antes de inicializar seu token de autenticação hierárquico para uso multilocatário, você deve criar uma chave de ramificação para cada locatário e criar um fornecedor de ID de chave de ramificação. Use o fornecedor da ID da chave da ramificação para criar um nome amigável para o `branch-key-ids` para facilitar o reconhecimento do `branch-key-id` correto para um locatário. Por exemplo, o nome amigável permite que você se refira a uma chave de ramificação como `tenant1` em vez de `b3f61619-4d35-48ad-a275-050f87e15122`.

Para operações de descriptografia, você pode configurar estaticamente um único token de autenticação hierárquico para restringir a descriptografia a um único locatário ou usar o fornecedor da ID da chave da ramificação para identificar qual locatário é responsável por descriptografar uma mensagem.

Primeiro, siga as etapas 1 e 2 dos procedimentos de [pré-requisitos](#). Depois, use os procedimentos a seguir para criar uma chave de ramificação para cada locatário, criar um fornecedor de ID de chave de ramificação e inicializar seu token de autenticação hierárquico para uso multilocatário.

Etapas 1: crie uma chave de ramificação para cada locatário em seu ambiente

Chame `CreateKey` para cada locatário.

A operação a seguir cria duas chaves de ramificação usando a chave KMS que você especificou ao criar seu serviço de armazenamento de chaves e adiciona as chaves de ramificação à tabela do DynamoDB que você criou para servir como seu armazenamento de chaves de ramificação. A mesma chave do KMS deve proteger todas as chaves de ramificação.

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());
```

Etapa 2: crie um fornecedor de ID de chave de ramificação

O exemplo a seguir cria um fornecedor de ID de chave de filial.

C# / .NET

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

Java

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

Etapa 3: inicie seu token de autenticação hierárquico com o fornecedor de ID da chave de ramificação

Para inicializar o token de autenticação hierárquico, você deve fornecer os seguintes valores:

- Um nome de armazenamento de chave de ramificação
- Um [tempo de vida do cache \(TTL\)](#)
- Um fornecedor de ID de chave de ramificação
- (Opcional) Um cache

Se você quiser personalizar o tipo de cache ou o número de entradas de materiais de chave de ramificação que podem ser armazenadas no cache local, especifique o tipo de cache e a capacidade de entrada ao inicializar o token de autenticação.

O tipo de cache define o modelo de segmentação. O chaveiro hierárquico fornece três tipos de cache que oferecem suporte a ambientes multilocatários: Padrão,, MultiThreaded StormTracking

Se você não especificar um cache, o token de autenticação hierárquico usará automaticamente o tipo de cache Default e definirá a capacidade de entrada como 1000.

Default (Recommended)

Para a maioria dos usuários, o cache Default atende aos requisitos de segmentação. O cache Default foi projetado para oferecer suporte a ambientes com muitos threads. Quando uma entrada de materiais de chave de filial expira, o cache padrão impede que vários threads sejam chamados AWS KMS e o Amazon DynamoDB notifica um thread de que a entrada de materiais de chave de ramificação expirará com 10 segundos de antecedência. Isso garante que somente um thread envie uma solicitação AWS KMS para atualizar o cache.

Para inicializar o token de autenticação hierárquico com um cache Default, especifique o seguinte valor:

- Capacidade de entrada: limita o número de entradas de materiais de chave da ramificação que podem ser armazenadas no cache local.

C# /.NET

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

Java

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

O padrão e StormTracking os caches suportam o mesmo modelo de segmentação, mas você só precisa especificar a capacidade de entrada para inicializar o chaveiro hierárquico com o cache padrão. Para personalizações de cache mais granulares, use o cache.

StormTracking

MultiThreaded

O MultiThreaded cache é seguro para uso em ambientes com vários processos, mas não fornece nenhuma funcionalidade para minimizar as chamadas do Amazon AWS KMS DynamoDB. Como resultado, quando uma entrada de materiais de chave de ramificação

expirar, todos os tópicos serão notificados ao mesmo tempo. Isso pode resultar em várias AWS KMS chamadas para atualizar o cache.

Para inicializar seu chaveiro hierárquico com um MultiThreaded cache, especifique os seguintes valores:

- Capacidade de entrada: limita o número de entradas de materiais de chave da ramificação que podem ser armazenadas no cache local.
- Tamanho de entrada de limpeza de tail: define o número de entradas a serem limpas se a capacidade de entrada for atingida.

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

O StormTracking cache foi projetado para suportar ambientes altamente multisegmentados. Quando uma entrada de materiais de chave de filial expira, o StormTracking cache impede que vários threads sejam chamados AWS KMS e o Amazon DynamoDB notifica um thread de que a entrada de materiais de chave de ramificação expirará com antecedência. Isso garante que somente um thread envie uma solicitação AWS KMS para atualizar o cache.

Para inicializar seu chaveiro hierárquico com um StormTracking cache, especifique os seguintes valores:

- Capacidade de entrada: limita o número de entradas de materiais de chave da ramificação que podem ser armazenadas no cache local.
- Tamanho de entrada de limpeza de tail: define o número de entradas de materiais de chave da ramificação a serem limpas por vez.

Valor padrão: 1 entrada

- Período de carência: define o número de segundos antes da expiração em que é feita uma tentativa de atualizar os materiais de chave da ramificação.

Valor padrão: 10 segundos

- Intervalo de carência: define o número de segundos entre as tentativas de atualizar os materiais de chave da ramificação.

Valor padrão: 1 segundo

- Fan out: define o número de tentativas simultâneas que podem ser feitas para atualizar os materiais de chave da ramificação.

Valor padrão: 20 tentativas

- Tempo de ativação (TTL) em trânsito: define o número de segundos até que uma tentativa de atualizar os materiais de chave de ramificação atinja o tempo limite. Sempre que o cache retorna `NoSuchEntry` em resposta a `GetCacheEntry`, essa chave de ramificação é considerada em trânsito até que a mesma chave seja gravada com uma entrada `PutCache`.

Valor padrão: 20 segundos

- Sleep: define o número de segundos que um thread deve ficar em repouso se `fanOut` for excedido.

Valor padrão: 20 milissegundos

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())

```

- (Opcional) Uma lista de Tokens de Concessão

Se você controlar o acesso à chave do KMS no token de autenticação hierárquico com [concessões](#), deverá fornecer todos os tokens de concessão necessários ao inicializar o token de autenticação.

O exemplo a seguir inicializa um chaveiro hierárquico com o fornecedor de ID de chave de filial criado na Etapa 2, um limite de cache TLL de 600 segundos e uma capacidade de entrada de 1000.

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

Etapa 4: criar nomes fáceis para cada chave de ramificação

O exemplo a seguir cria nomes amigáveis para as duas chaves de ramificação criadas na Etapa 1. O AWS Encryption SDK usa contextos de criptografia para mapear o nome fácil que você define para o `branch-key-id` associado.

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 chaveiros ECDH

Important

O AWS KMS chaveiro ECDH só está disponível na versão 3. x do AWS Encryption SDK for Java. O chaveiro AWS KMS ECDH foi introduzido na versão 1.5.0 da Material Providers Library.

Um chaveiro AWS KMS ECDH usa um acordo de chave assimétrica [AWS KMS keys](#) para derivar uma chave de embalagem simétrica compartilhada entre duas partes. Primeiro, o chaveiro usa o algoritmo de acordo de chaves Elliptic Curve Diffie-Hellman (ECDH) para derivar um segredo compartilhado da chave privada no par de chaves KMS do remetente e da chave pública do destinatário. Em seguida, o chaveiro usa o segredo compartilhado para derivar a chave de empacotamento compartilhada que protege suas chaves de criptografia de dados. A função de derivação de chave que o AWS Encryption SDK usa (KDF_CTR_HMAC_SHA384) para derivar a chave de empacotamento compartilhada está em conformidade com as recomendações do [NIST](#) para derivação de chaves.

A função de derivação de chave retorna 64 bytes de material de chaveamento. Para garantir que ambas as partes usem o material de codificação correto, AWS Encryption SDK usam os primeiros 32 bytes como chave de compromisso e os últimos 32 bytes como chave de empacotamento compartilhada. Na descriptografia, se o chaveiro não puder reproduzir a mesma chave de compromisso e chave de encapsulamento compartilhada armazenadas no texto cifrado do cabeçalho da mensagem, a operação falhará. Por exemplo, se você criptografar dados com um chaveiro configurado com a chave privada de Alice e a chave pública de Bob, um chaveiro configurado com a chave privada de Bob e a chave pública de Alice reproduzirá a mesma chave de compromisso e chave de empacotamento compartilhada e poderá descriptografar os dados. Se a chave pública de Bob não for de um par de chaves KMS, Bob poderá criar um [chaveiro ECDH bruto](#) para descriptografar os dados.

O chaveiro AWS KMS ECDH criptografa os dados com uma chave simétrica usando o AES-GCM. A chave de dados é então criptografada em envelope com a chave de empacotamento compartilhada derivada usando o AES-GCM. [Cada chaveiro AWS KMS ECDH pode ter apenas uma chave de embrulho compartilhada, mas você pode incluir vários chaveiros AWS KMS ECDH, sozinhos ou com outros chaveiros, em um chaveiro múltiplo.](#)

Tópicos

- [Permissões necessárias para AWS KMS chaveiros ECDH](#)
- [Criando um AWS KMS chaveiro ECDH](#)
- [Criando um AWS KMS chaveiro de descoberta ECDH](#)

Permissões necessárias para AWS KMS chaveiros ECDH

AWS Encryption SDK Não requer uma AWS conta e não depende de nenhum AWS serviço. No entanto, para usar um chaveiro AWS KMS ECDH, você precisa de uma AWS conta e das seguintes permissões mínimas AWS KMS keys no seu chaveiro. As permissões variam de acordo com o esquema de contrato de chaves que você usa.

- Para criptografar e descriptografar dados usando o esquema de contrato de *KmsPrivateKeyToStaticPublicKey* chave, você precisa de [kms: GetPublicKey e kms: DeriveSharedSecret no par de chaves KMS assimétrico](#) do remetente. Se você fornecer diretamente a chave pública codificada em DER do remetente ao instanciar seu chaveiro, precisará apenas da *DeriveSharedSecret* permissão [kms: no par de chaves KMS assimétrico](#) do remetente.
- Para descriptografar dados usando o esquema de contrato de *KmsPublicKeyDiscovery* chaves, você precisa das *GetPublicKey* permissões [kms: DeriveSharedSecret e kms: no par de chaves assimétrico KMS](#) especificado.

Criando um AWS KMS chaveiro ECDH

Para criar um chaveiro AWS KMS ECDH que criptografe e descriptografe dados, você deve usar o esquema de contrato de chave. *KmsPrivateKeyToStaticPublicKey* Para inicializar um chaveiro AWS KMS ECDH com o esquema de contrato de *KmsPrivateKeyToStaticPublicKey* chaves, forneça os seguintes valores:

- ID do remetente AWS KMS key

Deve identificar um par de chaves KMS de curva elíptica (ECC) assimétrica recomendado pelo NIST com um valor de `KeyUsage KEY_AGREEMENT`. A chave privada do remetente é usada para derivar o segredo compartilhado.

- (Opcional) Chave pública do remetente

[Deve ser uma chave pública X.509 codificada por DER, também conhecida como SubjectPublicKeyInfo \(SPKI\), conforme definido na RFC 5280.](#)

A AWS KMS [GetPublicKey](#) operação retorna a chave pública de um par de chaves KMS assimétrico no formato codificado em DER exigido.

Para reduzir o número de AWS KMS chamadas que seu chaveiro faz, você pode fornecer diretamente a chave pública do remetente. Se nenhum valor for fornecido para a chave pública do remetente, o chaveiro liga AWS KMS para recuperar a chave pública do remetente.

- Chave pública do destinatário

[Você deve fornecer a chave pública X.509 codificada em DER do destinatário, também conhecida como SubjectPublicKeyInfo \(SPKI\), conforme definido na RFC 5280.](#)

A AWS KMS [GetPublicKey](#) operação retorna a chave pública de um par de chaves KMS assimétrico no formato codificado em DER exigido.

- Especificação da curva

Identifica a especificação da curva elíptica nos pares de chaves especificados. Os pares de chaves do remetente e do destinatário devem ter a mesma especificação de curva.

Valores válidos: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

- (Opcional) Uma lista de Tokens de Concessão

Se você controlar o acesso à chave KMS em seu chaveiro AWS KMS ECDH com [concessões](#), deverá fornecer todos os tokens de concessão necessários ao inicializar o chaveiro.

Java

O exemplo a seguir cria um chaveiro AWS KMS ECDH com a chave KMS do remetente, a chave pública do remetente e a chave pública do destinatário. Este exemplo usa o `senderPublicKey` parâmetro opcional para fornecer a chave pública do remetente. Se você não fornecer a chave

pública do remetente, o chaveiro liga AWS KMS para recuperar a chave pública do remetente. Os pares de chaves do remetente e do destinatário estão na `ECC_NIST_P256` curva.

```
// 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();
```

Criando um AWS KMS chaveiro de descoberta ECDH

Ao descriptografar, é uma prática recomendada especificar as chaves que eles podem usar. AWS Encryption SDK Para seguir essa prática recomendada, use um chaveiro AWS KMS ECDH com o esquema de contrato de `KmsPrivateKeyToStaticPublicKey` chaves. No entanto, você também pode criar um chaveiro de descoberta AWS KMS ECDH, ou seja, um chaveiro AWS KMS ECDH que pode descriptografar qualquer mensagem em que a chave pública do par de chaves KMS especificado corresponda à chave pública do destinatário armazenada no texto cifrado da mensagem.

⚠ Important

Ao descriptografar mensagens usando o esquema de contrato de `KmsPublicKeyDiscovery` chave, você aceita todas as chaves públicas, independentemente de quem as possua.

Para inicializar um chaveiro AWS KMS ECDH com o esquema de contrato de `KmsPublicKeyDiscovery` chaves, forneça os seguintes valores:

- AWS KMS key ID do destinatário

Deve identificar um par de chaves KMS de curva elíptica (ECC) assimétrica recomendado pelo NIST com um valor de `KeyUsage KEY_AGREEMENT`

- Especificação da curva

Identifica a especificação da curva elíptica no par de chaves KMS do destinatário.

Valores válidos: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

- (Opcional) Uma lista de Tokens de Concessão

Se você controlar o acesso à chave KMS em seu chaveiro AWS KMS ECDH com [concessões](#), deverá fornecer todos os tokens de concessão necessários ao inicializar o chaveiro.

Java

O exemplo a seguir cria um chaveiro de descoberta AWS KMS ECDH com um par de chaves KMS na curva `ECC_NIST_P256`. Você deve ter as `DeriveSharedSecret` permissões [kms: GetPublicKey](#) e [kms:](#) no par de chaves KMS especificado. Esse chaveiro pode descriptografar qualquer mensagem em que a chave pública do par de chaves KMS especificado corresponda à chave pública do destinatário armazenada no texto cifrado da mensagem.

```
// 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();
```

Tokens de autenticação AES Raw

O AWS Encryption SDK permite que você use uma chave simétrica AES que você fornece como uma chave de empacotamento que protege sua chave de dados. Você precisa gerar, armazenar e proteger o material de chaves, de preferência em um módulo de segurança de hardware (HSM) ou em um sistema de gerenciamento de chaves. Use um token de autenticação AES bruto quando precisar fornecer a chave de empacotamento e criptografar as chaves de dados local ou offline.

O token de autenticação bruto do AES usa o algoritmo AES-GCM e uma chave de empacotamento que você especifica como uma matriz de bytes para criptografar chaves de dados. É possível especificar somente uma chave de encapsulamento em cada token de autenticação bruto do AES, mas você pode incluir vários tokens de autenticação brutos do AES, sozinhos ou com outros tokens de autenticação, em um [multitoken de autenticação](#).

O chaveiro AES bruto é equivalente e interoperava com a [JceMasterKey](#) classe no AWS Encryption SDK for Java e com a [RawMasterKey](#) classe no AWS Encryption SDK for Python quando são usados com chaves de criptografia AES. Você pode criptografar dados com uma implementação e descriptografá-los com qualquer outra implementação usando a mesma chave de encapsulamento. Para obter detalhes, consulte [Compatibilidade dos tokens de autenticação](#).

Nomes e namespaces de chaves

Para identificar a chave AES em um token de autenticação, o token de autenticação bruto do AES usa um namespace de chave e um nome de chave fornecidos por você. Esses valores não são secretos. Eles aparecem em texto simples no cabeçalho da [mensagem criptografada](#) que a operação de criptografia retorna. Recomendamos usar um namespace de chave em seu HSM ou sistema de gerenciamento de chaves e um nome de chave que identifique a chave AES nesse sistema.

Note

O namespace da chave e o nome da chave são equivalentes aos campos ID do provedor (ou provedor) e ID da chave no `JceMasterKey` e no `RawMasterKey`.

O AWS Encryption SDK for C e AWS Encryption SDK para.NET reserva o valor do namespace `aws-kms` chave para as chaves KMS. Não use esse valor de namespace em um token de autenticação AES bruto ou um token de autenticação RSA bruto com essas bibliotecas.

Se você cria tokens de autenticação diferentes para criptografar e descriptografar determinada mensagem, o namespace e os valores do nome são cruciais. Se o namespace e o nome da chave no token de autenticação de decodificação não corresponderem exatamente e com distinção entre maiúsculas e minúsculas ao namespace e ao nome da chave no token de autenticação de criptografia, o token de autenticação de decodificação não será usado, mesmo que os bytes do material da chave sejam idênticos.

Por exemplo, é possível definir um token de autenticação AES bruto com namespace `HSM_01` e nome de chave `AES_256_012`. Em seguida, você usa esse token de autenticação para criptografar alguns dados. Para descriptografar esses dados, construa um token de autenticação bruto do AES bruto com o mesmo namespace de chave, nome de chave e material de chave.

O exemplo a seguir mostra como criar um token de autenticação bruto do AES. A variável `AESWrappingKey` representa o material principal que você fornece.

C

Para instanciar um chaveiro AES bruto no AWS Encryption SDK for C, use.

`aws_cryptosdk_raw_aes_keyring_new()` Para obter um exemplo completo, consulte [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, aws_wrapping_key,
    wrapping_key_len);
```

C# / .NET

Para criar um chaveiro AES bruto AWS Encryption SDK para o .NET, use o `materialProviders.CreateRawAesKeyring()` método. Para ver um exemplo completo, consulte [raWAES.cs.KeyringExample](#)

O exemplo a seguir usa a versão 4.x do AWS Encryption SDK para .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

O AWS Encryption SDK para JavaScript no navegador obtém suas primitivas criptográficas da [WebCryptoAPI](#). Antes de construir o chaveiro, você deve usá-lo `RawAesKeyringWebCrypto.importCryptoKey()` para importar o material bruto da chave para o WebCrypto backend. Isso garante que o chaveiro esteja completo, mesmo que todas as chamadas sejam WebCrypto assíncronas.

Em seguida, para instanciar um token de autenticação AES bruto, use o método `RawAesKeyringWebCrypto()`. Você deve especificar o algoritmo de encapsulamento AES

(“pacote de encapsulamento”) com base no tamanho do seu material de chave. Para obter um exemplo completo, consulte [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

Para instanciar um chaveiro AES bruto no AWS Encryption SDK para JavaScript for Node.js, crie uma instância da classe `RawAesKeyringNode`. Você deve especificar o algoritmo de encapsulamento AES (“pacote de encapsulamento”) com base no tamanho do seu material de chave. Para obter um exemplo completo, consulte [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

Para instanciar um chaveiro AES bruto no AWS Encryption SDK for Java, use.

```
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);
```

Tokens de autenticação brutos do RSA

O token de autenticação bruto do RSA realiza a criptografia e a descryptografia assimétricas das chaves de dados na memória local com chaves de encapsulamento pública e privada fornecidas. Você precisa gerar, armazenar e proteger a chave privada, de preferência em um módulo de segurança de hardware (HSM) ou com o sistema de gerenciamento de chaves. A função de criptografia criptografa a chave de dados com chave pública do RSA. A função de descryptografia descryptografa a chave de dados usando a chave privada. Você pode selecionar entre os vários [modos de padding do RSA](#).

Um token de autenticação bruto do RSA que criptografa e descryptografa deve incluir uma chave pública e um par de chaves privadas assimétricas. No entanto, é possível criptografar dados com um token de autenticação bruto do RSA que tenha apenas uma chave pública e descryptografar dados com um token de autenticação bruto do RSA que tenha apenas uma chave privada. É possível incluir qualquer token de autenticação bruto do RSA em um [multitoken de autenticação](#). Se você configurar um token de autenticação bruto do RSA com uma chave pública e privada, certifique-se de que eles façam parte do mesmo par de chaves. Algumas implementações de linguagem do não AWS Encryption SDK construirão um chaveiro RSA bruto com chaves de pares diferentes. Outras pessoas confiam em você para verificar se suas chaves são do mesmo par de chaves.

O chaveiro RSA bruto é equivalente e interopera com o [JceMasterKey](#) in the AWS Encryption SDK for Java e o [RawMasterKey](#) in the AWS Encryption SDK for Python quando são usados com chaves de criptografia assimétrica RSA. Você pode criptografar dados com uma implementação e

descriptografá-los com qualquer outra implementação usando a mesma chave de encapsulamento. Para obter detalhes, consulte [Compatibilidade dos tokens de autenticação](#).

Note

O token de autenticação bruto do RSA não oferece suporte a chaves assimétricas do KMS. Se você quiser usar chaves RSA KMS assimétricas, versão 4. x do AWS Encryption SDK para .NET e versão 3. x dos AWS KMS chaveiros de AWS Encryption SDK for Java suporte que usam criptografia simétrica (SYMMETRIC_DEFAULT) ou RSA assimétrica. AWS KMS keys

Se você criptografar dados com um chaveiro RSA bruto que inclua a chave pública de uma chave RSA KMS, nem o AWS Encryption SDK nem poderá descriptografá-lo. AWS KMS Você não pode exportar a chave privada de uma chave KMS AWS KMS assimétrica para um chaveiro RSA bruto. [A operação de AWS KMS descriptografia não pode descriptografar a mensagem criptografada retornada.](#) AWS Encryption SDK

Ao criar um chaveiro RSA bruto no AWS Encryption SDK for C, certifique-se de fornecer o conteúdo do arquivo PEM que inclui cada chave como uma string C terminada em nulo, não como um caminho ou nome de arquivo. Ao criar um chaveiro RSA bruto JavaScript, esteja ciente da [possível incompatibilidade com outras](#) implementações de linguagem.

Namespaces e nomes

Para identificar a chave RSA em um token de autenticação, o token de autenticação bruto do RSA usa um namespace de chave e um nome de chave fornecidos por você. Esses valores não são secretos. Eles aparecem em texto simples no cabeçalho da [mensagem criptografada](#) que a operação de criptografia retorna. Recomendamos usar um namespace de chave e um nome de chave que identifique o par de chaves RSA (ou a sua chave privada) no HSM ou no sistema de gerenciamento de chaves.

Note

O namespace da chave e o nome da chave são equivalentes aos campos ID do provedor (ou provedor) e ID da chave no `JceMasterKey` e no `RawMasterKey`.

O AWS Encryption SDK for C reserva o valor do namespace da `aws-kms` chave para as chaves KMS. Não o use em um token de autenticação bruto do AES ou em um token de autenticação bruto do RSA com o AWS Encryption SDK for C.

Se você cria tokens de autenticação diferentes para criptografar e descriptografar determinada mensagem, o namespace e os valores do nome são cruciais. Se o namespace e o nome da chave no token de autenticação de descriptografia não corresponderem exatamente e com distinção entre maiúsculas e minúsculas ao namespace e ao nome da chave no token de autenticação de criptografia, o token de autenticação de descriptografia não será usado, mesmo que as chaves sejam do mesmo par de chaves.

O namespace da chave e o nome da chave do material da chave nos tokens de autenticação de criptografia e decodificação devem ser os mesmos, independentemente de o token de autenticação conter a chave pública RSA, a chave privada RSA ou ambas as chaves no par de chaves. Por exemplo, suponha que você criptografe dados com um token de autenticação RSA bruto para uma chave pública RSA com o namespace de chave HSM_01 e nome de chave RSA_2048_06. Para descriptografar esses dados, construa um token de autenticação RSA bruto com a chave privada (ou par de chaves) e o mesmo namespace e nome de chave.

Modo de preenchimento

Você deve especificar um modo de preenchimento para tokens de autenticação RSA brutos usados para criptografia e descriptografia, ou usar atributos de sua implementação de linguagem que o especifiquem para você.

O AWS Encryption SDK suporta os seguintes modos de preenchimento, sujeitos às restrições de cada idioma. Recomendamos um modo de preenchimento [OAEP](#), particularmente OAEP com SHA-256 e MGF1 com preenchimento SHA-256. O modo de preenchimento [PKCS1](#) é compatível somente para compatibilidade com versões anteriores.

- OAEP com SHA-1 e MGF1 com preenchimento SHA-1
- OAEP com SHA-256 e MGF1 com preenchimento SHA-256
- OAEP com SHA-384 e MGF1 com preenchimento SHA-384
- OAEP com SHA-512 e MGF1 com preenchimento SHA-512
- Preenchimento PKCS1 v1.5

Os exemplos a seguir mostram como criar um token de autenticação bruto do RSA com a chave pública e privada de um par de chaves RSA e o OAEP com SHA-256 e MGF1 com modo de preenchimento SHA-256. As variáveis `RSAPublicKey` e `RSAPrivateKey` representam o material principal fornecido por você.

C

Para criar um chaveiro RSA bruto no AWS Encryption SDK for C, use.

```
aws_cryptosdk_raw_rsa_keyring_new
```

Ao criar um chaveiro RSA bruto no AWS Encryption SDK for C, certifique-se de fornecer o conteúdo do arquivo PEM que inclui cada chave como uma string C terminada em nulo, não como um caminho ou nome de arquivo. Para obter um exemplo completo, consulte [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

Para instanciar um chaveiro RSA bruto no para.NET, AWS Encryption SDK use o método. `materialProviders.CreateRawRsaKeyring()` Para ver um exemplo completo, consulte [rawRSA .cs KeyringExample](#).

O exemplo a seguir usa a versão 4.x do AWS Encryption SDK para .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

O AWS Encryption SDK para JavaScript no navegador obtém suas primitivas criptográficas da [WebCrypto](#) biblioteca. Antes de construir o chaveiro, você deve usar `importPublicKey()` e/ou importar `importPrivateKey()` a matéria-prima da chave para o WebCrypto backend. Isso garante que o chaveiro esteja completo, mesmo que todas as chamadas sejam WebCrypto assíncronas. O objeto usado pelos métodos de importação inclui o algoritmo de encapsulamento e seu modo de preenchimento.

Depois de importar o material da chave, use o método `RawRsaKeyringWebCrypto()` para instanciar o token de autenticação. Ao criar um chaveiro RSA bruto JavaScript, esteja ciente da [possível incompatibilidade com outras](#) implementações de linguagem.

Para obter um exemplo completo, consulte [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

Para instanciar um chaveiro RSA bruto no AWS Encryption SDK para JavaScript Node.js, crie uma nova instância da classe `RawRsaKeyringNode`. O parâmetro `wrapKey` contém a chave pública. O parâmetro `unwrapKey` contém a chave privada. O construtor `RawRsaKeyringNode` calcula um modo de preenchimento padrão, embora você possa especificar um modo de preenchimento preferencial.

Ao criar um chaveiro RSA bruto JavaScript, esteja ciente da [possível incompatibilidade com outras](#) implementações de linguagem.

Para ver um exemplo completo, consulte [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);
```

Chaveiros ECDH brutos

Important

O chaveiro Raw ECDH só está disponível na versão 3. x do AWS Encryption SDK for Java. O chaveiro ECDH bruto foi introduzido na versão 1.5.0 da Material Providers Library.

O chaveiro ECDH bruto usa os pares de chaves públicas-privadas de curva elíptica que você fornece para derivar uma chave de empacotamento compartilhada entre duas partes. Primeiro, o chaveiro obtém um segredo compartilhado usando a chave privada do remetente, a chave pública do destinatário e o algoritmo de acordo de chave Elliptic Curve Diffie-Hellman (ECDH). Em seguida, o chaveiro usa o segredo compartilhado para derivar a chave de empacotamento compartilhada que protege suas chaves de criptografia de dados. A função de derivação de chave que o AWS Encryption SDK usa (KDF_CTR_HMAC_SHA384) para derivar a chave de empacotamento compartilhada está em conformidade com as recomendações do [NIST](#) para derivação de chaves.

A função de derivação de chave retorna 64 bytes de material de chave. Para garantir que ambas as partes usem o material de chave correto, AWS Encryption SDK usam os primeiros 32 bytes como chave de compromisso e os últimos 32 bytes como chave de empacotamento compartilhada. Na descryptografia, se o chaveiro não puder reproduzir a mesma chave de compromisso e chave de encapsulamento compartilhada armazenadas no texto cifrado do cabeçalho da mensagem, a operação falhará. Por exemplo, se você criptografar dados com um chaveiro configurado com a chave privada de Alice e a chave pública de Bob, um chaveiro configurado com a chave privada de Bob e a chave pública de Alice reproduzirá a mesma chave de compromisso e chave de empacotamento compartilhada e poderá descryptografar os dados. Se a chave pública de Bob for de um AWS KMS key par, Bob poderá criar um [chaveiro AWS KMS ECDH](#) para descryptografar os dados.

O chaveiro ECDH bruto criptografa os dados com uma chave simétrica usando o AES-GCM. A chave de dados é então criptografada em envelope com a chave de empacotamento compartilhada derivada usando o AES-GCM. [Cada chaveiro Raw ECDH pode ter apenas uma chave de embrulho compartilhada, mas você pode incluir vários chaveiros Raw ECDH, sozinhos ou com outros chaveiros, em um chaveiro múltiplo.](#)

Você é responsável por gerar, armazenar e proteger suas chaves privadas, preferencialmente em um módulo de segurança de hardware (HSM) ou sistema de gerenciamento de chaves. Os pares

de chaves do remetente e do destinatário devem estar na mesma curva elíptica. O AWS Encryption SDK suporta as seguintes especificações de curva elíptica:

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

Criando um chaveiro ECDH bruto

O chaveiro Raw ECDH suporta três esquemas de contrato principais: `RawPrivateKeyToStaticPublicKey`, e.

`EphemeralPrivateKeyToStaticPublicKey` `PublicKeyDiscovery` O esquema de contrato de chave selecionado determina quais operações criptográficas você pode realizar e como os materiais de chaveamento são montados.

Tópicos

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

RawPrivateKeyToStaticPublicKey

Use o esquema de contrato de `RawPrivateKeyToStaticPublicKey` chave para configurar estaticamente a chave privada do remetente e a chave pública do destinatário no chaveiro. Esse esquema de contrato chave pode criptografar e descriptografar dados.

Para inicializar um chaveiro ECDH bruto com o esquema de contrato de `RawPrivateKeyToStaticPublicKey` chave, forneça os seguintes valores:

- Chave privada do remetente

[Você deve fornecer a chave privada codificada por PEM do remetente \(PrivateKeyInfo estruturas PKCS #8\), conforme definido na RFC 5958.](#)

- Chave pública do destinatário

[Você deve fornecer a chave pública X.509 codificada em DER do destinatário, também conhecida como SubjectPublicKeyInfo \(SPKI\), conforme definido na RFC 5280.](#)

Você pode especificar a chave pública de um contrato de chave assimétrica (par de chaves KMS) ou a chave pública de um par de chaves gerado fora do. AWS

- Especificação da curva

Identifica a especificação da curva elíptica nos pares de chaves especificados. Os pares de chaves do remetente e do destinatário devem ter a mesma especificação de curva.

Valores válidos: ECC_NIST_P256, ECC_NIS_P384, ECC_NIST_P512

Java

O exemplo Java a seguir usa o esquema de contrato de `RawPrivateKeyToStaticPublicKey` chave para configurar estaticamente a chave privada do remetente e a chave pública do destinatário. Ambos os pares de chaves estão na `ECC_NIST_P256` curva.

```
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

Os chaveiros configurados com o esquema de contrato de `EphemeralPrivateKeyToStaticPublicKey` criam um novo par de chaves localmente e derivam uma chave de empacotamento compartilhada exclusiva para cada chamada criptografada.

Esse esquema de contrato de chave só pode criptografar mensagens. Para descriptografar mensagens criptografadas com o esquema de contrato de `EphemeralPrivateKeyToStaticPublicKey` chave, você deve usar um esquema de contrato de chave de descoberta configurado com a mesma chave pública do destinatário. Para descriptografar, você pode usar um chaveiro ECDH bruto com o algoritmo de acordo de chave ou, se a [PublicKeyDiscovery](#) chave pública do destinatário for de um par de chaves KMS de acordo de chave assimétrico, você pode AWS KMS usar um chaveiro ECDH com o esquema de contrato de chave. [KmsPublicKeyDiscovery](#)

Para inicializar um chaveiro ECDH bruto com o esquema de contrato de `EphemeralPrivateKeyToStaticPublicKey` chave, forneça os seguintes valores:

- Chave pública do destinatário

[Você deve fornecer a chave pública X.509 codificada em DER do destinatário, também conhecida como SubjectPublicKeyInfo \(SPKI\), conforme definido na RFC 5280.](#)

Você pode especificar a chave pública de um contrato de chave assimétrica (par de chaves KMS) ou a chave pública de um par de chaves gerado fora do. AWS

- Especificação da curva

Identifica a especificação da curva elíptica na chave pública especificada.

Ao criptografar, o chaveiro cria um novo par de chaves na curva especificada e usa a nova chave privada e a chave pública especificada para derivar uma chave de empacotamento compartilhada.

Valores válidos: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

Java

O exemplo a seguir cria um chaveiro ECDH bruto com o esquema de contrato de `EphemeralPrivateKeyToStaticPublicKey` chaves. Ao criptografar, o chaveiro criará um novo par de chaves localmente na curva especificada `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

Ao descriptografar, é uma prática recomendada especificar as chaves de encapsulamento que podem ser usadas. AWS Encryption SDK Para seguir essa prática recomendada, use um chaveiro ECDH que especifique a chave privada do remetente e a chave pública do destinatário. No entanto, você também pode criar um chaveiro de descoberta de ECDH bruto, ou seja, um chaveiro ECDH bruto que pode descriptografar qualquer mensagem em que a chave pública da chave especificada

corresponda à chave pública do destinatário armazenada no texto cifrado da mensagem. Esse esquema de contrato de chave só pode descriptografar mensagens.

⚠ Important

Ao descriptografar mensagens usando o esquema de contrato de `PublicKeyDiscovery` chave, você aceita todas as chaves públicas, independentemente de quem as possua.

Para inicializar um chaveiro ECDH bruto com o esquema de contrato de `PublicKeyDiscovery` chave, forneça os seguintes valores:

- Chave privada estática do destinatário

[Você deve fornecer a chave privada codificada por PEM do destinatário \(`PrivateKeyInfo` estruturas PKCS #8\), conforme definido na RFC 5958.](#)

- Especificação da curva

Identifica a especificação da curva elíptica na chave privada especificada. Os pares de chaves do remetente e do destinatário devem ter a mesma especificação de curva.

Valores válidos: `ECC_NIST_P256`, `ECC_NIS_P384`, `ECC_NIST_P512`

Java

O exemplo a seguir cria um chaveiro ECDH bruto com o esquema de contrato de `PublicKeyDiscovery` chaves. Esse chaveiro pode descriptografar qualquer mensagem em que a chave pública da chave privada especificada corresponda à chave pública do destinatário armazenada no texto cifrado da mensagem.

```
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();

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

Multitokens de autenticação

É possível combinar tokens de autenticação em um multitoken de autenticação. Um multitoken de autenticação é um token que consiste em um ou mais tokens de autenticação individuais do mesmo ou de outro tipo. O efeito é como se estivesse usando vários tokens de autenticação em uma série. Quando você usa um multitoken de autenticação para criptografar dados, qualquer uma das chaves de empacotamento em qualquer um de seus tokens de autenticação pode descriptografar esses dados.

Ao criar um multitoken de autenticação para criptografar dados, é possível designar um dos tokens de autenticação como o token de autenticação gerador. Todos os outros tokens de autenticação são conhecidos como tokens de autenticação filho. O token de autenticação gerador cria e criptografa a chave de dados em texto simples. Depois, todas as chaves de empacotamento em todos os tokens filho criptografam a mesma chave de dados em texto simples. O multitoken de autenticação retorna a chave em texto simples e uma chave de dados criptografada para cada chave de empacotamento do multitoken de autenticação. Se você criar um multitoken de autenticação sem um token gerador, poderá usá-lo para descriptografar dados, mas não para criptografá-los. Se o token de autenticação gerador for um [token de autenticação do KMS](#), a chave geradora no token de autenticação do AWS KMS gerará e criptografará a chave de texto simples. Em seguida, todas as chaves adicionais

AWS KMS keys no AWS KMS chaveiro e todas as chaves de embrulho em todos os chaveiros secundários do chaveiro múltiplo criptografam a mesma chave de texto sem formatação.

Ao descriptografar, ele AWS Encryption SDK usa os chaveiros para tentar descriptografar uma das chaves de dados criptografadas. Os tokens de autenticação são chamados na ordem em que são especificados no multitoken de autenticação. O processamento para assim que qualquer chave em qualquer token de autenticação pode descriptografar uma chave de dados criptografada.

A partir da [versão 1.7. x](#), [quando uma chave de dados criptografada é criptografada em um chaveiro AWS Key Management Service \(AWS KMS\) \(ou provedor de chave mestra\), AWS Encryption SDK sempre passa o ARN da chave para AWS KMS keyKeyId o parâmetro da AWS KMS operação Decrypt](#). Essa é uma prática AWS KMS recomendada que garante que você decodifique a chave de dados criptografada com a chave de empacotamento que você pretende usar.

Para ver um exemplo prático de um multitoken de autenticação, consulte:

- C: [multi_keyring.cpp](#)
- C# /.NET: [.cs MultiKeyringExample](#)
- JavaScript Node.js: [multi_keyring.ts](#)
- JavaScript Navegador: [multi_keyring.ts](#)
- Java: [MultiKeyringExample.java](#)

Para criar um multitoken de autenticação, primeiro instancie os tokens de autenticação filho. Neste exemplo, usamos um AWS KMS chaveiro e um chaveiro AES bruto, mas você pode combinar qualquer chaveiro compatível em um chaveiro múltiplo.

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);
```

Em seguida, crie o multitoken de autenticação e especifique seu token gerador, se houver. Neste exemplo, criamos um chaveiro múltiplo no qual o chaveiro é o AWS KMS chaveiro do gerador e o chaveiro AES é o chaveiro infantil.

C

No construtor de multitoken de autenticação no C, você especifica apenas seu token de autenticação gerador.

```
struct aws_cryptosdk_keyring *multi_keyring = aws_cryptosdk_multi_keyring_new(alloc,
    kms_keyring);
```

Para adicionar um token de autenticação filho ao multitoken de autenticação, use o método `aws_cryptosdk_multi_keyring_add_child`. Você precisa chamar o método uma vez para cada token de autenticação filho que adicionar.

```
// Add the Raw AES keyring (C only)
aws_cryptosdk_multi_keyring_add_child(multi_keyring, aes_keyring);
```

C# / .NET

O construtor.NET `CreateMultiKeyringInput` permite definir um token de autenticação gerador e tokens de autenticação secundários. O objeto `CreateMultiKeyringInput` resultante é imutável.

```
var createMultiKeyringInput = new CreateMultiKeyringInput
{
    Generator = kmsKeyring,
    ChildKeyrings = new List<IKeyring>() {aesKeyring}
};
```

```
var multiKeyring = materialProviders.CreateMultiKeyring(createMultiKeyringInput);
```

JavaScript Browser

JavaScript vários chaveiros são imutáveis. O construtor JavaScript de vários chaveiros permite que você especifique o chaveiro do gerador e vários chaveiros infantis.

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

const multiKeyring = new MultiKeyringWebCrypto(generator: kmsKeyring, children:
  [aesKeyring]);
```

JavaScript Node.js

JavaScript vários chaveiros são imutáveis. O construtor JavaScript de vários chaveiros permite que você especifique o chaveiro do gerador e vários chaveiros infantis.

```
const multiKeyring = new MultiKeyringNode(generator: kmsKeyring, children:
  [aesKeyring]);
```

Java

O `CreateMultiKeyringInput` construtor Java permite definir um gerador de chaveiros e um chaveiro secundário. O objeto `createMultiKeyringInput` resultante é imutável.

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

Agora, é possível usar o multitoken de autenticação para criptografar e descriptografar dados.

Linguagens de programação do AWS Encryption SDK

O AWS Encryption SDK está disponível para as linguagens de programação a seguir. As implementações de linguagem são interoperáveis. É possível criptografar com uma implementação de linguagem e descriptografar com outra. A interoperabilidade pode estar sujeita às restrições de linguagem. Em caso afirmativo, essas restrições estarão descritas no tópico sobre a implementação de linguagem. Além disso, ao criptografar e descriptografar, é necessário usar tokens de autenticação compatíveis ou chaves mestras e provedores de chaves mestras. Para obter mais detalhes, consulte [the section called “Compatibilidade dos tokens de autenticação”](#).

Tópicos

- [AWS Encryption SDK for C](#)
- [AWS Encryption SDK para .NET](#)
- [AWS Encryption SDK for Java](#)
- [AWS Encryption SDK para JavaScript](#)
- [AWS Encryption SDK for Python](#)
- [Interface da linha de comando do AWS Encryption SDK](#)

AWS Encryption SDK for C

O AWS Encryption SDK for C fornece uma biblioteca de criptografia do lado do cliente para desenvolvedores que estão criando aplicativos em C. Ele também serve como uma base para implementações do AWS Encryption SDK em linguagens de programação de nível mais elevado.

Como todas as implementações do AWS Encryption SDK, a AWS Encryption SDK for C oferece atributos avançados de proteção de dados. Os recursos incluem [criptografia de envelope](#), AAD (additional authenticated data - dados autenticados adicionais) e [pacotes de algoritmos](#) de chave simétrica segura e autenticada, como o AES-GCM de 256 bits com derivação de chave e assinatura.

Todas as implementações específicas a linguagens do AWS Encryption SDK são totalmente interoperáveis. Por exemplo, você pode criptografar dados com o AWS Encryption SDK for C e descriptografá-los com [qualquer implementação de linguagem compatível](#), incluindo a [CLI de criptografia da AWS](#).

O AWS Encryption SDK for C requer que o AWS SDK for C++ interaja com o AWS Key Management Service (AWS KMS). Você precisa usá-lo somente se estiver usando o [AWS KMStoken de](#)

[autenticação](#) opcional. No entanto, o AWS Encryption SDK não requer o AWS KMS nem qualquer outro serviço da AWS.

Saiba mais

- Para obter detalhes sobre programação com o AWS Encryption SDK for C, consulte os [exemplos de C](#), os [exemplos](#) no [repositório aws-encryption-sdk-c repository](#) no GitHub e a [documentação da API do AWS Encryption SDK for C](#).
- Para obter uma discussão sobre como usar o AWS Encryption SDK for C para criptografar dados para que você possa descriptografar em várias Regiões da AWS, consulte [Como descriptografar textos cifrados em várias regiões com o AWS Encryption SDK em C](#) no Blog de segurança da AWS.

Tópicos

- [Instalar o AWS Encryption SDK for C](#)
- [Usar a AWS Encryption SDK for C](#)
- [Exemplos do AWS Encryption SDK for C](#)

Instalar o AWS Encryption SDK for C

Instalar a versão mais recente do AWS Encryption SDK for C.

Note

Todas as versões do AWS Encryption SDK for C anteriores à 2.0.0 estão na fase de [fim do suporte](#).

Você pode atualizar com segurança a partir da versão 2.0.x e posteriores até a versão mais recente do AWS Encryption SDK for C sem realizar alterações no código ou nos dados. No entanto, os [novos atributos de segurança](#) introduzidos na versão 2.0.x não são compatíveis com versões anteriores. Para atualizar a partir de versões anteriores à 1.7.x até a versão 2.0.x e posteriores, primeiro será necessário atualizar para a versão 1.x mais recente do AWS Encryption SDK for C. Para obter mais detalhes, consulte [Como migrar seu AWS Encryption SDK](#).

Você pode encontrar instruções detalhadas para instalar e criar o AWS Encryption SDK for C no [arquivo README do repositório aws-encryption-sdk-c](#). Ele inclui instruções para criar nas plataformas Amazon Linux, Ubuntu, macOS e Windows.

Antes de começar, decida se deseja usar [tokens de autenticação do AWS KMS](#) no AWS Encryption SDK. Se você usar um token de autenticação do AWS KMS, precisará instalar o AWS SDK for C++. O SDK da AWS é necessário para interagir com o [AWS Key Management Service](#) (AWS KMS). Quando você usa tokens de autenticação do AWS KMS, o AWS Encryption SDK usa o AWS KMS para gerar e proteger as chaves de criptografia que protegem seus dados.

Não é necessário instalar o AWS SDK for C++ se você estiver usando outro tipo de token de autenticação, como um token de autenticação do AES bruto, um token de autenticação do RSA bruto ou um multitoken de autenticação que não inclua um token de autenticação do AWS KMS. No entanto, ao usar um tipo de token de autenticação bruto, você precisa gerar e proteger suas próprias chaves de encapsulamento brutas.

Para ajudar a decidir qual tipo de token de inicialização deve ser criado, consulte [the section called “Como escolher um token de autenticação”](#).

Se você estiver com problemas com a instalação, [registre um problema](#) no repositório do aws-encryption-sdk-c ou use os links de feedback desta página.

Usar a AWS Encryption SDK for C

Este tópico explica alguns recursos do AWS Encryption SDK for C que não são compatíveis em outras implementações de linguagem de programação.

Esses exemplos mostram como usar a versão 2.0.x e versões posteriores do AWS Encryption SDK for C. Para obter exemplos que usam versões anteriores, encontre sua versão na lista de [Versões](#) do [repositório aws-encryption-sdk-c](#) no GitHub.

Para obter detalhes sobre programação com o AWS Encryption SDK for C, consulte os [exemplos de C](#), os [exemplos](#) no [repositório aws-encryption-sdk-c repository](#) no GitHub e a [documentação da API do AWS Encryption SDK for C](#).

Consulte também: [Usar tokens de autenticação](#)

Tópicos

- [Padrões para criptografar e descriptografar dados](#)
- [Contagem de referências](#)

Padrões para criptografar e descriptografar dados

Ao usar o AWS Encryption SDK for C, você segue um padrão semelhante a este: criar um [token de autenticação](#), criar um [CMM](#) que usa o token de autenticação, criar uma sessão que usa o CMM (e o token de autenticação) e processar a sessão.

1. Carregar sequências de erro.

Chame o método `aws_cryptosdk_load_error_strings()` no código C++ ou C++. Ele carrega informações de erro que são muito úteis para depuração.

Você só precisa chamá-lo uma vez, como no método `main`.

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

2. Crie um token de autenticação.

Configure o [token de autenticação](#) com as chaves de empacotamento que você deseja usar para criptografar suas chaves de dados. Este exemplo usa um [token de autenticação do AWS KMS com uma AWS KMS key](#), mas você pode usar qualquer tipo de token de autenticação em seu lugar.

Para identificar uma AWS KMS key em um token de autenticação de criptografia no AWS Encryption SDK for C, especifique um [ARN de chave](#) ou um [ARN de alias](#). Em um token de autenticação de descriptografia, é necessário usar um ARN de chave. Para obter mais detalhes, consulte [Identificação AWS KMS keys em um AWS KMS chaveiro](#).

```
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. Crie uma sessão.

No AWS Encryption SDK for C, você usa uma sessão para criptografar uma única mensagem de texto simples ou para descriptografar uma única mensagem de texto cifrado, independentemente de seu tamanho. A sessão mantém o estado da mensagem durante todo o processamento.

Configure a sessão com um alocador, um token de autenticação e um modo:

`AWS_CRYPTOSDK_ENCRYPT` ou `AWS_CRYPTOSDK_DECRYPT`. Se você precisar alterar o modo da sessão, use o método `aws_cryptosdk_session_reset`.

Ao criar uma sessão com um token de autenticação, o AWS Encryption SDK for C cria automaticamente um gerenciador de materiais criptográficos (CMM) padrão para você. Você não precisa criar, manter ou destruir esse objeto.

Por exemplo, a sessão a seguir usa o alocador e o token de autenticação definido na etapa 1. Ao criptografar dados, o modo é o `AWS_CRYPTOSDK_ENCRYPT`.

```
struct aws_cryptosdk_session * session =
    aws_cryptosdk_session_new_from_keyring_2(allocator, AWS_CRYPTOSDK_ENCRYPT,
    kms_keyring);
```

4. Criptografe ou descriptografe os dados.

Para processar os dados na sessão, use o método `aws_cryptosdk_session_process`.

Se o buffer de entrada for grande o suficiente para conter todo o texto simples, e o buffer de saída for grande o suficiente para conter todo o texto cifrado, você pode chamar `aws_cryptosdk_session_process_full`. No entanto, se precisar lidar com dados de streaming, você poderá chamar `aws_cryptosdk_session_process` em um loop. Para obter um exemplo, consulte o [file_streaming.cpp](#). O `aws_cryptosdk_session_process_full` foi introduzido nas versões 1.9.x e 2.2.x. do AWS Encryption SDK.

Quando a sessão é configurada para criptografar dados, os campos em texto simples descrevem a entrada e os campos de texto cifrado descrevem a saída. O campo `plaintext` contém a mensagem que você deseja criptografar, e o campo `ciphertext` obtém a [mensagem criptografada](#) retornada pelo método de criptografia.

```
/* Encrypting data */
aws_cryptosdk_session_process_full(session,
    ciphertext,
    ciphertext_buffer_size,
    &ciphertext_length,
    plaintext,
    plaintext_length)
```

Quando a sessão é configurada para descriptografar dados, os campos em texto cifrado descrevem a entrada e os campos em texto não cifrado descrevem a saída. O campo `ciphertext` contém a [mensagem criptografada](#) retornada pelo método de criptografia, e o campo `plaintext` obtém a mensagem em texto simples retornada pelo método de descriptografia.

Para descriptografar os dados, chame o método `aws_cryptosdk_session_process_full`.

```
/* Decrypting data */
aws_cryptosdk_session_process_full(session,
                                   plaintext,
                                   plaintext_buffer_size,
                                   &plaintext_length,
                                   ciphertext,
                                   ciphertext_length)
```

Contagem de referências

Para evitar vazamentos de memória, libere as referências a todos os objetos que você criou ao concluir. Caso contrário, ocorrerão vazamentos de memória. O SDK fornece métodos para facilitar essa tarefa.

Sempre que você criar um objeto pai com um dos seguintes objetos filho, o objeto pai obtém e mantém uma referência ao objeto filho, da seguinte forma:

- Um [token de autenticação](#), como criar uma sessão com um token de autenticação
- Um [gerenciador de materiais criptográficos](#) (CMM) padrão, como criar uma sessão ou um CMM personalizado com um CMM padrão
- Um [cache de chaves de dados](#), como criar um CMM de armazenamento em cache com um token de autenticação e um cache

A menos que precise de uma referência independente ao objeto filho, você pode liberar a referência ao objeto filho assim que criar o objeto pai. A referência restante ao objeto filho é liberada quando o objeto pai é destruído. Esse padrão garante que você mantenha a referência a cada objeto somente pelo tempo necessário e não ocorra vazamento de memória causado por referências não liberadas.

Você só é responsável por liberar referências aos objetos filho que cria explicitamente. Você não é responsável por gerenciar referências a objetos criados pelo SDK para você. Se o SDK criar um

objeto, como o CMM padrão que o método `aws_cryptosdk_caching_cmm_new_from_keyring` adiciona a uma sessão, o SDK gerenciará a criação e a destruição do objeto e suas referências.

No exemplo a seguir, ao criar uma sessão com um [token de autenticação](#), a sessão obtém uma referência ao token de autenticação e mantém essa referência até que a sessão seja destruída. Se você não precisar manter uma referência adicional ao token de autenticação, poderá usar o método `aws_cryptosdk_keyring_release` para liberar o objeto do token de autenticação assim que a sessão for criada. Esse método diminui a contagem de referências para o token de autenticação. A referência da sessão ao token de autenticação é liberada quando você chama `aws_cryptosdk_session_destroy` para destruir a sessão.

```
// 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);
```

Para tarefas mais complexas, como reutilizar um token de autenticação para várias sessões ou especificar um pacote de algoritmos em um CMM, talvez seja necessário manter uma referência independente ao objeto. Se assim for, não chame os métodos de liberação imediatamente. Em vez disso, libere as referências quando você não estiver mais usando os objetos, além de destruir a sessão.

Essa técnica de contagem de referências também funciona quando você está usando CMMs alternativos, como o CMM de armazenamento em cache para o [armazenamento em cache de chaves de dados](#). Ao criar um CMM de armazenamento em cache de um cache e de um token de autenticação, o CMM de armazenamento em cache obtém uma referência aos dois objetos. A menos que precise delas para outra tarefa, você pode liberar suas referências independentes para cache e o token de autenticação assim que o CMM de armazenamento em cache for criado. Depois, ao criar uma sessão com o CMM de armazenamento em cache, você pode liberar sua referência para o CMM de armazenamento em cache.

Observe que você só é responsável por liberar referências a objetos que cria explicitamente. Os objetos criados pelos métodos para você, como o CMM padrão que é subjacente ao CMM de armazenamento em cache, são gerenciados pelo método.

```
/ 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);
```

Exemplos do AWS Encryption SDK for C

Os exemplos a seguir mostram como usar o AWS Encryption SDK for C para criptografar e descriptografar dados.

Esses exemplos mostram como usar as versões 2.0.x e posteriores do AWS Encryption SDK for C. Para obter exemplos que usam versões anteriores, encontre sua versão na lista de [Versões](#) do [repositório aws-encryption-sdk-c](#) no GitHub.

Ao instalar e criar o AWS Encryption SDK for C, o código-fonte para estes e outros exemplos está incluído no subdiretório `examples` e é compilado e criado no diretório `build`. Você também pode localizá-los no subdiretório [examples](#) do repositório do [aws-encryption-sdk-c](#) no GitHub.

Tópicos

- [Criptografar e descriptografar strings](#)

Criptografar e descriptografar strings

O exemplo a seguir mostra como usar o AWS Encryption SDK for C para criptografar e descriptografar uma string.

Este exemplo apresenta o [token de autenticação do AWS KMS](#), um tipo de token de autenticação que usa uma AWS KMS key no [AWS Key Management Service \(AWS KMS\)](#) para gerar e criptografar chaves de dados. O exemplo inclui código escrito em C++. O AWS Encryption SDK for C exige que o AWS SDK for C++ chame o AWS KMS ao usar o token de autenticação do AWS KMS. Se você estiver usando um token de autenticação que não interage com o AWS KMS, como um token de autenticação do AES bruto, um token de autenticação do RSA bruto ou um multitoken de autenticação que não inclui um token de autenticação do AWS KMS, o AWS SDK for C++ não será necessário.

Para obter ajuda na criação de uma AWS KMS key, consulte [Criação de chaves](#) no Guia do desenvolvedor do AWS Key Management Service. Para obter ajuda para identificar o AWS KMS keys em um token de autenticação do AWS KMS, consulte [Identificação AWS KMS keys em um AWS KMS chaveiro](#).

Consulte o exemplo de código completo: [string.cpp](#)

Tópicos

- [Criptografar uma string](#)
- [Descriptografar uma string](#)

Criptografar uma string

A primeira parte deste exemplo usa um token de autenticação AWS KMS com uma AWS KMS key para criptografar uma string de texto simples.

Etapa 1. Carregar sequências de erro.

Chame o método `aws_cryptosdk_load_error_strings()` no código C++ ou C++. Ele carrega informações de erro que são muito úteis para depuração.

Você só precisa chamá-lo uma vez, como no método `main`.

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

Etapa 2: estruturar o token de autenticação.

Criar um token de autenticação do AWS KMS para criptografia. O token de autenticação neste exemplo é configurado com uma AWS KMS key, mas você pode configurar um token de

autenticação do AWS KMS com várias AWS KMS keys, inclusive AWS KMS keys em diferentes Regiões da AWS e contas.

Para identificar uma AWS KMS key em um token de autenticação de criptografia no AWS Encryption SDK for C, especifique um [ARN de chave](#) ou um [ARN de alias](#). Em um token de autenticação de descryptografia, é necessário usar um ARN de chave. Para obter mais detalhes, consulte [Identificação AWS KMS keys em um AWS KMS chaveiro](#).

[Identificação AWS KMS keys em um AWS KMS chaveiro](#)

Ao criar um token de autenticação com várias AWS KMS keys, você especifica a AWS KMS key usada para gerar e criptografar a chave de dados de texto simples e uma matriz opcional de AWS KMS keys adicionais que criptografam a mesma chave de dados de texto simples. Nesse caso, especificamos apenas a AWS KMS key geradora.

Antes de executar esse código, substitua o ARN da chave de exemplo por um válido.

```
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);
```

Etapa 3: Criar uma sessão.

Crie uma sessão usando o alocador, um enumerador de modo e o token de autenticação.

Cada sessão requer um modo: `AWS_CRYPTOSDK_ENCRYPT` para criptografar ou `AWS_CRYPTOSDK_DECRYPT` para descryptografar. Para alterar o modo de uma sessão existente, use o método `aws_cryptosdk_session_reset`.

Depois de criar um sessão com o token de autenticação, você poderá liberar sua referência ao token de autenticação usando o método fornecido pelo SDK. A sessão retém uma referência ao objeto token de autenticação durante sua vida útil. Referências ao token de autenticação e aos objetos de sessão são liberadas quando você destrói a sessão. Essa técnica de [contagem de referência](#) ajuda a evitar vazamentos de memória e a evitar que os objetos sejam liberados enquanto estão em uso.

```
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);

```

Etapa 4: Definir o contexto de criptografia.

Um [contexto de criptografia](#) são dados autenticados adicionais arbitrários e que não são secretos. Quando você fornece um contexto de criptografia na criptografia, o AWS Encryption SDK vincula de forma criptográfica o contexto ao texto cifrado, de forma que o mesmo contexto de criptografia seja necessário para descriptografar os dados. O uso de um contexto de criptografia é opcional, mas o recomendamos como uma melhor prática.

Primeiro, crie uma tabela de hash que inclua as strings de contexto de criptografia.

```

/* 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)

```

Obtenha um ponteiro mutável para o contexto de criptografia na sessão. Depois, use a função `aws_cryptosdk_enc_ctx_clone` para copiar o contexto de criptografia na sessão. Mantemos a cópia em `my_enc_ctx` para que possamos validar o valor depois de descriptografar os dados.

O contexto de criptografia faz parte da sessão, não é um parâmetro transmitido para a função de processo da sessão. Isso garante que o mesmo contexto de criptografia seja usado para todos os segmentos de uma mensagem, mesmo se a função de processo de sessão for chamada várias vezes para criptografar a mensagem inteira.

```

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)
```

Etapa 5: Criptografar a string.

Para criptografar a string em texto simples, use o método `aws_cryptosdk_session_process_full` com a sessão no modo de criptografia. Esse método, introduzido nas versões 1.9.x e 2.2.x do AWS Encryption SDK, foi projetado para criptografia e descriptografia que não sejam de streaming. Para lidar com dados de streaming, chame o `aws_cryptosdk_session_process` continuamente.

Na criptografia, os campos em texto simples são campos de entrada; os campos em texto cifrado são campos de saída. Concluído o processamento, o campo `ciphertext_output` conterá a [mensagem criptografada](#), incluindo o texto cifrado real, as chaves de dados criptografadas e o contexto de criptografia. Você pode descriptografar essa mensagem criptografada usando o AWS Encryption SDK em qualquer linguagem de programação compatível.

```
/* 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;
}
```

Passo 6: Limpar a sessão.

A etapa final destroi a sessão, inclusive as referências ao CMM e ao token de autenticação.

Se você preferir, em vez de destruir a sessão, reutilize-a com o mesmo token de autenticação e CMM para descriptografar a string, ou para criptografar ou descriptografar outras mensagens. Para usar a sessão para descriptografia, use o método `aws_cryptosdk_session_reset` para alterar o modo para `AWS_CRYPTOSDK_DECRYPT`.

Descriptografar uma string

A segunda parte deste exemplo descriptografa uma mensagem criptografada que contém o texto cifrado da string original.

Etapa 1: carregar sequências de erro.

Chame o método `aws_cryptosdk_load_error_strings()` no código C++ ou C++. Ele carrega informações de erro que são muito úteis para depuração.

Você só precisa chamá-lo uma vez, como no método `main`.

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

Etapa 2: estruturar o token de autenticação.

Ao descriptografar dados no AWS KMS, você transmite a [mensagem criptografada](#) retornada pela API de criptografia. A [API Decrypt](#) não precisa de uma AWS KMS key como entrada. Em vez disso, o AWS KMS usa a mesma AWS KMS key para descriptografar o texto cifrado usado para criptografá-lo. No entanto, o AWS Encryption SDK permite especificar um token de autenticação do AWS KMS com AWS KMS keys ao criptografar e ao descriptografar.

Ao descriptografar, você pode configurar um token de autenticação com apenas as AWS KMS keys que você deseja usar para descriptografar a mensagem criptografada. Por exemplo, talvez você queira criar um token de autenticação com apenas a AWS KMS key usada por uma função específica em sua organização. O AWS Encryption SDK nunca usa uma AWS KMS key, a menos que ela apareça no token de autenticação de descriptografia. Se o SDK não puder descriptografar as chaves de dados criptografadas usando as AWS KMS keys no token de autenticação que você fornecer, seja porque nenhuma das AWS KMS keys no token de autenticação foi usada para criptografar nenhuma das chaves de dados ou porque o chamador não tem permissão para usar as AWS KMS keys no token de autenticação para descriptografar, ocorrerá uma falha na chamada de descriptografia.

Ao especificar uma AWS KMS key para um token de autenticação de descriptografia, é necessário usar seu [ARN de chave](#). Os [ARNs de alias](#) são permitidos somente em tokens de autenticação de criptografia. Para obter ajuda para identificar o AWS KMS keys em um token de autenticação do AWS KMS, consulte [Identificação AWS KMS keys em um AWS KMS chaveiro](#).

Neste exemplo, especificamos um token de autenticação configurado com a mesma AWS KMS key usada para criptografar a string. Antes de executar esse código, substitua o ARN da chave de exemplo por um válido.

```
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);
```

Etapa 3: Criar uma sessão.

Crie uma sessão usando o alocador e o token de autenticação. Para configurar a sessão para descryptografia, configure a sessão com o modo `AWS_CRYPTOSDK_DECRYPT`.

Depois de criar uma sessão com um token de autenticação, você poderá liberar sua referência ao token de autenticação usando o método fornecido pelo SDK. A sessão mantém uma referência ao objeto token de autenticação durante sua vida útil, e a sessão e o token de autenticação são liberados quando você destrói a sessão. Essa técnica de contagem de referência ajuda a evitar vazamentos de memória e a evitar que os objetos sejam liberados enquanto estão em uso.

```
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);
```

Etapa 4: Descryptografar a string.

Para descryptografar a string, use o método `aws_cryptosdk_session_process_full` com a sessão configurada para descryptografia. Esse método, introduzido nas versões 1.9.x e 2.2.x do AWS Encryption SDK, foi projetado para criptografia e descryptografia que não sejam de streaming. Para lidar com dados de streaming, chame o `aws_cryptosdk_session_process` continuamente.

Na descryptografia, os campos em texto cifrado são campos de entrada, e os campos em texto simples são campos de saída. O campo `ciphertext_input` contém a [mensagem criptografada](#) retornada pelo método de criptografia. Quando o processamento for concluído, o campo `plaintext_output` conterá a string em texto simples (descryptografada).

```
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;
}

```

Etapa 5: Verificar o contexto de criptografia.

Verifique se o contexto de criptografia real, o que foi usado para descriptografar a mensagem, contém o contexto de criptografia fornecido ao criptografar a mensagem. O contexto de criptografia real pode incluir pares extras porque o [gerenciador de materiais criptográficos](#) (CMM) pode adicionar pares ao contexto de criptografia fornecido antes de criptografar a mensagem.

No AWS Encryption SDK for C, você não precisa fornecer um contexto de criptografia ao descriptografar porque o contexto é incluído na mensagem criptografada retornada pelo SDK. No entanto, antes que a mensagem em texto simples seja retornada, sua função de descriptografia deve verificar se todos os pares no contexto de criptografia fornecido aparecem no contexto que foi usado para descriptografar a mensagem.

Primeiro, obtenha um ponteiro somente leitura para a tabela de hash na sessão. Essa tabela de hash contém o contexto de criptografia que foi usado para descriptografar a mensagem.

```

const struct aws_hash_table *session_enc_ctx =
    aws_cryptosdk_session_get_enc_ctx_ptr(session);

```

Depois, percorra o contexto na tabela de hash `my_enc_ctx` que você copiou ao efetuar a criptografia. Verifique se cada par na tabela de hash `my_enc_ctx` que foi usada para efetuar a criptografia aparece na tabela de hash `session_enc_ctx` que foi usada para efetuar a descriptografia. Se alguma chave estiver ausente ou se essa chave tiver um valor diferente, interrompa o processamento e escreva uma mensagem de erro.

```

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();
}
}
```

Passo 6: Limpar a sessão.

Depois de verificar o contexto de criptografia, destrua a sessão ou a reutilize. Se precisar reconfigurá-la, use o método `aws_cryptosdk_session_reset`.

```
aws_cryptosdk_session_destroy(session);
```

AWS Encryption SDK para .NET

AWS Encryption SDK para .NET é uma biblioteca de criptografia do lado do cliente para desenvolvedores que escrevem aplicações em C# e em outras linguagens de programação .NET. É compatível com Windows, macOS e Linux.

Todas as implementações de [linguagem de programação](#) do AWS Encryption SDK são interoperáveis. No entanto, se você criptografar dados usando o [CMM de contexto de criptografia necessário](#) na versão 4.x do AWS Encryption SDK para .NET, só poderá descriptografá-lo com a versão 4.x do AWS Encryption SDK para .NET ou versão 3.x do AWS Encryption SDK for Java.

Note

A versão 4.0.0 do AWS Encryption SDK para .NET se desvia da Especificação da mensagem AWS Encryption SDK. Como resultado, as mensagens criptografadas pela versão 4.0.0 só podem ser descriptografadas pela versão 4.0.0 ou posterior do AWS Encryption SDK para .NET. Eles não podem ser descriptografados por nenhuma outra implementação de linguagem de programação.

A versão 4.0.1 do AWS Encryption SDK para .NET grava mensagens de acordo com a especificação da mensagem do AWS Encryption SDK e é interoperável com outras implementações de linguagem de programação. Por padrão, a versão 4.0.1 pode ler mensagens criptografadas pela versão 4.0.0. No entanto, se você não quiser descriptografar mensagens criptografadas pela versão 4.0.0, você pode especificar a propriedade [NetV4_0_0_RetryPolicy](#) para impedir que o cliente leia essas mensagens. Para obter

mais informações, consulte as [notas da versão v4.0.1](#) no repositório `aws-encryption-sdk-dafny` no GitHub.

O AWS Encryption SDK para .NET difere de algumas das outras implementações de linguagem de programação do AWS Encryption SDK das seguintes maneiras:

- Não há suporte para [armazenamento em cache de chaves de dados](#)

 Note

A versão 4.0.x do AWS Encryption SDK para .NET é compatível com o [token de autenticação hierárquico do AWS KMS](#), uma solução alternativa de cache de materiais criptográficos.

- Não há suporte para streaming de dados
- [Não há registros ou rastreamentos de pilha](#) do AWS Encryption SDK para .NET
- [Requer o AWS SDK for .NET](#)

O AWS Encryption SDK para .NET inclui todos os atributos de segurança introduzidos nas versões 2.0.x e posteriores de outras implementações de linguagem do AWS Encryption SDK. No entanto, se você estiver usando o AWS Encryption SDK for .NET para descriptografar dados criptografados por uma versão pré-2.0.x, outra implementação de linguagem do AWS Encryption SDK, talvez seja necessário ajustar sua [política de compromisso](#). Para obter mais detalhes, consulte [Como definir sua política de compromisso](#).

O AWS Encryption SDK para .NET é um produto do AWS Encryption SDK no [Dafny](#), uma linguagem de verificação formal na qual você escreve especificações, o código para implementá-las e as provas para testá-las. O resultado é uma biblioteca que implementa os atributos do AWS Encryption SDK em uma estrutura que garante a correção funcional.

Saiba mais

- Para ver exemplos de como configurar opções no AWS Encryption SDK, como especificar um pacote alternativo de algoritmos, limitar chaves de dados criptografadas e usar chaves de várias regiões do AWS KMS, consulte [Configurar o AWS Encryption SDK](#).
- Para obter detalhes sobre como programar com o AWS Encryption SDK para .NET, consulte o diretório [aws-encryption-sdk-net](#) do repositório `aws-encryption-sdk-dafny` no GitHub.

Tópicos

- [Instalar o AWS Encryption SDK para .NET](#)
- [Depuração do AWS Encryption SDK para .NET](#)
- [Tokens de autenticação do AWS KMS no AWS Encryption SDK para .NET](#)
- [Contextos de criptografia necessários na versão 4.x](#)
- [Exemplos de AWS Encryption SDK para .NET](#)

Instalar o AWS Encryption SDK para .NET

O AWS Encryption SDK para .NET está disponível como pacote do [AWS.Cryptography.EncryptionSDK](#) no NuGet. Para obter detalhes sobre como instalar e criar o AWS Encryption SDK para .NET, consulte o arquivo [README.md](#) no repositório `aws-encryption-sdk-net`.

Versão 3.x

Versão 3.x do AWS Encryption SDK para .NET oferece suporte ao .NET Framework 4.5.2 – 4.8 somente no Windows. É compatível com o .NET Core 3.0+ e o .NET 5.0 e versões posteriores em todos os sistemas operacionais compatíveis.

Versão 4.x

A versão 4.x do AWS Encryption SDK para .NET oferece suporte ao .NET 6.0 e ao .NET Framework net48 e versões posteriores.

O AWS Encryption SDK para .NET exige o AWS SDK for .NET mesmo se você não estiver usando as chaves AWS Key Management Service (AWS KMS). É instalado com o pacote NuGet. No entanto, a menos que você esteja usando chaves AWS KMS, o AWS Encryption SDK para .NET não exige uma Conta da AWS, credenciais da AWS ou interação com nenhum serviço da AWS. Para obter ajuda para configurar uma conta da AWS, se necessário, consulte [Uso da AWS Encryption SDK com o AWS KMS](#).

Depuração do AWS Encryption SDK para .NET

O AWS Encryption SDK para .NET não gera nenhum log. As exceções no AWS Encryption SDK para .NET geram uma mensagem de exceção, mas nenhum rastreamento de pilha.

Para ajudar na depuração, certifique-se de habilitar o login no AWS SDK for .NET. Os registros e mensagens de erro do AWS SDK for .NET podem ajudar você a distinguir os erros decorrentes do AWS SDK for .NET e os do AWS Encryption SDK para .NET. Para obter ajuda com o registro em log do AWS SDK for .NET, consulte [AWSLogging](#) no Guia do desenvolvedor do AWS SDK for .NET. (Para ver o tópico, expanda a seção Abrir para ver o conteúdo do .NET Framework.)

Tokens de autenticação do AWS KMS no AWS Encryption SDK para .NET

Os token de autenticação básicos do AWS KMS no AWS Encryption SDK para .NET usam apenas uma chave do KMS. Eles também exigem um cliente do AWS KMS, o que dá a você a oportunidade de configurar o cliente para a Região da AWS da chave KMS.

Para criar um token de autenticação do AWS KMS com uma ou mais chaves de encapsulamento, use um multitoken de autenticação. O AWS Encryption SDK para .NET tem um multitoken especial que aceita uma ou mais chaves do AWS KMS e um multitoken padrão que aceita um ou mais tokens de autenticação de qualquer tipo compatível. Alguns programadores preferem usar um método multitoken de autenticação para criar todos os seus tokens de autenticação, e o AWS Encryption SDK para .NET é compatível com essa estratégia.

O AWS Encryption SDK para .NET fornece tokens de autenticação básicos de chave única e multitokens de autenticação para todos os casos de uso típicos, incluindo [chaves multirregionais do AWS KMS](#).

Por exemplo, você pode usar o método `CreateAwsKmsKeyring()` para criar um token de autenticação do AWS KMS com uma chave do AWS KMS.

Version 3.x

O exemplo a seguir usa a versão 3 x do AWS Encryption SDK para .NET para criar um cliente do AWS KMS padrão para a região que contém a chave especificada.

```
// 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

O exemplo a seguir usa a versão 4x do AWS Encryption SDK para .NET para criar um cliente do AWS KMS para a região que contém a chave especificada.

```
// 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);
```

Para criar um token de autenticação com uma ou mais chaves do AWS KMS, use o método `CreateAwsKmsMultiKeyring()`. Este exemplo usa duas chaves do AWS KMS. Para especificar uma chave do KMS, use o parâmetro `Generator`. O parâmetro `KmsKeyIds`, que especifica chaves KMS adicionais, é opcional.

A entrada para este token de autenticação não requer um cliente do AWS KMS. Em vez disso, o AWS Encryption SDK usa o cliente do AWS KMS padrão para cada região representada por uma chave do KMS no token de autenticação. Por exemplo, se a chave KMS identificada pelo valor do parâmetro `Generator` estiver na região Oeste dos EUA (Oregon) (`us-west-2`), o AWS

Encryption SDK criará um cliente do AWS KMS padrão para a região us-west-2. Se você precisar personalizar o cliente do AWS KMS, use o método `CreateAwsKmsKeyring()`.

O exemplo a seguir usa a versão 4.x do AWS Encryption SDK para .NET e o método `CreateAwsKmsKeyring()` para personalizar o cliente do AWS KMS.

```
// 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);
```

A versão 4.x do AWS Encryption SDK para .NET é compatível com tokens de autenticação do AWS KMS que usam criptografia simétrica (`SYMMETRIC_DEFAULT`) ou chaves RSA do KMS assimétricas. Tokens de autenticação do AWS KMS criados com chaves RSA do KMS assimétricas só podem conter um par de chaves.

Para criptografar com um token de autenticação de RSA assimétrico do AWS KMS, você não precisa de [kms:GenerateDataKey](#) ou [kms:Encrypt](#), porque deve especificar o material de chave pública que deseja usar para criptografia ao criar o token de autenticação. Nenhuma chamada do AWS KMS é feita ao criptografar com este token de autenticação. Para descriptografar com um token de autenticação de RSA assimétrico do AWS KMS, é necessária a permissão [KMS:Decrypt](#).

Para criar um token de autenticação de RSA assimétrico do AWS KMS, você deve fornecer o ARN de chave pública e de chave privada da sua chave RSA assimétrica do KMS. A chave pública deve

ser codificada em PEM. O exemplo a seguir cria um token de autenticação do AWS KMS com um par de chaves RSA assimétricas.

```
// 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);
```

Contextos de criptografia necessários na versão 4.x

Com a versão 4.x do AWS Encryption SDK para .NET, você pode usar o CMM de contexto de criptografia necessário para exigir [contextos de criptografia](#) em suas operações criptográficas. Um contexto de criptografia é um conjunto de pares de chave/valor não secretos. O contexto de criptografia é associado de maneira criptográfica aos dados criptografados de forma que o mesmo contexto de criptografia é necessário para descriptografar o campo. Ao usar o CMM de contexto de criptografia necessário, é possível especificar uma ou mais chaves de contexto de criptografia necessárias (chaves obrigatórias) que devem ser incluídas em todas as chamadas de criptografia e descriptografia.

Note

O contexto de criptografia necessário (CMM) só é interoperável com a versão 3.x do AWS Encryption SDK for Java. Não é interoperável com nenhuma outra implementação de linguagem de programação. Se você criptografar dados usando o CMM de contexto de criptografia necessário, só poderá descriptografá-los com a versão 3.x do AWS Encryption SDK for Java ou a versão 4.x do AWS Encryption SDK para .NET.

Ao criptografar, o AWS Encryption SDK verifica se todas as chaves de contexto de criptografia necessárias estão incluídas no contexto de criptografia que você especificou. O AWS Encryption SDK sinaliza os contextos de criptografia que você especificou. Somente os pares de chave/valor que não são chaves obrigatórias são serializados e armazenados em texto simples no cabeçalho da mensagem criptografada retornada pela operação de criptografia.

Ao descriptografar, você deve fornecer um contexto de criptografia que contenha todos os pares de chave/valor que representam as chaves necessárias. O AWS Encryption SDK usa esse contexto de criptografia e os pares de chave/valor armazenados no cabeçalho da mensagem criptografada para reconstruir o contexto de criptografia original que você especificou na operação de criptografia. Se o AWS Encryption SDK não puder reconstruir o contexto de criptografia original, a operação de descriptografia falhará. Se você fornecer um par de chave/valor que contenha a chave necessária com um valor incorreto, a mensagem criptografada não poderá ser descriptografada. Você deve fornecer o mesmo par de chave/valor especificado na criptografia.

Important

Considere cuidadosamente quais valores você escolhe para as chaves necessárias no contexto de criptografia. Você deverá fornecer as mesmas chaves e os valores correspondentes novamente na descriptografia. Se você não conseguir reproduzir as chaves necessárias, a mensagem criptografada não poderá ser descriptografada.

O exemplo a seguir inicializa um token de autenticação do AWS KMS com o CMM de contexto de criptografia necessário.

```
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);
```

Se você usar um token de autenticação do AWS KMS, o AWS Encryption SDK para .NET também usará o contexto de criptografia para fornecer dados autenticados adicionais (AAD) nas chamadas que o token de autenticação faz para o AWS KMS.

Exemplos de AWS Encryption SDK para .NET

Os exemplos a seguir mostram os padrões básicos de codificação usados ao programar com o AWS Encryption SDK para .NET. Especificamente, você deve instanciar o AWS Encryption SDK e a biblioteca de fornecedores de materiais. Em seguida, antes de chamar cada método, você deve instanciar um objeto que define a entrada para o método. Isso é muito parecido com o padrão de codificação usado no AWS SDK for .NET.

Para ver exemplos de como configurar opções no AWS Encryption SDK, como especificar um pacote alternativo de algoritmos, limitar chaves de dados criptografadas e usar chaves de várias regiões do AWS KMS, consulte [Configurar o AWS Encryption SDK](#).

Para obter mais exemplos de programação com o AWS Encryption SDK for .NET, consulte os [exemplos](#) no diretório `aws-encryption-sdk-net` do repositório `aws-encryption-sdk-dafny` no GitHub.

Criptografia de dados no AWS Encryption SDK para .NET

Este exemplo mostra o padrão básico para criptografar dados. Ele criptografa um pequeno arquivo com chaves de dados protegidas por uma chave de encapsulamento do AWS KMS.

Etapa 1: instanciar o AWS Encryption SDK e a biblioteca dos fornecedores de materiais.

Comece instanciando o AWS Encryption SDK e a biblioteca dos fornecedores de materiais. Você usará os métodos do AWS Encryption SDK para criptografar e descriptografar dados. Você usará os métodos na biblioteca de fornecedores de materiais para criar os tokens de autenticação que especificam quais chaves protegem seus dados.

A forma como você instancia o AWS Encryption SDK e a biblioteca de fornecedores de materiais difere entre as versões 3.x e 4.x do AWS Encryption SDK para .NET. Todas as etapas a seguir são as mesmas para ambas as versões, 3.x e 4.x, do AWS Encryption SDK para .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());
```

Etapa 2: crie um objeto de entrada para o token de autenticação.

Cada método que cria um token de autenticação tem uma classe de objeto de entrada correspondente. Por exemplo, para criar o objeto de entrada para o método `CreateAwsKmsKeyring()`, crie uma instância da classe `CreateAwsKmsKeyringInput`.

Embora a entrada desse token de autenticação não especifique uma [chave geradora](#), a única chave do KMS especificada pelo parâmetro `KmsKeyId` é a chave geradora. Ela gera e criptografa a chave de dados que criptografa os dados.

Esse objeto de entrada requer um cliente do AWS KMS para a Região da AWS da chave do KMS. Para criar um cliente do AWS KMS, instancie a classe

`AmazonKeyManagementServiceClient` no AWS SDK for .NET. Chamar o construtor `AmazonKeyManagementServiceClient()` sem parâmetros cria um cliente com os valores padrão.

Em um token de autenticação do AWS KMS usado para criptografar com o AWS Encryption SDK para .NET, é possível [identificar as chaves do KMS](#) usando o ID de chave, o ARN de chave, o nome de alias ou o ARN de alias. Em um token de autenticação AWS KMS usado para criptografia e descryptografia, você deve usar um ARN de chave para identificar cada chave do KMS. Se você planeja reutilizar seu token de autenticação de criptografia para descryptografar, use um identificador ARN de chave para todas as chaves 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  
};
```

Etapa 3: criar o token de autenticação.

Para criar o token de autenticação, chame o método do token de autenticação com o objeto de entrada do token de autenticação. Este exemplo usa o método `CreateAwsKmsKeyring()`, que usa apenas uma chave do KMS.

```
var keyring = materialProviders.CreateAwsKmsKeyring(kmsKeyringInput);
```

Etapa 4: defina um contexto de criptografia.

Um [contexto de criptografia](#) é um elemento opcional, mas extremamente recomendado, nas operações de criptografia do AWS Encryption SDK. Você pode definir um ou mais pares de chave-valor não secretos.

Note

Com a versão 4.X do AWS Encryption SDK para .NET, você pode exigir um contexto de criptografia em todas as solicitações de criptografia com o [CMM de contexto de criptografia necessário](#).

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

Etapa 5: crie o objeto de entrada para a criptografia.

Antes de chamar o método `Encrypt()`, crie uma instância da classe `EncryptInput`.

```
string plaintext = File.ReadAllText("C:\\Documents\\CryptoTest\\TestFile.txt");

// Define the encrypt input
var encryptInput = new EncryptInput
{
    Plaintext = plaintext,
    Keyring = keyring,
    EncryptionContext = encryptionContext
};
```

Etapa 6: criptografe o texto sem formatação.

Use o método `Encrypt()` do AWS Encryption SDK para criptografar o texto simples usando o token de autenticação definido por você.

O `EncryptOutput` que o método `Encrypt()` retorna tem métodos para obter a mensagem criptografada (`Ciphertext`), o contexto de criptografia e o pacote de algoritmos.

```
var encryptOutput = encryptionSdk.Encrypt(encryptInput);
```

Etapa 7: obtenha a mensagem criptografada.

O método `Decrypt()` no AWS Encryption SDK para .NET usa o membro `Ciphertext` da instância `EncryptOutput`.

O membro `Ciphertext` do objeto `EncryptOutput` é a [mensagem criptografada](#), um objeto portátil que inclui dados criptografados, chaves de dados criptografadas e metadados, incluindo o contexto de criptografia. É possível armazenar com segurança a mensagem criptografada por um período prolongado ou enviá-la ao método `Decrypt()` para recuperar o texto sem formatação.

```
var encryptedMessage = encryptOutput.Ciphertext;
```

Descriptografia em modo estrito no AWS Encryption SDK para .NET

As práticas recomendadas indicam que você especifique as chaves usadas para descriptografar dados, uma opção conhecida como modo estrito. O AWS Encryption SDK usa somente as chaves do KMS que você especifica no token de autenticação para descriptografar o texto cifrado. As chaves no token de autenticação de descriptografia devem incluir pelo menos uma das chaves que criptografaram os dados.

Este exemplo mostra o padrão básico de descriptografia no modo estrito com o AWS Encryption SDK para .NET.

Etapa 1: instanciar a o AWS Encryption SDK e a biblioteca de fornecedores de materiais.

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

Etapa 2: criar o objeto de entrada para seu token de autenticação.

Para especificar os parâmetros do método do token de autenticação, crie um objeto de entrada. Cada método de token de autenticação no AWS Encryption SDK para .NET tem um objeto de entrada correspondente. Como esse exemplo usa o método `CreateAwsKmsKeyring()` para criar o token de autenticação, ele instancia a classe `CreateAwsKmsKeyringInput` para a entrada.

Em um token de autenticação de descriptografia, você deve usar um ARN de chave para identificar chaves do 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
};
```

Etapa 3: criar o token de autenticação.

Para criar o token de autenticação da descryptografia, este exemplo usa o método `CreateAwsKmsKeyring()` e o objeto de entrada do token de autenticação.

```
var keyring = materialProviders.CreateAwsKmsKeyring(kmsKeyringInput);
```

Etapa 4: crie o objeto de entrada para descryptografar.

Para criar o objeto de entrada para o método `Decrypt()`, instancie a classe `DecryptInput`.

O parâmetro `Ciphertext` do construtor `DecryptInput()` usa o membro `Ciphertext` do objeto `EncryptOutput` que o método `Encrypt()` retornou. A propriedade `Ciphertext` representa a [mensagem criptografada](#), que inclui os dados criptografados, as chaves de dados criptografadas e os metadados que o AWS Encryption SDK necessita para descryptografar a mensagem.

Com a versão 4.x do AWS Encryption SDK para .NET, você pode usar o parâmetro opcional `EncryptionContext` para especificar o contexto de criptografia no método `Decrypt()`.

Use o parâmetro `EncryptionContext` para verificar se o contexto de criptografia usado na criptografia está incluído no contexto de criptografia usado para descryptografar o texto cifrado. O AWS Encryption SDK adiciona pares ao contexto de criptografia, incluindo a assinatura digital, se você estiver usando um pacote de algoritmos com assinatura, como o pacote de algoritmos padrão.

```
var encryptedMessage = encryptOutput.Ciphertext;

var decryptInput = new DecryptInput
{
    Ciphertext = encryptedMessage,
    Keyring = keyring,
    EncryptionContext = encryptionContext // OPTIONAL
};
```

Etapa 5: descryptografe o texto cifrado.

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

Etapa 6: verifique o contexto de criptografia – versão 3.x

O método `Decrypt()` da versão 3.x do AWS Encryption SDK para .NET não usa um contexto de criptografia. Ele obtém os valores do contexto de criptografia dos metadados na mensagem criptografada. No entanto, antes de retornar ou usar o texto simples, é recomendável verificar se o contexto de criptografia usado para descriptografar o texto cifrado inclui o contexto de criptografia que você forneceu ao criptografar.

Verifique se o contexto de criptografia usado na criptografia está incluído no contexto de criptografia usado para descriptografar o texto cifrado. O AWS Encryption SDK adiciona pares ao contexto de criptografia, incluindo a assinatura digital, se você estiver usando um pacote de algoritmos com assinatura, como o pacote de algoritmos padrão.

```
// 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");
}
```

Descriptografar com um token de autenticação de descoberta no AWS Encryption SDK para .NET

Em vez de especificar as chaves KMS para a descriptografia, você pode fornecer um token de autenticação de descoberta do AWS KMS, que é um token de autenticação que não especifica nenhuma chave KMS. Um token de autenticação de descoberta permite que o AWS Encryption SDK descriptografe os dados usando qualquer chave KMS criptografada, desde que o chamador tenha permissão de para descriptografar na chave. Para obter as práticas recomendadas, adicione um filtro de descoberta que limita as chaves do KMS que podem ser usadas para aquelas específicas das Contas da AWS de uma partição especificada.

O AWS Encryption SDK para .NET fornece um token de autenticação de descoberta básico que requer um cliente de AWS KMS e um token de autenticação de descoberta múltiplo que requer que você especifique uma ou mais Regiões da AWS. Tanto o cliente quanto as regiões limitam as chaves

do KMS que podem ser usadas para descriptografar a mensagem criptografada. Os objetos de entrada dos dois tokens de autenticação usam o filtro de descoberta recomendado.

O exemplo a seguir mostra o padrão para descriptografar dados com um token de autenticação de descoberta do AWS KMS e um filtro de descoberta.

Etapa 1: instanciar o AWS Encryption SDK e a biblioteca dos fornecedores de materiais.

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

Etapa 2: criar o objeto de entrada para o token de autenticação.

Para especificar os parâmetros do método do token de autenticação, crie um objeto de entrada. Cada método de token de autenticação no AWS Encryption SDK para .NET tem um objeto de entrada correspondente. Como esse exemplo usa o método `CreateAwsKmsDiscoveryKeyring()` para criar o token de autenticação, ele instancia a classe `CreateAwsKmsDiscoveryKeyringInput` para a entrada.

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

var discoveryKeyringInput = new CreateAwsKmsDiscoveryKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    DiscoveryFilter = new DiscoveryFilter()
    {
        AccountIds = accounts,
        Partition = "aws"
    }
};
```

Etapa 3: criar o token de autenticação.

Para criar o token de autenticação da descriptografia, este exemplo usa o método `CreateAwsKmsDiscoveryKeyring()` e o objeto de entrada do token de autenticação.

```
var discoveryKeyring =
    materialProviders.CreateAwsKmsDiscoveryKeyring(discoveryKeyringInput);
```

Etapa 4: crie o objeto de entrada para descriptografar.

Para criar o objeto de entrada para o método `Decrypt()`, instancie a classe `DecryptInput`. O valor do parâmetro `Ciphertext` é o membro `Ciphertext` do objeto `EncryptOutput` que o método `Encrypt()` retorna.

Com a versão 4.x do AWS Encryption SDK para .NET, você pode usar o parâmetro opcional `EncryptionContext` para especificar o contexto de criptografia no método `Decrypt()`.

Use o parâmetro `EncryptionContext` para verificar se o contexto de criptografia usado na criptografia está incluído no contexto de criptografia usado para descriptografar o texto cifrado. O AWS Encryption SDK adiciona pares ao contexto de criptografia, incluindo a assinatura digital, se você estiver usando um pacote de algoritmos com assinatura, como o pacote de algoritmos padrão.

```
var ciphertext = encryptOutput.Ciphertext;

var decryptInput = new DecryptInput
{
    Ciphertext = ciphertext,
    Keyring = discoveryKeyring,
    EncryptionContext = encryptionContext // OPTIONAL
};

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

Etapa 5: verificar o contexto de criptografia - versão 3.x

O método `Decrypt()` da versão 3.x do AWS Encryption SDK para .NET não usa um contexto de criptografia em `Decrypt()`. Ele obtém os valores do contexto de criptografia dos metadados na mensagem criptografada. No entanto, antes de retornar ou usar o texto simples, é recomendável verificar se o contexto de criptografia usado para descriptografar o texto cifrado inclui o contexto de criptografia que você forneceu ao criptografar.

Verifique se o contexto de criptografia usado na criptografia foi incluído no contexto de criptografia usado para descriptografar o texto cifrado. O AWS Encryption SDK adiciona pares ao contexto de criptografia, incluindo a assinatura digital, se você estiver usando um pacote de algoritmos com assinatura, como o pacote de algoritmos padrão.

```
// 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");
}
```

AWS Encryption SDK for Java

Este tópico explica como instalar e usar o AWS Encryption SDK for Java. Para obter detalhes sobre a programação com o AWS Encryption SDK for Java, consulte o [aws-encryption-sdk-java](#) repositório em GitHub. Para obter a documentação da API, consulte o [Javadoc](#) para AWS Encryption SDK for Java.

Tópicos

- [Pré-requisitos](#)
- [Instalação](#)
- [AWS KMS Chaveiros no AWS Encryption SDK for Java](#)
- [Contextos de criptografia necessários na versão 3.x](#)
- [Exemplos do AWS Encryption SDK for Java](#)

Pré-requisitos

Antes de instalar o AWS Encryption SDK for Java, verifique se você tem os pré-requisitos a seguir.

Um ambiente de desenvolvimento Java

Você precisará do Java 8 ou posterior. No site da Oracle, acesse [Java SE Downloads](#) e faça download e instale o Java SE Development Kit (JDK).

Se você usa o Oracle JDK, também precisará fazer download e instalar os [arquivos de política de jurisdição de força ilimitada JCE \(Java Cryptography Extension\)](#).

Bouncy Castle

O AWS Encryption SDK for Java requer o [Bouncy Castle](#).

- O AWS Encryption SDK for Java versões 1.6.1 e posteriores usam Bouncy Castle para serializar e desserializar objetos de criptografia. Você pode usar o Bouncy Castle ou o [Bouncy Castle FIPS](#) para atender a esse requisito. Para obter ajuda para instalar e configurar o Bouncy Castle FIPS, consulte a [Documentação do BC FIPS](#), especialmente os Guias do usuário e os PDFs de Política de segurança .
- Versões anteriores do AWS Encryption SDK for Java usam a API de criptografia do Bouncy Castle para Java. Este requisito só é atendido por não FIPS Bouncy Castle.

Se você não tiver o Bouncy Castle, visite o [Bouncy Castle latest releases](#) para fazer download do arquivo do provedor que corresponde a seu JDK. [Você também pode usar o Apache Maven para obter o artefato para o provedor padrão do Bouncy Castle \(bcprov-ext-jdk15on\) ou o artefato para o Bouncy Castle FIPS \(bc-fips\).](#)

AWS SDK for Java

Versão 3. x of the AWS Encryption SDK for Java requer o AWS SDK for Java 2.x, mesmo se você não usar AWS KMS chaves.

Versão 2. x ou anterior do AWS Encryption SDK for Java não requer AWS SDK for Java o. No entanto, o AWS SDK for Java é necessário para usar o [AWS Key Management Service](#) (AWS KMS) como provedor de chave mestra. A partir da versão 2.4.0 do AWS Encryption SDK for Java, o AWS Encryption SDK for Java é compatível com as versões 1.x e 2.x do AWS SDK for Java. os códigos do AWS Encryption SDK para o AWS SDK for Java versões 1.x e 2.x são interoperáveis. Por exemplo, você pode criptografar dados com um código AWS Encryption SDK compatível com o AWS SDK for Java 1.x e descriptografá-los usando um código compatível com o AWS SDK for Java 2.x (ou vice-versa). As versões do AWS Encryption SDK for Java anteriores à 2.4.0 somente são compatíveis com o AWS SDK for Java 1.x. Para obter mais informações sobre a atualização da sua versão do AWS Encryption SDK, consulte [Como migrar seu AWS Encryption SDK](#).

Ao atualizar seu código do AWS Encryption SDK for Java de AWS SDK for Java 1.x para AWS SDK for Java 2.x, substitua as referências à [interface AWKMS](#) no AWS SDK for Java 1.x por referências à [interface KmsClient](#) no AWS SDK for Java 2.x. O AWS Encryption SDK for Java não é compatível com a [interface KmsAsyncClient](#). Além disso, atualize seu código para usar os objetos relacionados ao AWS KMS no namespace kmsdkv2, em vez do namespace kms.

Para instalar o AWS SDK for Java, use o Apache Maven.

- Para [importar todo o AWS SDK for Java](#) como uma dependência, declare-o no arquivo `pom.xml`.

- Para criar uma dependência somente para o módulo AWS KMS no AWS SDK for Java 1.x, siga as instruções para [especificar módulos específicos](#), e defina o `artifactId` como `aws-java-sdk-kms`.
- Para criar uma dependência somente para o módulo AWS KMS no AWS SDK for Java 2.x, siga as instruções para [especificar módulos específicos](#). Defina o `groupId` como `software.amazon.awssdk` e `artifactId` como `kms`.

Para ver mais mudanças, consulte [Qual a diferença entre a versão 1.x e a 2.x do AWS SDK for Java](#) no Guia do Desenvolvedor do AWS SDK for Java 2.x.

Os exemplos de Java no Guia do Desenvolvedor do AWS Encryption SDK usam o AWS SDK for Java 2.x.

Instalação

Instalar a versão mais recente do AWS Encryption SDK for Java.

Note

Todas as versões AWS Encryption SDK for Java anteriores à 2.0.0 estão em [end-of-support fase](#).

Você pode atualizar com segurança a partir da versão 2.0.x e posteriores até a versão mais recente do AWS Encryption SDK for Java sem realizar alterações no código ou nos dados. No entanto, os [novos atributos de segurança](#) introduzidos na versão 2.0.x não são compatíveis com versões anteriores. Para atualizar a partir de versões anteriores à 1.7.x até a versão 2.0.x e posteriores, primeiro será necessário atualizar para a versão 1.x mais recente do AWS Encryption SDK. Para obter detalhes, consulte [Como migrar seu AWS Encryption SDK](#).

É possível instalar o AWS Encryption SDK for Java das seguintes maneiras.

Manualmente

Para instalar o AWS Encryption SDK for Java, clone ou baixe o [aws-encryption-sdk-java](#) GitHub repositório.

Uso do Apache Maven

O AWS Encryption SDK for Java está disponível por meio do [Apache Maven](#) com a definição de dependência a seguir.

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-encryption-sdk-java</artifactId>
  <version>3.0.0</version>
</dependency>
```

Depois de instalar o SDK, comece examinando o [exemplo de código Java](#) neste guia e o [Javadoc](#) ativado. [GitHub](#)

AWS KMSchaveiros no AWS Encryption SDK for Java

Versão 3. x do AWS Encryption SDK for Java usa [chaveiros](#) para realizar a criptografia de [envelopes](#). Os AWS KMS chaveiros básicos do pacote AWS Encryption SDK for Java levam apenas uma chave KMS. Eles também exigem um cliente do AWS KMS, o que dá a você a oportunidade de configurar o cliente para a Região da AWS da chave KMS.

Para criar um token de autenticação do AWS KMS com uma ou mais chaves de encapsulamento, use um multitoken de autenticação. AWS Encryption SDK for JavaTem um porta-chaves especial que aceita uma ou mais AWS KMS teclas e um porta-chaves padrão que aceita um ou mais chaveiros de qualquer tipo compatível. Alguns programadores preferem usar um método de vários chaveiros para criar todos os seus chaveiros, e ele apóia essa estratégia. AWS Encryption SDK for Java

AWS Encryption SDK for Java[Ele fornece chaveiros básicos de chave única e chaveiros múltiplos para todos os casos de uso típicos, incluindo chaves multirregionais. AWS KMS](#)

Por exemplo, para criar um AWS KMS chaveiro com uma AWS KMS chave, você pode usar o método `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);
```

Para criar um token de autenticação com uma ou mais chaves do AWS KMS, use o método `CreateAwsKmsMultiKeyring()`. Este exemplo usa duas chaves KMS. Para especificar uma chave do KMS, use o parâmetro `generator`. O parâmetro `msKeyIds`, que especifica chaves KMS adicionais, é opcional.

A entrada para este token de autenticação não requer um cliente do AWS KMS. Em vez disso, o AWS Encryption SDK usa o cliente do AWS KMS padrão para cada região representada por uma chave do KMS no token de autenticação. Por exemplo, se a chave KMS identificada pelo valor do parâmetro `Generator` estiver na região Oeste dos EUA (Oregon) (`us-west-2`), o AWS Encryption SDK criará um cliente do AWS KMS padrão para a região `us-west-2`. Se você precisar personalizar o cliente do AWS KMS, use o método `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();

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);
```

AWS Encryption SDK for Java suporta AWS KMS chaveiros que usam criptografia simétrica (`SYMMETRIC_DEFAULT`) ou chaves RSA KMS assimétricas. AWS KMS chaveiros criados com chaves RSA KMS assimétricas só podem conter um par de chaves.

Para criptografar com um AWS KMS chaveiro RSA assimétrico, você não precisa de [kms:GenerateDataKey](#) ou [kms:Encrypt](#) porque você deve especificar o material de chave pública que deseja usar para criptografia ao criar o chaveiro. Nenhuma chamada do AWS KMS é feita ao criptografar com este token de autenticação. Para descriptografar com um token de autenticação de RSA assimétrico do AWS KMS, é necessária a permissão [KMS:Decrypt](#).

Para criar um token de autenticação de RSA assimétrico do AWS KMS, você deve fornecer o ARN de chave pública e de chave privada da sua chave RSA assimétrica do KMS. A chave pública deve ser codificada em PEM. O exemplo a seguir cria um token de autenticação do AWS KMS com um par de chaves RSA assimétricas.

```
// 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);
```

Contextos de criptografia necessários na versão 3.x

Com a versão 3. x do AWS Encryption SDK for Java, você pode usar o contexto de criptografia necessário CMM para exigir [contextos de criptografia](#) em suas operações criptográficas. Um contexto de criptografia é um conjunto de pares de chave/valor não secretos. O contexto de criptografia é associado de maneira criptográfica aos dados criptografados de forma que o mesmo contexto de criptografia é necessário para descriptografar o campo. Ao usar o CMM de contexto de criptografia necessário, é possível especificar uma ou mais chaves de contexto de criptografia necessárias (chaves obrigatórias) que devem ser incluídas em todas as chamadas de criptografia e descriptografia.

Note

O contexto de criptografia necessário (CMM) só é interoperável com a versão 4. x do AWS Encryption SDK para o.NET. Não é interoperável com nenhuma outra implementação de linguagem de programação. Se você criptografar dados usando o contexto de criptografia necessário CMM, só poderá descriptografá-los com a versão 3. x do AWS Encryption SDK for Java ou versão 4. x do AWS Encryption SDK para o.NET.

Ao criptografar, o AWS Encryption SDK verifica se todas as chaves de contexto de criptografia necessárias estão incluídas no contexto de criptografia que você especificou. O AWS Encryption SDK sinaliza os contextos de criptografia que você especificou. Somente os pares de chave/valor que não são chaves obrigatórias são serializados e armazenados em texto simples no cabeçalho da mensagem criptografada retornada pela operação de criptografia.

Ao descriptografar, você deve fornecer um contexto de criptografia que contenha todos os pares de chave/valor que representam as chaves necessárias. O AWS Encryption SDK usa esse contexto de criptografia e os pares de chave/valor armazenados no cabeçalho da mensagem criptografada para reconstruir o contexto de criptografia original que você especificou na operação de criptografia. Se o AWS Encryption SDK não puder reconstruir o contexto de criptografia original, a operação de descriptografia falhará. Se você fornecer um par de chave/valor que contenha a chave necessária com um valor incorreto, a mensagem criptografada não poderá ser descriptografada. Você deve fornecer o mesmo par de chave/valor especificado na criptografia.

⚠ Important

Considere cuidadosamente quais valores você escolhe para as chaves necessárias no contexto de criptografia. Você deverá fornecer as mesmas chaves e os valores correspondentes novamente na descriptografia. Se você não conseguir reproduzir as chaves necessárias, a mensagem criptografada não poderá ser descriptografada.

O exemplo a seguir inicializa um token de autenticação do AWS KMS com o CMM de contexto de criptografia necessário.

```
// 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()
        );
```

Exemplos do AWS Encryption SDK for Java

Os exemplos a seguir mostram como usar o AWS Encryption SDK for Java para criptografar e descriptografar dados. Esses exemplos mostram como usar a versão 3. x e posterior do AWS Encryption SDK for Java. Versão 3. x do AWS Encryption SDK for Java substitui os [fornecedores de chaves mestras](#) por [chaveiros](#). Para exemplos que usam versões anteriores, encontre sua versão na lista de [lançamentos](#) do [aws-encryption-sdk-java](#) repositório em GitHub.

Tópicos

- [Criptografar e descriptografar strings](#)
- [Criptografar e descriptografar streams de bytes](#)
- [Criptografando e descriptografando fluxos de bytes com um chaveiro múltiplo](#)

Criptografar e descriptografar strings

O exemplo a seguir mostra como usar a versão 3. x do AWS Encryption SDK for Java para criptografar e descriptografar strings. Antes de usar a string, converta-a em uma matriz de bytes.

Este exemplo usa um [AWS KMSChaveiro](#). Ao criptografar com um AWS KMS chaveiro, você pode usar um ID de chave, ARN de chave, nome de alias ou ARN de alias para identificar as chaves KMS. Ao descriptografar, você deve usar um ARN de chave para identificar as chaves KMS.

Quando você chama o método `encryptData()`, ele retorna uma [mensagem criptografada](#) (`CryptoResult`) que inclui o texto cifrado, as chaves de dados criptografadas e o contexto de criptografia. Quando você chama `getResult` no objeto `CryptoResult`, ele retorna uma versão de cadeia codificada em base 64 da [mensagem criptografada](#) que você pode passar para o método `decryptData()`.

Da mesma forma, quando você chama `decryptData()`, o objeto `CryptoResult` que ele retorna contém a mensagem de texto simples e um ID de AWS KMS key. Antes do seu aplicativo retornar o texto simples, verifique se o ID da AWS KMS key e o contexto de criptografia na mensagem criptografada são os esperados.

```
// 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
</li>
 * </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);
}
}
```

Criptografar e descriptografar streams de bytes

O exemplo a seguir mostra como usar o AWS Encryption SDK para criptografar e descriptografar streams de bytes.

Este exemplo usa um [chaveiro AES bruto](#).

Ao criptografar, o método `AwsCrypto.builder().withEncryptionAlgorithm()` é usado para especificar um conjunto de algoritmos sem [assinaturas digitais](#).

Ao descriptografar, para garantir que o texto cifrado não esteja assinado, este exemplo usa o método `createUnsignedMessageDecryptingStream()`. O `createUnsignedMessageDecryptingStream()` método falhará se encontrar um texto cifrado com uma assinatura digital.

Se você estiver criptografando com o conjunto de algoritmos padrão, que inclui assinaturas digitais, use o método `createDecryptingStream()` em seu lugar, conforme mostrado no próximo exemplo.

```
// 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("Examp
{
    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");
}
}

```

Criptografando e descriptografando fluxos de bytes com um chaveiro múltiplo

O exemplo a seguir mostra como usar o AWS Encryption SDK com um [chaveiro múltiplo](#). Quando você usa um multitoken de autenticação para criptografar dados, qualquer uma das chaves de empacotamento em qualquer um de seus tokens de autenticação pode descriptografar esses dados. Este exemplo usa um [AWS KMSChaveiro e um chaveiro RSA bruto como chaveiros](#) secundários.

Este exemplo criptografa com o [pacote de algoritmos padrão](#), que inclui uma [assinatura digital](#). Durante o streaming, o AWS Encryption SDK libera texto simples após as verificações de integridade, mas antes de verificar a assinatura digital. Para evitar o uso do texto simples até que a assinatura seja verificada, este exemplo armazena o texto simples em buffer e o grava no disco somente após a conclusão da descriptografia e da verificação.

```
// 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());
    }
}
```

AWS Encryption SDK para JavaScript

O AWS Encryption SDK para JavaScript foi projetado para fornecer uma biblioteca de criptografia do lado do cliente para desenvolvedores que estão escrevendo aplicativos de navegador da Web em JavaScript ou aplicativos de servidor Web em Node.js.

Como todas as implementações do AWS Encryption SDK, a AWS Encryption SDK para JavaScript oferece atributos avançados de proteção de dados. Os recursos incluem [criptografia de envelope](#), [dados autenticados adicionais](#) (AAD) e [pacotes de algoritmos](#) de chave simétrica segura e autenticada, como o AES-GCM de 256 bits com derivação de chave e assinatura.

Todas as implementações específicas de linguagem do AWS Encryption SDK são projetadas para serem interoperáveis, sujeitas às restrições da linguagem. Para obter detalhes sobre restrições de linguagem para JavaScript, consulte [the section called “Compatibilidade”](#).

Saiba mais

- Para obter detalhes sobre como programar com o AWS Encryption SDK para JavaScript, consulte o repositório [aws-encryption-sdk-javascript](#) no GitHub.
- Para obter exemplos de programação, consulte [the section called “Exemplos”](#) e os módulos [example-browser](#) e [example-node](#) no repositório [aws-encryption-sdk-javascript](#).
- Para obter um exemplo real do uso do AWS Encryption SDK para JavaScript para criptografar dados em uma aplicação Web, consulte [Como habilitar a criptografia em um navegador com o AWS Encryption SDK para JavaScript e Node.js](#) no Blog de segurança da AWS.

Tópicos

- [Compatibilidade do AWS Encryption SDK para JavaScript](#)
- [Instalar o AWS Encryption SDK para JavaScript](#)
- [Módulos no AWS Encryption SDK para JavaScript](#)
- [Exemplos do AWS Encryption SDK para JavaScript](#)

Compatibilidade do AWS Encryption SDK para JavaScript

O AWS Encryption SDK para JavaScript é projetado para ser interoperável com outras implementações de linguagem do AWS Encryption SDK. Na maioria dos casos, você pode criptografar dados com o AWS Encryption SDK para JavaScript e descriptografá-los com qualquer outra implementação de linguagem, incluindo a [Interface de linha de comando do AWS Encryption SDK](#). E você pode usar o AWS Encryption SDK para JavaScript para descriptografar [mensagens criptografadas](#) produzidas por outras implementações de linguagem do AWS Encryption SDK.

No entanto, ao usar o AWS Encryption SDK para JavaScript, você precisa estar ciente de alguns problemas de compatibilidade na implementação da linguagem JavaScript e em navegadores da Web.

Além disso, ao usar implementações de linguagem diferentes, configure provedores de chaves mestras, chaves mestras e tokens de autenticação compatíveis. Para obter mais detalhes, consulte [Compatibilidade dos tokens de autenticação](#).

Compatibilidade do AWS Encryption SDK para JavaScript

A implementação do AWS Encryption SDK do JavaScript difere de outras implementações de linguagem nas seguintes maneiras:

- A operação de criptografia do AWS Encryption SDK para JavaScript não retorna texto cifrado não emoldurado. No entanto, o AWS Encryption SDK para JavaScript descriptografará texto cifrado e não emoldurado retornado por outras implementações de linguagem do AWS Encryption SDK.
- Começando com o Node.js versão 12.9.0, o Node.js é compatível com as seguintes opções de empacotamento de chave RSA:
 - OAEP com SHA1, SHA256, SHA384 ou SHA512
 - OAEP com SHA1 e MGF1 com SHA1
 - PKCS1v15

- Antes da versão 12.9.0, o Node.js era compatível apenas com as seguintes opções de empacotamento de chave RSA:
 - OAEP com SHA1 e MGF1 com SHA1
 - PKCS1v15

Compatibilidade do navegador

Alguns navegadores da Web não são compatíveis com operações de criptografia básicas exigidas pelo AWS Encryption SDK para JavaScript. Você pode compensar algumas das operações ausentes configurando um fallback para a API WebCrypto que o navegador implementa.

Limitações de navegador da Web

As seguintes limitações são comuns a todos os navegadores da Web:

- A API WebCrypto não é compatível com o empacotamento de chaves PKCS1v15.
- Os navegadores não são compatíveis com chaves de 192 bits.

Operações de criptografia necessárias

O AWS Encryption SDK para JavaScript requer as seguintes operações em navegadores da Web. Se um navegador não for compatível com estas operações, ele será incompatível com o AWS Encryption SDK para JavaScript.

- O navegador deve incluir `crypto.getRandomValues()`, que é um método para gerar valores de criptografia aleatórios. Para obter informações sobre as versões do navegador da Web compatíveis com o `crypto.getRandomValues()`, consulte [Posso usar crypto.getRandomValues\(\)?](#)

Fallback necessário

O AWS Encryption SDK para JavaScript requer as seguintes bibliotecas e operações em navegadores da Web. Se você oferecer suporte a um navegador da Web que não atenda a esses requisitos, deverá configurar um fallback. Caso contrário, as tentativas de usar o AWS Encryption SDK para JavaScript com o navegador falharão.

- A API WebCrypto, que executa operações de criptografia básicas em aplicativos Web, não está disponível para todos os navegadores. Para obter informações sobre as versões do navegador da Web compatíveis com a criptografia da Web, consulte [Posso usar criptografia da Web?](#).
- As versões modernas do navegador da Web Safari não são compatíveis com a criptografia AES-GCM de zero bytes, exigida pelo AWS Encryption SDK. Se o navegador implementar a API WebCrypto, mas não puder usar o AES-GCM para criptografar zero bytes, o AWS Encryption SDK para JavaScript usará a biblioteca de fallback somente para criptografia de zero bytes. Ele usa a API WebCrypto para todas as outras operações.

Para configurar um fallback para qualquer limitação, adicione as seguintes instruções ao seu código. Na função [configureFallback](#) especifique uma biblioteca que seja compatível com os recursos ausentes. O exemplo a seguir usa a Microsoft Research JavaScript Cryptography Library (`msrCrypto`), mas você pode substituí-la por uma biblioteca compatível. Para obter um exemplo completo, consulte [fallback.ts](#).

```
import { configureFallback } from '@aws-crypto/client-browser'  
configureFallback(msrCrypto)
```

Instalar o AWS Encryption SDK para JavaScript

O AWS Encryption SDK para JavaScript consiste em uma coleção de módulos interdependentes. Vários dos módulos são apenas coleções de módulos projetados para funcionar em conjunto. Alguns módulos são projetados para funcionar de forma independente. Alguns módulos são necessários para todas as implementações; alguns outros são necessários apenas para casos especiais. Para obter informações sobre os módulos no AWS Encryption SDK para JavaScript, consulte [Módulos no AWS Encryption SDK para JavaScript](#) e o arquivo README .md em cada um dos módulos no repositório [aws-encryption-sdk-javascript](#) no GitHub.

Note

Todas as versões do AWS Encryption SDK para JavaScript anteriores à 2.0.0 estão na fase de [fim do suporte](#).

Você pode atualizar com segurança a partir da versão 2.0.x e posteriores até a versão mais recente do AWS Encryption SDK para JavaScript sem realizar alterações no código ou nos dados. No entanto, os [novos atributos de segurança](#) introduzidos na versão 2.0.x não são compatíveis com versões anteriores. Para atualizar a partir de versões anteriores à 1.7.x até a versão 2.0. x e posteriores, primeiro será necessário atualizar para a versão 1.x mais

recente do AWS Encryption SDK para JavaScript. Para obter mais detalhes, consulte [Como migrar seu AWS Encryption SDK](#).

Para instalar os módulos, use o [gerenciador de pacotes npm](#).

Por exemplo, para instalar o módulo `client-node`, que inclui todos os módulos que você precisa programar com o AWS Encryption SDK para JavaScript no Node.js, use o seguinte comando.

```
npm install @aws-crypto/client-node
```

Para instalar o módulo `client-browser`, que inclui todos os módulos que você precisa programar com o AWS Encryption SDK para JavaScript no navegador, use o seguinte comando.

```
npm install @aws-crypto/client-browser
```

Para obter exemplos de como usar o AWS Encryption SDK para JavaScript, consulte os exemplos nos módulos `example-node` e `example-browser` no repositório [aws-encryption-sdk-javascript](#) no GitHub.

Módulos no AWS Encryption SDK para JavaScript

Os módulos no AWS Encryption SDK para JavaScript facilitam a instalação do código que você precisa para seus projetos.

Módulos para JavaScript Node.js

[nó do cliente](#)

Inclui todos os módulos de que você precisa para programar com o AWS Encryption SDK para JavaScript no Node.js.

[caching-materials-manager-node](#)

Exporta funções compatíveis com o recurso de [armazenamento em cache de chaves de dados](#) no AWS Encryption SDK para JavaScript em Node.js.

[decrypt-node](#)

Exporta funções que descriptografam e verificam mensagens criptografadas que representam dados e streams de dados. Incluído no módulo `client-node`.

[encrypt-node](#)

Exporta funções que criptografam e assinam diferentes tipos de dados. Incluído no módulo `client-node`.

[example-node](#)

Exporta exemplos práticos de programação com o AWS Encryption SDK para JavaScript em Node.js. Inclui exemplos de diferentes tipos de tokens de autenticação e diferentes tipos de dados.

[hkdf-node](#)

Exporta uma [Função de derivação de chave baseada em HMAC](#) (HKDF) que o AWS Encryption SDK para JavaScript em Node.js usa em pacotes de algoritmos específicos. O AWS Encryption SDK para JavaScript no navegador usa a função HKDF nativa na API WebCrypto.

[integration-node](#)

Define testes que verificam se o AWS Encryption SDK para JavaScript em Node.js é compatível com outras implementações de linguagem do AWS Encryption SDK.

[kms-keyring-node](#)

Exporta funções compatíveis com tokens de autenticação do AWS KMS em Node.js.

[raw-aes-keyring-node](#)

Exporta funções que são compatíveis com [tokens de autenticação brutos do AES](#) no Node.js.

[raw-rsa-keyring-node](#)

Exporta funções compatíveis com [tokens de autenticação brutos do RSA](#) em Node.js.

Módulos para Navegador JavaScript

[client-browser](#)

Inclui todos os módulos de que você precisa para programar com o AWS Encryption SDK para JavaScript no navegador.

[caching-materials-manager-browser](#)

Exporta funções compatíveis com o atributo de [armazenamento em cache de chaves de dados](#) para JavaScript no navegador.

[decrypt-browser](#)

Exporta funções que descriptografam e verificam mensagens criptografadas que representam dados e streams de dados.

[encrypt-browser](#)

Exporta funções que criptografam e assinam diferentes tipos de dados.

[example-browser](#)

Exemplos práticos de programação com o AWS Encryption SDK para JavaScript no navegador. Inclui exemplos de diferentes tipos de tokens de autenticação e diferentes tipos de dados.

[integration-browser](#)

Define testes que verificam se o AWS Encryption SDK for JavaScript no navegador é compatível com outras implementações de linguagem do AWS Encryption SDK.

[kms-keyring-browser](#)

Exporta funções compatíveis com [tokens de autenticação do AWS KMS](#) no navegador.

[raw-aes-keyring-browser](#)

Exporta funções compatíveis com [tokens de autenticação brutos do AES](#) no navegador.

[raw-rsa-keyring-browser](#)

Exporta funções compatíveis com [tokens de autenticação brutos do RSA](#) no navegador.

Módulos para todas as implementações

[cache-material](#)

É compatível com o recurso de [armazenamento em cache de chaves de dados](#). Fornece código para montar o material de criptografia que é armazenado em cache com cada chave de dados.

[kms-keyring](#)

Exporta funções compatíveis com [tokens de autenticação do KMS](#).

[material-management](#)

Implementa o [gerenciador de material de criptografia](#) (CMM).

[raw-keyring](#)

Exporta funções necessárias para tokens de autenticação brutos do AES e do RSA.

[serialize](#)

Exporta funções que o SDK usa para serializar sua saída.

[web-crypto-backend](#)

Exporta funções que usam a API WebCrypto AWS Encryption SDK para JavaScript no navegador.

Exemplos do AWS Encryption SDK para JavaScript

Os exemplos a seguir mostram como usar o AWS Encryption SDK para JavaScript para criptografar e descriptografar dados.

Você pode encontrar mais exemplos de uso do AWS Encryption SDK para JavaScript nos módulos [example-node](#) e [example-browser](#) no repositório [aws-encryption-sdk-javascript](#) no GitHub. Esses módulos de exemplo não são instalados quando você instala os módulos `client-browser` ou `client-node`.

Consulte os exemplos de código completos: nó: [kms_simple.ts](#), navegador: [kms_simple.ts](#)

Tópicos

- [Criptografar dados com um token de autenticação do AWS KMS](#)
- [Descriptografar dados com um token de autenticação do AWS KMS](#)

Criptografar dados com um token de autenticação do AWS KMS

O exemplo a seguir mostra como usar o AWS Encryption SDK para JavaScript para criptografar e descriptografar uma string curta ou uma matriz de bytes.

Este exemplo apresenta o [token de autenticação do AWS KMS](#), um tipo de token de autenticação que usa uma AWS KMS key para gerar e criptografar chaves de dados. Para obter ajuda na criação de uma AWS KMS key, consulte [Criação de chaves](#) no Guia do desenvolvedor do AWS Key Management Service. Para obter ajuda para identificar a AWS KMS keys em um token de autenticação do AWS KMS, consulte [Identificação AWS KMS keys em um AWS KMS chaveiro](#)

Etapa 1: Criar o token de autenticação.

Criar um token de autenticação do AWS KMS para criptografia.

Ao criptografar com um token de autenticação do AWS KMS, você deve especificar uma chave geradora, ou seja, uma AWS KMS key do que é usada para gerar a chave de dados de texto simples e criptografá-la. Você também pode especificar zero ou mais chaves adicionais que criptografam a mesma chave de dados de texto simples. O token de autenticação retorna a chave de dados de texto simples e uma cópia criptografada dessa chave de dados para cada AWS KMS key no token de autenticação, incluindo a chave geradora. Para descriptografar os dados, você precisa descriptografar qualquer uma das chaves de dados criptografadas.

Para especificar as AWS KMS keys para um token de autenticação de criptografia no AWS Encryption SDK para JavaScript, você pode usar [qualquer identificador de chave do AWS KMS compatível](#). Este exemplo usa uma chave geradora, que é identificada por seu [ARN de alias](#), e uma chave adicional, que é identificada por um [ARN de chave](#).

Note

Se você planeja reutilizar o token de autenticação do AWS KMS para descriptografar, deverá usar ARNs de chave para identificar as AWS KMS keys no token de autenticação.

Antes de executar esse código, substitua os identificadores da AWS KMS key de exemplo por identificadores válidos. Você deve ter as [permissões necessárias para usar as AWS KMS keys](#) no token de autenticação.

JavaScript Browser

Comece fornecendo suas credenciais para o navegador. Os exemplos do AWS Encryption SDK para JavaScript usam o [webpack.DefinePlugin](#), que substitui as constantes de credenciais por suas credenciais reais. Mas você pode usar qualquer método para fornecer suas credenciais. Depois, use as credenciais para criar um cliente do AWS KMS.

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

const clientProvider = getClient(KMS, {
  credentials: {
    accessKeyId,
```

```
    secretAccessKey,  
    sessionToken  
  }  
})
```

Especifique as AWS KMS keys para a chave geradora e a chave adicional. Crie um token de autenticação do AWS KMS usando o cliente do AWS KMS e as AWS 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 })
```

Etapa 2: Definir o contexto de criptografia.

Um [contexto de criptografia](#) são dados autenticados adicionais arbitrários e que não são secretos. Quando você fornece um contexto de criptografia na criptografia, o AWS Encryption SDK vincula de forma criptográfica o contexto ao texto cifrado, de forma que o mesmo contexto de criptografia seja necessário para descriptografar os dados. O uso de um contexto de criptografia é opcional, mas o recomendamos como uma melhor prática.

Crie um objeto simples que inclua os pares de contexto de criptografia. A chave e o valor em cada par devem ser uma string.

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'
}
```

Etapa 3: Criptografar os dados.

Para criptografar os dados de texto simples, chame a função `encrypt`. Passe o token de autenticação do AWS KMS, os dados de texto simples e o contexto de criptografia.

A função `encrypt` retorna uma [mensagem criptografada](#) (`result`) que contém os dados criptografados, as chaves de dados criptografados e metadados importantes, incluindo o contexto de criptografia e a assinatura.

Você pode [descriptografar essa mensagem criptografada](#) usando o AWS Encryption SDK em qualquer linguagem de programação compatível.

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 })
```

Descriptografar dados com um token de autenticação do AWS KMS

Você pode usar o AWS Encryption SDK para JavaScript para descriptografar a mensagem criptografada e recuperar os dados originais.

Neste exemplo, descriptografamos os dados que criptografamos no exemplo [the section called “Criptografar dados com um token de autenticação do AWS KMS”](#).

Etapa 1: Criar o token de autenticação.

Para descriptografar os dados, passe a [mensagem criptografada](#) (`result`) que a função `encrypt` retornou. A mensagem criptografada inclui os dados criptografados, as chaves de dados criptografadas e metadados importantes, incluindo o contexto de criptografia e a assinatura.

Você também deve especificar um [token de autenticação do AWS KMS](#) ao descriptografar. Você pode usar o mesmo token de autenticação usado para criptografar os dados ou um token de autenticação diferente. Para ter êxito, pelo menos uma AWS KMS key no token de autenticação de descriptografia deve poder descriptografar uma das chaves de dados criptografadas na mensagem criptografada. Como nenhuma chave de dados é gerada, você não precisa especificar uma chave geradora em um token de autenticação de descriptografia. Se você fizer isso, a chave geradora e as chaves adicionais serão tratadas da mesma maneira.

Para especificar uma AWS KMS key para um token de autenticação de descriptografia no AWS Encryption SDK para JavaScript, você deve usar o [ARN da chave](#). Caso contrário, a AWS KMS key não é reconhecida. Para obter ajuda para identificar a AWS KMS keys em um token de autenticação do AWS KMS, consulte [Identificação AWS KMS keys em um AWS KMS chaveiro](#)

Note

Se você usar o mesmo token de autenticação para criptografar e descriptografar, use ARNs de chave para identificar as AWS KMS keys no token de autenticação.

Neste exemplo, criamos um token de autenticação que inclui apenas uma das AWS KMS keys no token de autenticação de criptografia. Antes de executar esse código, substitua o ARN da chave de exemplo por um válido. Você deve ter a permissão `kms:Decrypt` na AWS KMS key.

JavaScript Browser

Comece fornecendo suas credenciais para o navegador. Os exemplos do AWS Encryption SDK para JavaScript usam o [webpack.DefinePlugin](#), que substitui as constantes de credenciais por suas credenciais reais. Mas você pode usar qualquer método para fornecer suas credenciais. Depois, use as credenciais para criar um cliente do AWS KMS.

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

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

Em seguida, crie um token de autenticação do AWS KMS usando o cliente do AWS KMS. Este exemplo usa apenas uma das AWS KMS keys do token de autenticação de criptografia.

```
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 })
```

Etapa 2: Descriptografar os dados.

Chame a função `decrypt`. Passe o token de autenticação de descriptografia que você acabou de criar (`keyring`) e a [mensagem criptografada](#) que a função `encrypt` retornou (`result`). O AWS Encryption SDK usa o token de autenticação para descriptografar uma das chaves de dados criptografadas. Ele usa a chave de dados de texto simples para descriptografar os dados.

Se a chamada for bem-sucedida, o campo `plaintext` conterá os dados de texto simples (descriptografados). O campo `messageHeader` contém metadados sobre o processo de descriptografia, incluindo o contexto de criptografia usado para descriptografar os dados.

JavaScript Browser

```
const { plaintext, messageHeader } = await decrypt(keyring, result)
```

JavaScript Node.js

```
const { plaintext, messageHeader } = await decrypt(keyring, result)
```

Etapa 3: Verificar o contexto de criptografia.

O [contexto de criptografia](#) que foi usado para descriptografar os dados é incluído no cabeçalho da mensagem (`messageHeader`) que a função `decrypt` retorna. Antes do aplicativo retornar os dados de texto simples, verifique se o contexto de criptografia fornecido durante a criptografia está incluído no contexto de criptografia usado ao descriptografar. Uma incompatibilidade pode indicar que os dados foram adulterados ou que você não descriptografou o texto cifrado correto.

Ao verificar o contexto de criptografia, não exija uma correspondência exata. Ao usar um algoritmo de criptografia com a assinatura, o [gerenciador de material de criptografia](#) (CMM) adiciona a chave de assinatura pública ao contexto de criptografia antes de criptografar a mensagem. Mas todos os pares de contexto de criptografia que você enviou devem ser incluídos no contexto de criptografia que foi retornado.

Primeiro, obtenha o contexto de criptografia do cabeçalho da mensagem. Depois, verifique se cada par de chave-valor no contexto de criptografia original (`context`) corresponde a um par de chave-valor no contexto de criptografia retornado (`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')
  })
```

Se a verificação de contexto de criptografia for bem-sucedida, você poderá retornar os dados de texto simples.

AWS Encryption SDK for Python

Este tópico explica como instalar e usar o AWS Encryption SDK for Python. Para obter detalhes sobre a programação com o AWS Encryption SDK for Python, consulte o [aws-encryption-sdk-python](#) repositório em GitHub. Para obter a documentação da API, consulte [Ler os documentos](#).

Tópicos

- [Pré-requisitos](#)
- [Instalação](#)
- [Código de exemplo do AWS Encryption SDK for Python](#)

Pré-requisitos

Antes de instalar o AWS Encryption SDK for Python, verifique se você tem os seguintes pré-requisitos.

Uma versão compatível do Python

O Python 3.8 ou posterior é exigido pelas AWS Encryption SDK for Python versões 3.2.0 e posteriores.

As versões anteriores do AWS Encryption SDK oferecem suporte ao Python 2.7 e ao Python 3.4 e posteriores, mas recomendamos que você use a versão mais recente do AWS Encryption SDK.

Para fazer download do Python, consulte [Downloads do Python](#).

A ferramenta de instalação do pip para Python

O Python 3.6 e versões posteriores incluem pip, embora você possa querer atualizá-lo. Para obter mais informações sobre a atualização ou a instalação do pip consulte [Instalação](#) na documentação do pip.

Instalação

Instalar a versão mais recente do AWS Encryption SDK for Python.

Note

Todas as versões AWS Encryption SDK for Python anteriores à 3.0.0 estão em [end-of-supportfase](#).

Você pode atualizar com segurança a partir da versão 2.0.x e posteriores até a versão mais recente do AWS Encryption SDK sem realizar alterações no código ou nos dados. No entanto, os [novos atributos de segurança](#) introduzidos na versão 2.0.x não são compatíveis com versões anteriores. Para atualizar a partir de versões anteriores à 1.7.x até a versão 2.0.x e posteriores, primeiro será necessário atualizar para a versão 1.x mais recente do AWS Encryption SDK. Para obter detalhes, consulte [Como migrar seu AWS Encryption SDK](#).

Use `pip` para instalar o AWS Encryption SDK for Python, conforme mostrado nos exemplos a seguir.

Para instalar a versão mais recente

```
pip install aws-encryption-sdk
```

Para obter mais detalhes sobre o uso do `pip` para instalar e atualizar pacotes, consulte [Instalação de pacotes](#).

AWS Encryption SDK for Python Isso requer a [biblioteca de criptografia \(pyca/cryptography\)](#) em todas as plataformas. Todas as versões do `pip` instalam e criam automaticamente a biblioteca `cryptography` no Windows. O `pip` 8.1 e versões posteriores instala e compila `cryptography` automaticamente no Linux. Se você usar uma versão anterior do `pip` e seu ambiente Linux não tiver as ferramentas necessárias para criar a biblioteca `cryptography`, será necessário instalá-las. Para obter mais informações, consulte [Building Cryptography on Linux](#).

As versões 1.10.0 e 2.5.0 do AWS Encryption SDK for Python fixam a dependência [criptográfica](#) entre 2.5.0 e 3.3.2. Outras versões do AWS Encryption SDK for Python instalam a versão mais recente da criptografia. Se você precisar de uma versão do `cryptography` posterior à 3.3.2, recomendamos que use a versão principal mais recente do AWS Encryption SDK for Python.

Para obter a versão de desenvolvimento mais recente do AWS Encryption SDK for Python, acesse o [aws-encryption-sdk-python](#) repositório em GitHub.

Depois de instalar o AWS Encryption SDK for Python, comece examinando o [código de exemplo do Python](#) neste guia.

Código de exemplo do AWS Encryption SDK for Python

Os exemplos a seguir mostram como usar o AWS Encryption SDK for Python para criptografar e descriptografar dados.

Esses exemplos mostram como usar a versão 2.0.x e versões posteriores do AWS Encryption SDK for Python. Para obter exemplos que usam versões anteriores, encontre sua versão na lista de [Versões](#) do repositório [aws-encryption-sdk-python](#) no GitHub.

Tópicos

- [Criptografar e descriptografar strings](#)
- [Criptografar e descriptografar streams de bytes](#)
- [Criptografia e descriptografia de fluxos de bytes com vários provedores de chaves mestras](#)
- [Usar armazenamento em cache de chaves de dados para criptografar mensagens](#)

Criptografar e descriptografar strings

O exemplo a seguir mostra como usar o AWS Encryption SDK para criptografar e descriptografar strings. Este exemplo usa uma AWS KMS key no [AWS Key Management Service\(AWS KMS\)](#) como a chave mestra.

Ao realizar a criptografia, o construtor `StrictAwsKmsMasterKeyProvider` usa um ID de chave, um ARN de chave, um nome de alias ou um ARN de alias. Ao descriptografar, ele [exige um ARN de chave](#). Nesse caso, como o parâmetro `keyArn` é usado para criptografar e descriptografar, seu valor deve ser um ARN de chave. Para obter mais informações sobre IDs de chaves AWS KMS, consulte [Identificadores de chave](#) no Guia do Desenvolvedor do AWS Key Management Service.

```
# 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_REQUIRE_DECRYPT)

    # 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()
)
```

Criptografar e descriptografar streams de bytes

O exemplo a seguir mostra como usar o AWS Encryption SDK para criptografar e descriptografar streams de bytes. Este exemplo não usa a AWS. Ele usa um provedor de chaves mestras efêmero estático.

Ao criptografar, este exemplo usa um pacote de algoritmos alternativo sem [assinaturas digitais](#) (AES_256_GCM_HKDF_SHA512_COMMIT_KEY). Esse pacote de algoritmos é adequado quando os usuários que estão criptografando e descriptografando dados são igualmente confiáveis. Em seguida, ao descriptografar, o exemplo usa o modo de streaming decrypt-unsigned, que falhará se encontrar texto cifrado assinado. O modo de streaming decrypt-unsigned foi introduzido nas versões 1.9.x e 2.2.x. do AWS Encryption SDK.

```
# 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 unsigned 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
```

Criptografia e descriptografia de fluxos de bytes com vários provedores de chaves mestras

O exemplo a seguir mostra como usar o AWS Encryption SDK com mais de um provedor de chaves mestras. O uso de mais de um provedor de chaves mestras criará redundância se um provedor de chaves mestras não estiver disponível para a descriptografia. Este exemplo usa uma AWS KMS key e um par de chaves RSA como as chaves mestras.

Este exemplo criptografa com o [pacote de algoritmos padrão](#), que inclui uma [assinatura digital](#). Durante o streaming, o AWS Encryption SDK libera texto simples após as verificações de integridade, mas antes de verificar a assinatura digital. Para evitar o uso do texto simples até que a assinatura seja verificada, este exemplo armazena o texto simples em buffer e o grava no disco somente após a conclusão da descriptografia e da verificação.

```
# 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_REQUIRE_DECRYPT)

# 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 verification 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 verification 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)
```

Usar armazenamento em cache de chaves de dados para criptografar mensagens

O exemplo a seguir mostra como usar o [armazenamento em cache de chaves de dados](#) no AWS Encryption SDK for Python. O objetivo dele é mostrar como configurar uma instância do [cache local](#) (LocalCryptoMaterialsCache) com o valor de capacidade necessária e uma instância do [gerenciador de materiais criptográficos de armazenamento em cache](#) (CMM de armazenamento em cache) com [limites de segurança de cache](#).

Este exemplo muito básico cria uma função que criptografa um string fixa. Ele permite especificar uma AWS KMS key, o tamanho de cache necessário (capacidade) e um valor de idade máxima. Para obter um exemplo real mais complexo de armazenamento em cache de chaves de dados, consulte [Exemplo de código de armazenamento em cache de chaves de dados](#).

Embora seja opcional, esse exemplo também usa um [contexto de criptografia](#) como dados autenticados adicionais. Quando você descriptografar dados que foram criptografados com um contexto de criptografia, certifique-se de que seu aplicativo verifique se o contexto de criptografia é o esperado antes de retornar os dados em texto simples para o seu chamador. Um contexto de criptografia é um elemento de melhor prática de qualquer operação de criptografia ou descriptografia, mas ele exerce um papel especial no armazenamento em cache de chaves de dados. Para obter detalhes, consulte [Contexto de criptografia: como selecionar entradas 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_REQUIRE_DECRYPT)

    # 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 da linha de comando do AWS Encryption SDK

A interface de linha de comando do AWS Encryption SDK (CLI de criptografia da AWS) permite que você use o AWS Encryption SDK para criptografar e descriptografar dados interativamente na linha de comando e em scripts. Você não precisa ter competência em criptografia ou em programação.

Note

Todas as versões da CLI de criptografia da AWS anteriores à 4.0.0 estão na fase de [fim do suporte](#).

Você pode atualizar com segurança a partir da versão 2.1.x e posteriores até a versão mais recente da CLI de criptografia da AWS sem realizar alterações no código ou nos dados. No entanto, os [novos atributos de segurança](#) introduzidos na versão 2.1.x não são compatíveis com versões anteriores. Para atualizar a partir da versão 1.7.x ou anteriores, primeiro será necessário atualizar para a versão 1.x mais recente da CLI de criptografia da AWS. Para obter mais detalhes, consulte [Como migrar seu AWS Encryption SDK](#).

Os novos atributos de segurança foram lançados originalmente nas versões 1.7.x e 2.0.x da CLI de criptografia da AWS. No entanto, a versão 1.8x da CLI de criptografia da AWS substituiu a versão 1.7.x e a versão 2.1.x da CLI de criptografia da AWS substituiu a 2.0.x. Para obter detalhes, consulte o [boletim de segurança](#) pertinente no repositório [aws-encryption-sdk-cli](#) no GitHub.

Como todas as implementações do AWS Encryption SDK, a CLI de criptografia da AWS oferece atributos avançados de proteção de dados. Os atributos incluem [criptografia envelopada](#), dados autenticados adicionais (AAD) e [pacotes de algoritmos](#) de chave simétrica segura e autenticada, como o AES-GCM de 256 bits com derivação de chave, [confirmação de chave](#) e assinatura.

A CLI de criptografia da AWS é baseada no [AWS Encryption SDK for Python](#) e é compatível com o Linux, o macOS e o Windows. Você pode executar comandos e scripts para criptografar e descriptografar seus dados em seu shell preferido no Linux ou no macOS, em uma janela de prompt de comando (cmd.exe) no Windows e em um console do PowerShell em qualquer sistema.

Todas as implementações específicas de linguagens do AWS Encryption SDK, incluindo a CLI de criptografia da AWS, são interoperáveis. Por exemplo, você pode criptografar dados com o [AWS Encryption SDK for Java](#) e descriptografá-los com a CLI de criptografia da AWS.

Este tópico apresenta a CLI de criptografia da AWS, explica como instalar e usá-la e fornece vários exemplos para ajudar você a começar. Para um início rápido, consulte [Como criptografar e descriptografar seus dados com a CLI de criptografia da AWS](#) no blog de segurança da AWS. Para obter informações mais detalhadas, consulte [Leia os documentos](#) e junte-se a nós no desenvolvimento da CLI de criptografia da AWS no repositório [aws-encryption-sdk-cli](#) no GitHub.

Desempenho

A CLI de criptografia da AWS foi criada com base no AWS Encryption SDK for Python. Cada vez que executa a CLI, você inicia uma nova instância do runtime do Python. Para melhorar o desempenho, sempre que possível, use um único comando em vez de uma série de comandos independentes. Por exemplo, execute um comando que processe os arquivos em um diretório de forma recursiva, em vez de executar comandos separados para cada arquivo.

Tópicos

- [Instalando a interface de linha de AWS Encryption SDK comando](#)
- [Como usar a CLI de criptografia da AWS](#)
- [Exemplos da CLI de criptografia da AWS](#)
- [Referência da sintaxe e de parâmetros da CLI do AWS Encryption SDK](#)
- [Versões da CLI de criptografia da AWS](#)

Instalando a interface de linha de AWS Encryption SDK comando

Este tópico explica como instalar a CLI AWS de criptografia. Para obter informações detalhadas, consulte o [aws-encryption-sdk-cli](#) repositório GitHub e [leia os documentos](#).

Tópicos

- [Instalar os pré-requisitos](#)
- [Instalando e atualizando a CLI AWS de criptografia](#)

Instalar os pré-requisitos

A CLI de AWS criptografia é baseada no AWS Encryption SDK for Python. Para instalar a CLI do AWS Encryption, você precisa do Python e da ferramenta de gerenciamento de pacotes `pip` do Python. O Python e o `pip` estão disponíveis em todas as plataformas compatíveis.

Instale os seguintes pré-requisitos antes de instalar a CLI de criptografia AWS ,

Python

O Python 3.8 ou posterior é exigido pelas versões 4.2.0 e posteriores do Encryption AWS CLI.

As versões anteriores da CLI de AWS criptografia oferecem suporte ao Python 2.7 e 3.4 e versões posteriores, mas recomendamos que você use a versão mais recente da CLI de criptografia. AWS

O Python está incluído na maioria das instalações do Linux e do macOS, mas é necessário atualizar para o Python 3.6 ou versões posteriores. É recomendável usar a versão mais recente do Python. No Windows, você precisa instalar o Python: ele não é instalado por padrão. Para fazer download do Python, consulte [Downloads do Python](#).

Para determinar se o Python está instalado, na linha de comando, digite:

```
python
```

Para verificar a versão do Python, use o parâmetro `-V` (V maiúsculo).

```
python -V
```

No Windows, depois de instalar o Python, adicione o caminho para o arquivo `Python.exe` ao valor da variável de ambiente `Path`.

Por padrão, o Python é instalado em todos os diretórios de usuário ou em um diretório de perfil de usuário (`$home` ou `%userprofile%`) no subdiretório `AppData\Local\Programs\Python`. Para encontrar o local do arquivo `Python.exe` no sistema, verifique uma das seguintes chaves de registro. Você pode usar PowerShell para pesquisar o registro.

```
PS C:\> dir HKLM:\Software\Python\PythonCore\version\InstallPath
# -oI-
```

```
PS C:\> dir HKCU:\Software\Python\PythonCore\version\InstallPath
```

pip

pip é o gerenciador de pacotes do Python. Para instalar a CLI de AWS criptografia e suas dependências, você precisa da pip versão 8.1 ou posterior. Para obter ajuda para instalar ou atualizar o pip, consulte [Instalação](#) na documentação do pip.

Nas instalações do Linux, as versões pip anteriores à 8.1 não podem criar a biblioteca de criptografia exigida pela CLI de AWS criptografia. Se você optar por não atualizar sua versão do pip, poderá instalar as ferramentas de compilação separadamente. Para obter mais informações, consulte [Criação de criptografia no Linux](#).

AWS Command Line Interface

O AWS Command Line Interface (AWS CLI) é necessário somente se você estiver usando AWS KMS keys in AWS Key Management Service (AWS KMS) com a CLI de AWS criptografia. Se você estiver usando um [provedor de chave mestra](#) diferente, AWS CLI isso não é obrigatório.

Para usar AWS KMS keys com a CLI de AWS criptografia, você precisa [instalar](#) e [configurar o](#). AWS CLI A configuração disponibiliza as credenciais que você usa para autenticar para a AWS KMS AWS CLI de criptografia.

Instalando e atualizando a CLI AWS de criptografia

Instale a versão mais recente da CLI AWS de criptografia. [Quando você usa pip para instalar a CLI de AWS criptografia, ela instala automaticamente as bibliotecas de que a CLI precisa, incluindo a biblioteca de criptografia Python e a. AWS Encryption SDK for PythonAWS SDK for Python \(Boto3\)](#)

Note

[Versões da CLI de AWS criptografia anteriores à 4.0.0 estão em fase. end-of-support](#)

Você pode atualizar com segurança a partir da versão 2.1.x e posteriores até a versão mais recente da CLI de criptografia da AWS sem realizar alterações no código ou nos dados. No entanto, os [novos atributos de segurança](#) introduzidos na versão 2.1.x não são compatíveis com versões anteriores. Para atualizar a partir da versão 1.7. x ou anterior, você deve primeiro atualizar para a última 1. versão x da CLI AWS de criptografia. Para obter detalhes, consulte [Como migrar seu AWS Encryption SDK](#).

Novos recursos de segurança foram lançados originalmente nas versões 1.7 do AWS Encryption CLI. x e 2.0. x. No entanto, a versão AWS 1.8 do Encryption CLI. x substitui a

versão 1.7. x e CLI de AWS criptografia 2.1. x substitui 2.0. x. Para obter detalhes, consulte a [consultoria de segurança](#) relevante no [aws-encryption-sdk-cli](#) repositório em GitHub.

Para instalar a versão mais recente da CLI AWS de criptografia

```
pip install aws-encryption-sdk-cli
```

Para atualizar para a versão mais recente da CLI de AWS criptografia

```
pip install --upgrade aws-encryption-sdk-cli
```

Para encontrar os números de versão da sua CLI de AWS criptografia e AWS Encryption SDK

```
aws-encryption-cli --version
```

A saída lista os números de versão de ambas as bibliotecas.

```
aws-encryption-sdk-cli/2.1.0 aws-encryption-sdk/2.0.0
```

Para atualizar para a versão mais recente da CLI de AWS criptografia

```
pip install --upgrade aws-encryption-sdk-cli
```

A instalação da CLI de AWS criptografia também instala a versão mais recente do AWS SDK for Python (Boto3), se ainda não estiver instalada. Se o Boto3 estiver instalado, o instalador verifica a versão do Boto3 e a atualiza, se necessário.

Para encontrar sua versão instalada do Boto3

```
pip show boto3
```

Para atualizar para a versão mais recente do Boto3

```
pip install --upgrade boto3
```

Para instalar a versão da CLI de AWS criptografia atualmente em desenvolvimento, consulte o [aws-encryption-sdk-cli](#) repositório em GitHub

Para obter mais detalhes sobre o uso do `pip` para instalar e atualizar pacotes do Python, consulte a [documentação do pip](#).

Como usar a CLI de criptografia da AWS

Este tópico explica como usar os parâmetros na CLI de criptografia da AWS. Para ver exemplos, consulte [Exemplos da CLI de criptografia da AWS](#). Para obter a documentação completa, consulte [Leia os documentos](#). A sintaxe mostrada nesses exemplos é para a versão 2.1.x e versões posteriores da CLI de criptografia da AWS.

Note

Todas as versões da CLI de criptografia da AWS anteriores à 4.0.0 estão na fase de [fim do suporte](#).

Você pode atualizar com segurança a partir da versão 2.1.x e posteriores até a versão mais recente da CLI de criptografia da AWS sem realizar alterações no código ou nos dados. No entanto, os [novos atributos de segurança](#) introduzidos na versão 2.1.x não são compatíveis com versões anteriores. Para atualizar a partir da versão 1.7.x ou anteriores, primeiro será necessário atualizar para a versão 1.x mais recente da CLI de criptografia da AWS. Para obter mais detalhes, consulte [Como migrar seu AWS Encryption SDK](#).

Os novos atributos de segurança foram lançados originalmente nas versões 1.7.x e 2.0.x da CLI de criptografia da AWS. No entanto, a versão 1.8x da CLI de criptografia da AWS substituiu a versão 1.7.x e a versão 2.1.x da CLI de criptografia da AWS substituiu a 2.0.x. Para obter detalhes, consulte o [boletim de segurança](#) pertinente no repositório [aws-encryption-sdk-cli](#) no GitHub.

Para obter um exemplo de como usar o atributo de segurança que limita as chaves de dados criptografadas, consulte [Limitar as chaves de dados criptografadas](#).

Para obter um exemplo de como usar chaves multirregionais do AWS KMS, consulte [Usar AWS KMS keys multirregionais](#).

Tópicos

- [Como criptografar e descriptografar dados](#)
- [Como especificar chaves de encapsulamento](#)
- [Como fornecer entrada](#)
- [Como especificar o local de saída](#)

- [Como usar um contexto de criptografia](#)
- [Como especificar uma política de compromisso](#)
- [Como armazenar parâmetros em um arquivo de configuração](#)

Como criptografar e descriptografar dados

A CLI de criptografia da AWS usa os atributos do AWS Encryption SDK para facilitar a criptografia e a descriptografia de dados com segurança.

Note

O parâmetro `--master-keys` foi descontinuado na versão 1.8.x da CLI de criptografia da AWS e removido na versão 2.1.x.. Em vez disso, use o parâmetro `--wrapping-keys`. A partir da versão 2.1.x, o parâmetro `--wrapping-keys` passou a ser necessário ao criptografar e descriptografar. Para obter mais detalhes, consulte [Referência da sintaxe e de parâmetros da CLI do AWS Encryption SDK](#).

- Ao criptografar dados na CLI de criptografia da AWS, você especifica seus dados de texto simples e [uma chave de encapsulamento](#) (ou chave mestra), como uma AWS KMS key no AWS Key Management Service (AWS KMS). Se estiver usando um provedor de chaves mestras personalizado, você também precisará especificar o provedor. Você também especifica locais de saída para a [mensagem criptografada](#) e para os metadados sobre a operação de criptografia. Um [contexto de criptografia](#) é opcional, mas recomendado.

Na versão 1.8.x, o parâmetro `--commitment-policy` é obrigatório quando você usar o parâmetro `--wrapping-keys`; caso contrário ele não será válido. A partir da versão 2.1x, o parâmetro `--commitment-policy` passou a ser opcional, mas é recomendado.

```
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
```

A CLI de criptografia da AWS criptografa seus dados com uma chave de dados exclusiva. Em seguida, ele criptografa cada chave de dados sob a chave de encapsulamento especificada.

Ela retorna uma [mensagem criptografada](#) e os metadados sobre a operação. A mensagem criptografada contém os dados criptografados (texto cifrado) e uma cópia criptografada da chave de dados. Você não precisa se preocupar com o armazenamento, o gerenciamento ou a perda da chave de dados.

- Ao descriptografar os dados, você passa sua mensagem criptografada, o contexto de criptografia opcional e o local para a saída do texto não criptografado e os metadados. Você também especifica as chaves de encapsulamento que a CLI de criptografia da AWS pode usar para descriptografar a mensagem, ou informa à CLI de criptografia da AWS que ela pode usar qualquer chave de encapsulamento que criptografe a mensagem.

A partir da versão 1.8.x, o parâmetro `--wrapping-keys` passou a ser opcional, mas é recomendado. A partir da versão 2.1.x, o parâmetro `--wrapping-keys` passou a ser necessário ao criptografar e descriptografar.

Ao descriptografar, você pode usar o atributo `key` do parâmetro `--wrapping-keys` para especificar as chaves de encapsulamento que descriptografam seus dados. A especificação de uma chave de encapsulamento do AWS KMS ao descriptografar é opcional, mas é uma [prática recomendada](#) que impede que você use uma chave que não pretendia usar. Se estiver usando um provedor de chaves mestras personalizado, você deverá especificar o provedor.

Se você não usar o atributo `key`, deverá definir o atributo de [discovery](#) do parâmetro `--wrapping-keys` como `true`, o que permite que a CLI de criptografia da AWS seja descriptografada usando qualquer chave de encapsulamento que criptografou a mensagem.

É uma prática recomendada usar o parâmetro `--max-encrypted-data-keys`, para evitar a descriptografia de uma mensagem malformada com um número excessivo de chaves de dados criptografadas. Especifique o número esperado de chaves de dados criptografadas (um para cada chave de encapsulamento usada na criptografia) ou uma quantidade máxima razoável (como 5). Para obter mais detalhes, consulte [Limitar as chaves de dados criptografadas](#).

O parâmetro `--buffer` retorna texto simples somente após o processamento de todas as entradas, incluindo a verificação da assinatura digital, se houver uma.

O parâmetro `--decrypt-unsigned` descriptografa o texto cifrado e garante que as mensagens não sejam assinadas antes de serem descriptografadas. Use esse parâmetro se você usou o parâmetro `--algorithm` e selecionou um pacote de algoritmos sem assinatura digital para criptografar dados. Se o texto cifrado for assinado, a descriptografia falhará.

Você pode usar `--decrypt` ou `--decrypt-unsigned` para fazer a descriptografia, mas não ambos.

```
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
```

A CLI de criptografia da AWS usa a chave mestra para descriptografar a chave de dados na mensagem criptografada. Em seguida, ela usa a chave de dados para descriptografar os dados. Ela retorna os dados em texto não criptografado e os metadados sobre a operação.

Como especificar chaves de encapsulamento

Ao criptografar dados na CLI de criptografia da AWS, você precisa especificar pelo menos uma [chave de encapsulamento \(ou chave mestra\)](#). Você pode usar AWS KMS keys no AWS Key Management Service (AWS KMS), chaves de encapsulamento de um [provedor de chave mestra](#) personalizado ou ambos. O provedor de chaves mestras personalizado pode ser qualquer provedor de chaves mestras compatível com o Python.

Para especificar as chaves de encapsulamento nas versões 1.8.x e posteriores, use o parâmetro `--wrapping-keys` (`-w`). O valor deste parâmetro é uma coleção de [atributos](#) com o formato `attribute=value`. Os atributos que você usa dependem do provedor de chaves mestras e do comando.

- AWS KMS. Em comandos `encrypt`, você deve especificar um parâmetro `--wrapping-keys` com um atributo `key`. A partir da versão 2.1.x, o parâmetro `--wrapping-keys` também passou a ser necessário em comandos de descriptografia. Ao descriptografar, o parâmetro `--wrapping-keys` deve ter um atributo `key` ou um atributo `discovery` (mas não os dois) com o valor de `true`. Todos os outros atributos são opcionais.
- Provedor de chaves mestras personalizado. Você deve especificar um parâmetro `--wrapping-keys` em cada comando. O valor do parâmetro deve ter os atributos `key` e `provider`.

Você pode incluir [vários parâmetros --wrapping-keys](#) e vários atributos key no mesmo comando.

Encapsulando os atributos dos parâmetros de chave

O valor do parâmetro --wrapping-keys consiste nos seguintes atributos e seus valores. Um parâmetro --wrapping-keys (ou parâmetro --master-keys) é necessário em todos os comandos de criptografia. A partir da versão 2.1.x, o parâmetro --wrapping-keys também passou a ser necessário em comandos de descriptografia.

Se um nome ou valor de atributo incluir espaços ou caracteres especiais, coloque o nome e o valor entre aspas. Por exemplo, --wrapping-keys key=12345 "provider=my cool provider".

Chave: especifique uma chave de encapsulamento

Use o atributo key para identificar uma chave de encapsulamento. Ao criptografar, o valor pode ser qualquer identificador de chave que o provedor de chaves mestras reconhece.

```
--wrapping-keys key=1234abcd-12ab-34cd-56ef-1234567890ab
```

Em um comando encrypt, cada valor do parâmetro você deve incluir pelo menos um atributo key e um valor. Para criptografar sua chave de dados em várias chaves de encapsulamento, use [vários atributos key](#).

```
aws-encryption-cli --encrypt --wrapping-keys  
key=1234abcd-12ab-34cd-56ef-1234567890ab key=1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d
```

Em comandos encrypt que usam AWS KMS keys, o valor da chave pode ser o ID da chave, o ARN da chave, um nome de alias ou um ARN de alias. Por exemplo, este comando encrypt usa um ARN do alias no valor do atributo key. Para obter detalhes sobre os identificadores de chave de uma AWS KMS key, consulte [Identificadores de chave](#) no Guia do desenvolvedor do AWS Key Management Service.

```
aws-encryption-cli --encrypt --wrapping-keys key=arn:aws:kms:us-west-2:111122223333:alias/ExampleAlias
```

Em comandos decrypt que usam um provedor de chaves mestras personalizado, os atributos key e provider são necessários.

```
\\ Custom master key provider  
aws-encryption-cli --decrypt --wrapping-keys provider='myProvider' key='100101'
```

Nos comandos `decrypt` que usam o AWS KMS, você pode usar o atributo `key` para especificar a AWS KMS key a ser utilizada para descriptografar ou o [atributo `discovery`](#) com o valor `true`, que permite que a CLI de criptografia da AWS use qualquer AWS KMS key que tenha sido usada para criptografar a mensagem. Se você especificar uma AWS KMS key, ela deverá ser uma das chaves de encapsulamento usadas para criptografar a mensagem.

A especificação da chave de encapsulamento é uma [AWS Encryption SDK prática recomendada](#). Isso garante que você use a AWS KMS key que pretende usar.

Em um comando `decrypt`, o valor do atributo `key` deve ser um [ARN de chave](#).

```
\\ AWS KMS key
aws-encryption-cli --decrypt --wrapping-keys key=arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

Descoberta: use qualquer AWS KMS key ao descriptografar

Se você não precisar limitar o uso de AWS KMS keys ao descriptografar, pode usar o atributo `discovery` com um valor de `true`. Um valor de `true` permite que a CLI de criptografia da AWS descriptografe usando qualquer AWS KMS key que criptografou a mensagem. Se você não especificar um atributo `discovery`, a descoberta será `false` (padrão). O atributo `descoberta` é válido somente em comandos de descriptografia e somente quando a mensagem tiver sido criptografada com AWS KMS keys.

O atributo `discovery` com o valor definido como `true` é uma alternativa ao uso do atributo `key` para especificar uma AWS KMS key. Ao descriptografar uma mensagem criptografada com AWS KMS keys, cada parâmetro `--wrapping-keys` deve ter um atributo `key` ou um atributo `discovery` (mas não os dois) com o valor definido como `true`.

Quando `discovery` for `true`, uma prática recomendada é usar os atributos `discovery-partition` e `discovery-account` para limitar as AWS KMS keys usadas nos atributos de Contas da AWS especificados por você. No exemplo a seguir, os atributos `discovery` permitem que a CLI de criptografia da AWS use qualquer um dos atributos de AWS KMS key na Contas da AWS especificada.

```
aws-encryption-cli --decrypt --wrapping-keys \
  discovery=true \
  discovery-partition=aws \
  discovery-account=111122223333 \
  discovery-account=444455556666
```

Provider: especifique o provedor de chaves mestras

O atributo `provider` identifica o [provedor de chaves mestres](#). O valor padrão é `aws-kms` que representa o AWS KMS. Se estiver usando outro provedor de chaves mestres, o atributo `provider` será necessário.

```
--wrapping-keys key=12345 provider=my_custom_provider
```

Para obter mais informações sobre o uso de provedores de chaves mestras personalizadas (que não sejam AWS KMS), consulte o tópico Configuração avançada no arquivo [README](#) do repositório da [CLI e criptografia da AWS](#).

Região: especificar uma Região da AWS

Use o atributo `region` para especificar a Região da AWS de uma AWS KMS key. Esse atributo é válido apenas em comandos `encrypt` e somente quando o provedor de chaves mestras é o AWS KMS.

```
--encrypt --wrapping-keys key=alias/primary-key region=us-east-2
```

Os comandos da CLI de criptografia da AWS usa a Região da AWS especificada no valor do atributo `key` se ele incluir uma região, como uma ARN. Se o valor de `key` especificar uma Região da AWS, o atributo `region` será ignorado.

O atributo `region` tem precedência sobre outras especificações de região. Se você não usar um atributo `region`, os comandos da CLI de criptografia da AWS usarão a Região da AWS especificada em seu [perfil nomeado](#) da AWS CLI, caso exista, ou em seu perfil padrão.

Profile: especifique um perfil nomeado

Use o atributo `profile` para especificar um AWS CLI perfil nomeado [da](#) . Os perfis nomeados podem incluir credenciais e uma Região da AWS. Esse atributo é válido somente quando o provedor de chaves mestras é o AWS KMS.

```
--wrapping-keys key=alias/primary-key profile=admin-1
```

Você pode usar o atributo `profile` para especificar credenciais alternativas em comandos `encrypt` e `decrypt`. Em um comando `encrypt`, a CLI de criptografia da AWS usa a Região da AWS no perfil nomeado apenas quando o valor de `key` não incluir uma região e não existir nenhum atributo `region`. Em um comando `decrypt`, a Região da AWS no perfil de nome é ignorada.

Como especificar várias chaves mestras

Você pode especificar várias chaves de encapsulamento (ou chaves mestras) em cada comando.

Se você especificar mais de uma chave de encapsulamento, a primeira chave de encapsulamento gerará (e criptografará) a chave de dados usada para criptografar seus dados. As outras chaves de encapsulamento criptografam a mesma chave de dados. A [mensagem criptografada](#) resultante contém os dados criptografados ("texto cifrado") e uma coleção de chaves de dados criptografadas, criptografadas por cada chave de encapsulamento. Qualquer uma das chaves de encapsulamento podem descriptografar uma chave de dados e descriptografar os dados.

Há duas maneiras de especificar várias chaves de encapsulamento:

- Incluir vários atributos `key` no valor do parâmetro `--wrapping-keys`.

```
$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
```

- Incluir vários parâmetros `--wrapping-keys` no mesmo comando. Use essa sintaxe quando os valores dos atributos que você especificar não se aplicarem a todas as chaves de encapsulamento no comando.

```
--wrapping-keys region=us-east-2 key=alias/test_key \
--wrapping-keys region=us-west-1 key=alias/test_key
```

O atributo `discovery` com um valor definido como `true` permite que a CLI de criptografia da AWS use qualquer AWS KMS key que criptografe a mensagem. Se você usar vários parâmetros `--wrapping-keys` no mesmo comando, o uso de `discovery=true` em qualquer parâmetro `--wrapping-keys` substituirá efetivamente os limites do atributo `key` em outros parâmetros `--wrapping-keys`.

Por exemplo, no comando a seguir, o atributo `key` no primeiro parâmetro de `--wrapping-keys` limita a CLI de criptografia da AWS à AWS KMS key especificada. No entanto, o atributo `discovery` no segundo parâmetro de `--wrapping-keys` permite que a CLI de criptografia da AWS use qualquer AWS KMS key especificada para descriptografar a mensagem.

```
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
```

Como fornecer entrada

A operação de criptografia na CLI de criptografia da AWS usa os dados de texto simples como entrada e retorna uma [mensagem criptografada](#). A operação de descriptografia usa uma mensagem criptografada como entrada e retorna dados de texto não criptografado.

O parâmetro `--input (-i)`, que informa à CLI de criptografia da AWS onde localizar a entrada, é necessário em todos os comandos da CLI de criptografia da AWS.

Você pode fornecer entrada de qualquer uma das seguintes formas:

- Usar um arquivo.

```
--input myData.txt
```

- Usar um padrão de nome de arquivo.

```
--input testdir/*.xml
```

- Use um diretório ou um padrão de nome de diretório. Quando a entrada é um diretório, o parâmetro `--recursive (-r, -R)` é necessário.

```
--input testdir --recursive
```

- Redirecionar a entrada para o comando (stdin). Use um valor de `-` para o parâmetro `--input`. (O parâmetro `--input` sempre é necessário.)

```
echo 'Hello World' | aws-encryption-cli --encrypt --input -
```

Como especificar o local de saída

O parâmetro `--output` informa à CLI de criptografia da AWS onde gravar os resultados da operação de criptografia ou descriptografia. Ele é necessário em todos os comandos da CLI de

criptografia da AWS. A CLI de criptografia da AWS cria um novo arquivo de saída para cada arquivo de entrada na operação.

Se um arquivo de saída já existir, por padrão, a CLI de criptografia da AWS imprime um aviso e, em seguida, substitui o arquivo. Para evitar a substituição, use o parâmetro `--interactive`, que solicita sua confirmação antes de substituir, ou `--no-overwrite`, que ignora a entrada se a saída puder provocar uma substituição. Para suprimir o aviso de substituição, use `--quiet`. Para capturar erros e avisos da CLI de criptografia da AWS, use o operador de redirecionamento `2>&1` para gravá-los no fluxo de saída.

Note

Os comandos que substituem arquivos de saída começam excluindo o arquivo de saída. Se o comando falhar, o arquivo de saída talvez já tenha sido excluído.

Você pode definir o local da saída de várias maneiras.

- Especificar um nome de arquivo. Se você especificar um caminho para o arquivo, todos os diretórios no caminho devem existir antes do comando ser executado.

```
--output myEncryptedData.txt
```

- Especificar um diretório. O diretório de saída deve existir antes do comando ser executado.

Se a entrada contiver subdiretórios, o comando reproduzirá os subdiretórios no diretório especificado.

```
--output Test
```

Quando o local de saída for um diretório (sem nomes de arquivos), a CLI de criptografia da AWS cria nomes de arquivos de saída com base nos nomes de arquivo de entrada mais um sufixo. As operações de criptografia acrescentam `.encrypted` ao nome do arquivo de entrada e as operações de descryptografia acrescentam `.decrypted`. Para alterar o sufixo, use o parâmetro `--suffix`.

Por exemplo, se você criptografar `file.txt`, o comando `encrypt` criará `file.txt.encrypted`. Se você descryptografar `file.txt.encrypted`, o comando `decrypt` criará `file.txt.encrypted.decrypted`.

- Gravar na linha de comando (stdout). Insira um valor de - para o parâmetro --output. Você pode usar --output - para redirecionar a saída em outro comando ou programa.

```
--output -
```

Como usar um contexto de criptografia

A CLI de criptografia da AWS permite que você forneça um contexto de criptografia em comandos encrypt e decrypt. Ele não é necessário, mas é uma melhor prática criptográfica que recomendamos.

Um contexto de criptografia é um tipo de dados autenticados adicionais arbitrários e que não são segredos. Na CLI de criptografia da AWS, o contexto de criptografia consiste em uma coleção de pares name=value. Você pode usar qualquer conteúdo nos pares, incluindo informações sobre os arquivos; dados que o ajudam a encontrar a operação de criptografia em logs; ou dados que suas concessões ou políticas exigem.

Em um comando encrypt

O contexto de criptografia que você especifica em um comando encrypt, junto com qualquer par que o [CMM](#) adicionar, é associado de maneira criptográfica aos dados criptografados. Ele também é incluído (em não criptografado) na [mensagem criptografada](#) que o comando retorna. Se você estiver usando uma AWS KMS key, o contexto de criptografia também poderá ser exibido em texto não criptografado em registros e logs de auditoria, como o AWS CloudTrail.

O exemplo a seguir mostra um contexto de criptografia com três pares name=value.

```
--encryption-context purpose=test dept=IT class=confidential
```

Em um comando decrypt

Em um comando decrypt, o contexto de criptografia ajuda a confirmar se você está descriptografando a mensagem criptografada correta.

Não é necessário fornecer um contexto de criptografia em um comando decrypt, mesmo que um contexto de criptografia tenha sido usado na criptografia. No entanto, se você fizer isso, a CLI de criptografia da AWS verificará se cada elemento no contexto de criptografia do comando

`decrypt` corresponde a um elemento no contexto de criptografia da mensagem criptografada. Se um elemento não corresponder, o comando `decrypt` falhará.

Por exemplo, o comando a seguir descriptografa a mensagem criptografada somente se o contexto de criptografia incluir `dept=IT`.

```
aws-encryption-cli --decrypt --encryption-context dept=IT ...
```

Um contexto de criptografia é uma parte importante de sua estratégia de segurança. No entanto, ao escolher um contexto de criptografia, lembre-se de que seus valores não são secretos. Não inclua dados confidenciais no contexto de criptografia.

Como especificar um contexto de criptografia

- Em um comando `encrypt`, use o parâmetro `--encryption-context` com um ou mais pares `name=value`. Use um espaço para separar cada par.

```
--encryption-context name=value [name=value] ...
```

- Em um comando `decrypt`, o valor do parâmetro `--encryption-context` pode incluir pares `name=value`, elementos `name` (sem valores) ou uma combinação de ambos.

```
--encryption-context name[=value] [name] [name=value] ...
```

Se o `name` ou o `value` em um par de `name=value` incluir espaços ou caracteres especiais, coloque o par inteiro entre aspas.

```
--encryption-context "department=software engineering" "Região da AWS=us-west-2"
```

Por exemplo, este comando `encrypt` inclui um contexto de criptografia com dois pares, `purpose=test` e `dept=23`.

```
aws-encryption-cli --encrypt --encryption-context purpose=test dept=23 ...
```

Esse comando `decrypt` tem êxito. O contexto de criptografia em cada comando é um subconjunto do contexto de criptografia original.

```
\\ 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 ...
```

No entanto, esses comandos `decrypt` falharão. O contexto de criptografia na mensagem criptografada não contém os elementos específicos.

```
aws-encryption-cli --decrypt --encryption-context dept=Finance ...
```

```
aws-encryption-cli --decrypt --encryption-context scope ...
```

Como especificar uma política de compromisso

Para definir a [política de compromisso](#) para o comando, use o [parâmetro `--commitment-policy`](#). Esse parâmetro foi apresentado na versão 1.8.x.. Ele é válido em comandos de criptografia e descryptografia. A política de compromisso que você definir será válida somente para o comando no qual ela aparece. Se você não definir uma política de compromisso para um comando, a CLI de criptografia da AWS usará o valor padrão.

Por exemplo, o valor do parâmetro a seguir define a política de compromisso como `require-encrypt-allow-decrypt`, que sempre criptografa com o confirmação de chave, mas descryptografa um texto cifrado criptografado com ou sem confirmação de chave.

```
--commitment-policy require-encrypt-allow-decrypt
```

Como armazenar parâmetros em um arquivo de configuração

Você pode economizar tempo e evitar erros de digitação salvando os parâmetros e valores usados com frequência na CLI de criptografia da AWS em arquivos de configuração.

Um arquivo de configuração é um arquivo de texto que contém parâmetros e valores para um comando da CLI de criptografia da AWS. Ao fazer referência a um arquivo de configuração em um comando da CLI de criptografia da AWS, a referência é substituída pelos parâmetros e valores no arquivo de configuração. O efeito será o mesmo como se você tivesse digitado o conteúdo do arquivo na linha de comando. Um arquivo de configuração pode ter qualquer nome e pode ser localizado em qualquer diretório que o usuário atual pode acessar.

O arquivo de configuração de exemplo a seguir, `key.conf`, especifica duas AWS KMS keys em diferentes regiões.

```
--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
```

Para usar o arquivo de configuração em um comando, prefixe o nome do arquivo com uma arroba (@). Em um console do PowerShell, use um caractere backtick para escape da arroba (`@).

Este comando de exemplo usa o arquivo `key.conf` em um comando `encrypt`.

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
```

Regras do arquivo de configuração

As regras para uso de arquivos de configuração são:

- Você pode incluir vários parâmetros em cada arquivo de configuração e listá-los em qualquer ordem. Liste cada parâmetro com seus valores (se houver) em uma linha separada.
- Use `#` para adicionar um comentário a toda ou a parte de uma linha.
- Você pode incluir referências a outros arquivos de configuração. Não use um backtick para escape do sinal `@`, mesmo no PowerShell.
- Se você usar aspas em um arquivo de configuração, o texto entre aspas não pode abranger várias linhas.

Por exemplo, este é o conteúdo de um arquivo `encrypt.conf` de exemplo.

```
# 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
```

Você também pode incluir vários arquivos de configuração em um comando. Este comando de exemplo usa os arquivos de configuração `encrypt.conf` e `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
```

Próximo: [Experimente os exemplos da CLI de criptografia da AWS](#)

Exemplos da CLI de criptografia da AWS

Use os seguintes exemplos para testar a CLI de Criptografia da AWS na plataforma de sua preferência. Para obter ajuda com chaves mestras e outros parâmetros, consulte [Como usar a CLI de criptografia da AWS](#). Para obter uma referência rápida, consulte [Referência da sintaxe e de parâmetros da CLI do AWS Encryption SDK](#).

Note

Os exemplos a seguir usam a sintaxe da CLI de Criptografia da AWS versão 2.1x.. Os novos atributos de segurança foram lançados originalmente nas versões 1.7.x e 2.0.x da CLI de criptografia da AWS. No entanto, a versão 1.8x da CLI de criptografia da AWS substituiu a versão 1.7.x e a versão 2.1.x da CLI de criptografia da AWS substituiu a 2.0.x. Para obter detalhes, consulte o [boletim de segurança](#) pertinente no repositório [aws-encryption-sdk-cli](#) no GitHub.

Para obter um exemplo de como usar o atributo de segurança que limita as chaves de dados criptografadas, consulte [Limitar as chaves de dados criptografadas](#).

Para obter um exemplo de como usar chaves multirregionais do AWS KMS, consulte [Usar AWS KMS keys multirregionais](#).

Tópicos

- [Criptografar um arquivo](#)
- [Descriptografar um arquivo](#)
- [Criptografar todos os arquivos em um diretório](#)
- [Descriptografar todos os arquivos em um diretório](#)
- [Criptografar e descriptografar na linha de comando](#)
- [Uso de várias chaves mestras](#)
- [Criptografar e descriptografar em scripts](#)
- [Usar o armazenamento em cache de chaves de dados](#)

Criptografar um arquivo

Este exemplo usa a CLI de Criptografia da AWS para criptografar o conteúdo do arquivo `hello.txt`, que contém uma string "Hello World".

Quando você executa um comando `encrypt` em um arquivo, a CLI de Criptografia da AWS obtém o conteúdo do arquivo, gera uma [chave de dados](#) exclusiva, criptografa o conteúdo do arquivo sob a chave de dados e, em seguida, grava a [mensagem criptografada](#) em um novo arquivo.

O primeiro comando salva o ARN da AWS KMS key em uma variável `$keyArn`. Ao criptografar com uma AWS KMS key, você pode identificá-la usando um ID de chave, ARN de chave, nome de alias ou ARN de alias. Para obter detalhes sobre os identificadores de chave de uma AWS KMS key, consulte [Identificadores de chave](#) no Guia do desenvolvedor do AWS Key Management Service.

O segundo comando criptografa o conteúdo do arquivo. O comando usa o parâmetro `--encrypt` para especificar a operação, e o parâmetro `--input` para indicar o arquivo a ser criptografado. O [--wrapping-keys parâmetro](#) e seu atributo `key` necessário informam o comando para usar a AWS KMS key representada pelo ARN da chave.

O comando usa o parâmetro `--metadata-output` para especificar um arquivo de texto para os metadados sobre a operação de criptografia. Como prática recomendada, o comando usa o parâmetro `--encryption-context` para especificar um [contexto de criptografia](#).

Esse comando também usa o [parâmetro `--commitment-policy`](#) para definir explicitamente a política de compromisso. Na versão 1.8. x, ele é necessário quando você usa o parâmetro `--wrapping-keys`. A partir da versão 2.1x, o parâmetro `--commitment-policy` passou a ser opcional, mas é recomendado.

O valor do parâmetro `--output`, um ponto (`.`), informa o comando para gravar o arquivo de saída no diretório atual.

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 .

```

Quando o comando `encrypt` é bem-sucedido, ele não retorna nenhuma saída. Para determinar se o comando foi bem-sucedido, verifique o valor booleano na variável `$?`. Quando o comando é bem-sucedido, o valor de `$?` é `0` (Bash) ou `True` (PowerShell). Quando o comando falha, o valor de `$?` é não zero (Bash) ou `False` (PowerShell).

Bash

```
$ echo $?  
0
```

PowerShell

```
PS C:\> $?  
True
```

Você também pode usar um comando de listagem de diretório para ver se o comando `encrypt` criou um novo arquivo, `hello.txt.encrypted`. Como o comando `encrypt` não especificou um nome de arquivo para a saída, a CLI de Criptografia da AWS gravou a saída em um arquivo com o mesmo nome que o arquivo de entrada mais um sufixo `.encrypted`. Para usar outro sufixo ou suprimir o sufixo, use o parâmetro `--suffix`.

O arquivo `hello.txt.encrypted` contém uma [mensagem criptografada](#) que inclui o texto cifrado do arquivo `hello.txt`, uma cópia criptografada da chave de dados e metadados adicionais incluindo o contexto de criptografia.

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
```

Descriptografar um arquivo

Este exemplo usa a CLI de Criptografia da AWS para descriptografar o conteúdo do arquivo `Hello.txt.encrypted` que foi criptografado no exemplo anterior.

O comando `decrypt` usa o parâmetro `--decrypt` para indicar a operação, e o parâmetro `--input` para identificar o arquivo a ser descriptografado. O valor do parâmetro `--output` é um ponto que representa o diretório atual.

O parâmetro `--wrapping-keys` com um atributo `key` especifica a chave de encapsulamento usada para descriptografar a mensagem criptografada. Em comandos `decrypt` com AWS KMS keys, o valor do atributo `key` deve ser um [ARN de chave](#). O parâmetro `--wrapping-keys` é obrigatório em comandos `encrypt`. Se você usar AWS KMS keys, poderá usar o atributo `key` para especificar AWS KMS keys para descriptografar ou o atributo `discovery` com um valor definido como `true` (mas não ambos). Se estiver usando outro provedor de chaves mestras, os atributos `key` e `provider` serão necessários.

A partir da versão 2.1x, o parâmetro `--commitment-policy` passou a ser opcional, mas é recomendado. Usá-lo explicitamente deixa clara sua intenção, mesmo se você especificar o valor padrão, `requires-encrypt-require-decrypt`.

O parâmetro `--encryption-context` é opcional no comando `decrypt`, mesmo quando um [contexto de criptografia](#) é fornecido no comando `encrypt`. Nesse caso, o comando `decrypt` usa o mesmo contexto de criptografia que foi fornecido no comando `encrypt`. Antes de descriptografar, a CLI de criptografia da AWS verifica se o contexto de criptografia na mensagem criptografada inclui um par de `purpose=test`. Caso contrário, o comando `decrypt` falhará.

O parâmetro `--metadata-output` especifica um arquivo de metadados sobre a operação de descriptografia. O valor do parâmetro `--output`, um ponto (`.`), grava o arquivo de saída no diretório atual.

É uma prática recomendada usar o parâmetro `--max-encrypted-data-keys`, para evitar a descriptografia de uma mensagem malformada com um número excessivo de chaves de dados criptografadas. Especifique o número esperado de chaves de dados criptografadas (um para cada chave de encapsulamento usada na criptografia) ou uma quantidade máxima razoável (como 5). Para obter mais detalhes, consulte [Limitar as chaves de dados criptografadas](#).

Ele `--buffer` retorna texto sem formatação somente após o processamento de todas as entradas, incluindo a verificação da assinatura digital, se houver uma.

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 .

```

Quando um comando decrypt é bem-sucedido, ele não retorna nenhuma saída. Para determinar se o comando foi bem-sucedido, obtenha o valor da variável \$? . Você também pode usar um comando de listagem de diretório para ver se o comando criou um novo arquivo com um sufixo .decrypted. Para ver o conteúdo de texto não criptografado, use um comando para obter o conteúdo do arquivo, como 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
```

Criptografar todos os arquivos em um diretório

Este exemplo usa a criptografia da CLI de criptografia da AWS para criptografar o conteúdo de todos os arquivos em um diretório.

Quando um comando afeta vários arquivos, a CLI de criptografia da AWS processa cada arquivo individualmente. Ela obtém o conteúdo do arquivo, obtém uma [chave de dados](#) exclusiva para o arquivo da chave mestra, criptografa o conteúdo do arquivo sob a chave de dados e grava os resultados em um novo arquivo no diretório de saída. Como resultado, você pode descriptografar os arquivos de saída de maneira independente.

Essa listagem do diretório `TestDir` mostra os arquivos de texto não criptografado que desejamos criptografar.

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
```

O primeiro comando salva o [nome do atributo da Amazon \(ARN\)](#) de uma AWS KMS key na variável `$keyArn`.

O segundo comando criptografa o conteúdo dos arquivos no diretório `TestDir` e grava os arquivos de conteúdo criptografado no `TestEnc`. Se o diretório `TestEnc` não existir, o comando falhará. Como o local de entrada é um diretório, o parâmetro `--recursive` é obrigatório.

O [parâmetro `--wrapping-keys`](#) e seu atributo-chave obrigatório especificam a chave de encapsulamento a ser usada. O comando `encrypt` inclui um [contexto de criptografia](#), `dept=IT`. Quando você especifica um contexto de criptografia em um comando que criptografa vários arquivos, o mesmo contexto de criptografia é usado para todos os arquivos.

O comando também tem um parâmetro `--metadata-output` para informar à CLI de criptografia da AWS onde gravar os metadados sobre as operações de criptografia. A CLI de criptografia da AWS grava um registro de metadados para cada arquivo que foi criptografado.

A partir da versão 2.1.x, o [parâmetro `--commitment-policy-parameter`](#) passou a ser opcional, mas é recomendado. Se o comando ou script falhar porque não consegue decifrar um texto cifrado, a configuração explícita da política de compromisso pode ajudar a detectar o problema rapidamente.

Quando o comando é concluído, a CLI de criptografia da AWS grava os arquivos criptografados no diretório `TestEnc`, mas não retorna nenhuma saída.

O último comando lista os arquivos no diretório `TestEnc`. Há um arquivo de saída de conteúdo criptografado para cada arquivo de entrada de conteúdo de texto não criptografado. Como o comando não especificou um sufixo alternativo, o comando `encrypt` acrescentou `.encrypted` a cada um dos nomes de arquivos de entrada.

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
```

Descriptografar todos os arquivos em um diretório

Este exemplo descriptografa todos os arquivos em um diretório. Ele começa com os arquivos no diretório TestEnc que foram criptografados no exemplo anterior.

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
```

Este comando `decrypt` descriptografa todos os arquivos no diretório TestEnc e grava os arquivos de texto não criptografado no diretório TestDec. O parâmetro `--wrapping-keys` com um atributo de chave e um valor de [ARN de chave](#) informa à CLI de criptografia da AWS qual AWS KMS keys usar para descriptografar os arquivos. O comando usa o parâmetro `--interactive` para informar à CLI de criptografia da AWS para avisar você antes de substituir um arquivo com o mesmo nome.

Esse comando também usa o contexto de criptografia que foi fornecido quando os arquivos foram criptografados. Ao descriptografar vários arquivos, a CLI de criptografia da AWS verifica o contexto de criptografia de cada arquivo. Se a verificação do contexto de criptografia em qualquer arquivo falhar, a CLI de criptografia da AWS rejeita o arquivo, grava um aviso, registra a falha nos metadados e continua verificando os arquivos restantes. Se a CLI de criptografia da AWS não conseguir descriptografar um arquivo por qualquer outro motivo, todo o comando `decrypt` falhará imediatamente.

Neste exemplo, as mensagens criptografadas em todos os arquivos de entrada contêm o elemento do contexto de criptografia de `dept=IT`. No entanto, se você estiver descriptografando mensagens com diferentes contextos de criptografia, você ainda poderá verificar parte do contexto

de criptografia. Por exemplo, se algumas mensagens tiverem um contexto de criptografia de dept=finance e outras tiverem dept=IT, você poderá verificar se o contexto de criptografia sempre contém um nome dept sem especificar o valor. Se desejar ser mais específico, você poderá descriptografar os arquivos em comandos separados.

O comando decrypt não retorna nenhuma saída, mas você pode usar um comando de listagem de diretórios para ver se ele criou novos arquivos com o sufixo .decrypted. Para ver o conteúdo de texto não criptografado, use um comando para obter o conteúdo do arquivo.

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

```

Criptografar e descriptografar na linha de comando

Estes exemplos mostram como redirecionar a entrada para comandos (stdin) e gravar a saída na linha de comando (stdout). Eles explicam como representar stdin e stdout em um comando e como usar ferramentas de codificação Base64 internas para impedir que o shell interprete caracteres não ASCII incorretamente.

Este exemplo redireciona uma string de texto não criptografado para um comando encrypt e salva a mensagem criptografada em uma variável. Em seguida, ele redireciona a mensagem criptografada na variável para um comando decrypt, que grava sua saída no pipeline (stdout).

O exemplo consiste em três comandos:

- O primeiro comando salva o [ARN da chave](#) de um AWS KMS key em uma variável \$keyArn.

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'

```

- O segundo comando redireciona a string `Hello World` para o comando `encrypt` e salva o resultado na variável `$encrypted`.

Os parâmetros `--input` e `--output` são obrigatórios em todos os comandos da CLI de criptografia da AWS. Para indicar que a entrada está sendo redirecionada para o comando (stdin), use um hífen (-) para o valor do parâmetro `--input`. Para enviar a saída para a linha de comando (stdout), use um hífen para o valor do parâmetro `--output`.

O parâmetro `--encode` codifica a saída em Base64 antes de retorná-la. Isso evita que o shell interprete incorretamente os caracteres não ASCII na mensagem criptografada.

Como esse comando é apenas uma prova de conceito, omitimos o contexto de criptografia e suprimimos os metadados (-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
```

- O terceiro comando redireciona a mensagem criptografada na variável `$encrypted` para descriptografar o comando.

Esse comando `decrypt` usa `--input -` para indicar que a entrada é proveniente do pipeline (stdin) e do `--output -` para enviar a saída para o pipeline (stdout). (O parâmetro de entrada usa o local da entrada, não os bytes reais da entrada. Portanto, você não pode usar a variável `$encrypted` como o valor do parâmetro `--input`.)

Este exemplo usa o atributo de descoberta do parâmetro `--wrapping-keys` para permitir que a CLI de criptografia da AWS use qualquer AWS KMS key para descriptografar os dados. Ele não especifica uma [política de compromisso](#), portanto, usa o valor padrão para a versão 2.1.x e posteriores, `require-encrypt-require-decrypt`.

Como a saída foi criptografada e, em seguida, codificada, o comando `decrypt` usa o parâmetro `--decode` para decodificar a entrada codificada em Base64 antes de descriptografá-la. Você também pode usar o parâmetro `--decode` para decodificar a entrada codificada em Base64 antes de criptografá-la.

Novamente, o comando omite o contexto de criptografia e suprime os metadados (`-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
```

Você também pode executar operações de criptografia e descriptografia em um único comando sem a variável de intervenção.

Como no exemplo anterior, os parâmetros `--input` e `--output` têm um valor `-` e o comando usa o parâmetro `--encode` para codificar a saída, e o parâmetro `--decode` para decodificar a entrada.

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
```

Uso de várias chaves mestras

Este exemplo mostra como usar várias chaves mestras ao criptografar e descriptografar dados na CLI de criptografia da AWS.

Quando você usa várias chaves mestras para criptografar dados, qualquer uma das chaves mestras pode ser usada para descriptografar os dados. Essa estratégia garante que você possa descriptografar os dados mesmo que uma das chaves mestras esteja indisponível. Se você estiver armazenando os dados criptografados em várias Regiões da AWS, essa estratégia permite que você use uma chave mestra na mesma região para descriptografar os dados.

Quando você criptografa com várias chaves mestras, a primeira chave mestra desempenha uma função especial. Ela gera a chave de dados que é usada para criptografar os dados. As demais chaves mestras criptografam a chave de dados de texto não criptografado. A [mensagem criptografada](#) resultante inclui os dados criptografados e uma coleção de chaves de dados criptografadas, uma para cada chave mestra. Embora a primeira chave mestra tenha gerado a chave de dados, qualquer uma das chaves mestras poderá descriptografar uma das chaves de dados, que pode ser usada para descriptografar os dados.

Criptografia com três chaves mestres

Este comando de exemplo usa três chaves de encapsulamento para criptografar o arquivo `Finance.log`, uma em cada uma das três Regiões da AWS.

Ele grava a mensagem criptografada no diretório `Archive`. O comando usa o parâmetro `--suffix` sem nenhum valor para suprimir o sufixo. Portanto, os nomes dos arquivos de entrada e saída serão o mesmos.

O comando usa o parâmetro `--wrapping-keys` com três atributos `key`. Você também pode usar vários parâmetros `--wrapping-keys` no mesmo comando.

Para criptografar o arquivo de log, a CLI de criptografia da AWS solicita a primeira chave de encapsulamento na lista, `$key1`, para gerar a chave de dados que ela usa para criptografar os dados. Em seguida, ela usa cada uma das outras chaves de encapsulamento para criptografar uma cópia de texto não criptografado da mesma chave de dados. A mensagem criptografada no arquivo de saída inclui todas as três chaves de dados criptografadas.

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
```

Este comando descriptografa a cópia criptografada do arquivo `Finance.log` e grava-o em um arquivo `Finance.log.clear` no diretório `Finance`. Para descriptografar dados criptografados abaixo de três AWS KMS keys, especifique os mesmos três AWS KMS keys ou qualquer subconjunto deles. Este exemplo especifica somente um dos AWS KMS keys.

Para informar à CLI de criptografia da AWS qual AWS KMS keys usar para descriptografar seus dados, use o atributo da chave do parâmetro. `--wrapping-keys` Em comandos `decrypt` com AWS KMS keys, o valor do atributo da chave deve ser um [ARN da chave](#).

É preciso ter permissão para chamar a [API Decrypt](#) no AWS KMS keys que você especificar. Para obter mais informações, consulte [Autenticação e controle de acesso do AWS KMS](#).

Como prática recomendada, estes exemplos usam o parâmetro `--max-encrypted-data-keys` para evitar a descriptografia de uma mensagem malformada com um número excessivo de chaves de dados criptografadas. Embora o exemplo use somente uma chave de encapsulamento para decodificação, a mensagem criptografada tem três (3) chaves de dados criptografadas; uma para cada uma das três chaves de encapsulamento usadas na criptografia. Especifique o número esperado de chaves de dados criptografadas ou um valor máximo razoável, como 5. Se especificar um valor máximo menor que 3, o comando falhará. Para obter mais detalhes, consulte [Limitar as chaves de dados criptografadas](#).

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
```

Criptografar e descriptografar em scripts

Este exemplo mostra como usar a CLI de criptografia da AWS em scripts. Você pode escrever scripts que apenas criptografam e descriptografam dados ou scripts que criptografam ou descriptografam como parte de um processo de gerenciamento de dados.

Neste exemplo, o script obtém uma coleção de arquivos de log, compacta-os, criptografa-os e, em seguida, copia os arquivos criptografados em um bucket do Amazon S3. Esse script processa cada arquivo separadamente, para que você possa descriptografá-los e expandi-los de maneira independente.

Ao compactar e criptografar arquivos, certifique-se de compactar antes de criptografar. Dados criptografados corretamente não podem ser compactados.

Warning

Tenha cuidado ao compactar dados que incluam segredos e dados que possam ser controlados por um ator mal-intencionado. O tamanho final dos dados compactados pode revelar inadvertidamente informações confidenciais sobre seu conteúdo.

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"
    }
}
}
}
}
```

Usar o armazenamento em cache de chaves de dados

Este exemplo usa o [armazenamento em cache de chaves de dados](#) em um comando que criptografa um grande número de arquivos.

Por padrão, a CLI de criptografia da AWS (e outras versões do AWS Encryption SDK) gera uma chave de dados exclusiva para cada arquivo que criptografa. Embora o uso de uma chave de dados exclusiva para cada operação seja uma prática recomendada de criptografia, a reutilização limitada de chaves de dados é aceitável em algumas situações. Se você estiver considerando o armazenamento em cache de chaves de dados, consulte um engenheiro de segurança para compreender os requisitos de segurança do seu aplicativo e determinar os limites de segurança apropriados para você.

Neste exemplo, o armazenamento em cache de chaves de dados acelera a operação de criptografia reduzindo a frequência de solicitações ao provedor de chaves mestras.

O comando neste exemplo criptografa um diretório grande com vários subdiretórios que contêm um total de aproximadamente 800 pequenos arquivos de log. O primeiro comando salva o ARN da AWS KMS key em uma variável `keyARN`. O segundo comando criptografa todos os arquivos no diretório de entrada (recursivamente) e os grava em um diretório de arquivo morto. O comando usa o parâmetro `--suffix` para especificar o sufixo `.archive`.

O parâmetro `--caching` permite o armazenamento em cache da chave de dados. O atributo `capacity`, que limita o número de chaves de dados no cache, está definido como 1, porque o processamento de arquivos seriais nunca usa mais de uma chave de dados de cada vez. O atributo `max_age`, que determina por quanto tempo a chave de dados armazenada em cache pode ser usada, está definido como 10 segundos.

O atributo opcional `max_messages_encrypted` está definido como 10 mensagens, portanto, uma única chave de dados nunca é usada para criptografar mais de 10 arquivos. A limitação do número de arquivos criptografados por cada chave de dados reduz o número de arquivos que devem ser afetados no caso improvável de uma chave de dados estar comprometida.

Para executar esse comando em arquivos de log gerados pelo sistema operacional, você pode precisar de permissões de administrador (`sudo` no Linux; `Run as Administrator` (Executar como administrador) no 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
```

Para testar o efeito do armazenamento em cache da chave de dados, este exemplo usa o cmdlet [Measure-Command](#) no PowerShell. Ao executar esse exemplo sem o armazenamento em cache da chave de dados, ele demora cerca de 25 segundos para ser concluído. Esse processo gera uma nova chave de dados para cada arquivo no diretório.

```
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     : 0.007070311166666667
TotalMinutes   : 0.42421867
TotalSeconds   : 25.4531202
TotalMilliseconds : 25453.1202
```

O armazenamento em cache da chave de dados acelera o processo, mesmo quando você limita cada chave de dados para um máximo de 10 arquivos. O comando agora demora menos de 12 segundos para ser concluído e reduz o número de chamadas ao provedor de chaves mestras para 1/10 do valor original.

```

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    : 0.003281462222222222
TotalMinutes  : 0.19688773333333333
TotalSeconds  : 11.813264
TotalMilliseconds : 11813.264

```

Se você eliminar a restrição `max_messages_encrypted`, todos os arquivos serão criptografados com a mesma chave de dados. Essa alteração aumenta o risco de reutilização de chaves de dados sem tornar o processo muito mais rápido. No entanto, ela reduz o número de chamadas ao provedor de chaves mestras para 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      : 0.1708722783333333
TotalSeconds      : 10.2523367
TotalMilliseconds : 10252.3367
```

Referência da sintaxe e de parâmetros da CLI do AWS Encryption SDK

Este tópico fornece diagramas da sintaxe e breves descrições dos parâmetros para ajudá-lo a usar a interface da linha de comando (CLI) do AWS Encryption SDK. Para obter ajuda com chaves mestras e outros parâmetros, consulte [Como usar a CLI de criptografia da AWS](#). Para ver exemplos, consulte [Exemplos da CLI de criptografia da AWS](#). Para obter a documentação completa, consulte [Leia os documentos](#).

Tópicos

- [Sintaxe da CLI de criptografia da AWS](#)
- [Parâmetros de linha de comando da CLI de criptografia da AWS](#)
- [Parâmetros avançados](#)

Sintaxe da CLI de criptografia da AWS

Esses diagramas de sintaxe da CLI de criptografia da AWS mostram a sintaxe de cada tarefa executada com a CLI de criptografia da AWS. Eles representam a sintaxe recomendada nas versões 2.1.x e posteriores da CLI de criptografia da AWS.

Os novos atributos de segurança foram lançados originalmente nas versões 1.7.x e 2.0.x da CLI de criptografia da AWS. No entanto, a versão 1.8x da CLI de criptografia da AWS substituiu a versão 1.7.x e a versão 2.1.x da CLI de criptografia da AWS substituiu a 2.0.x. Para obter detalhes, consulte o [boletim de segurança](#) pertinente no repositório [aws-encryption-sdk-cli](#) no GitHub.

Note

A menos que indicado na descrição do parâmetro, cada parâmetro ou atributo pode ser usado apenas uma vez em cada comando.

Se você usar um atributo que não seja compatível com um parâmetro, a CLI de criptografia da AWS ignorará esse atributo incompatível sem exibir um aviso ou erro.

Obter ajuda

Para obter a sintaxe completa da CLI de criptografia da AWS, com descrições dos parâmetros, use `--help` ou `-h`.

```
aws-encryption-cli (--help | -h)
```

Obter a versão

Para obter o número da versão da instalação da sua CLI de criptografia da AWS, use `--version`. Não deixe de incluir a versão ao fazer perguntas, relatar problemas ou compartilhar dicas sobre como usar a CLI de criptografia da AWS.

```
aws-encryption-cli --version
```

Criptografar dados

O diagrama da sintaxe a seguir mostra os parâmetros usados por um comando `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]
```

Descriptografar dados

O diagrama da sintaxe a seguir mostra os parâmetros usados por um comando `decrypt`.

Na versão 1.8.x, o parâmetro `--wrapping-keys` é opcional ao descriptografar, mas é recomendado. A partir da versão 2.1.x, o parâmetro `--wrapping-keys` passou a ser necessário ao criptografar e descriptografar. Para AWS KMS keys, você pode usar o atributo `key` para especificar chaves de encapsulamento (prática recomendada) ou definir o atributo `discovery` como `true`, o que não limita as chaves de encapsulamento que podem ser usadas pela CLI de criptografia da AWS.

```
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]
```

Usar arquivos de configuração

Você pode fazer referência a arquivos de configuração que contêm parâmetros e seus valores. Isso é equivalente a digitar os parâmetros e os valores no comando. Para ver um exemplo, consulte [Como armazenar parâmetros em um arquivo de configuração](#).

```
aws-encryption-cli @<configuration_file>

# In a PowerShell console, use a backtick to escape the @.
aws-encryption-cli `@<configuration_file>
```

Parâmetros de linha de comando da CLI de criptografia da AWS

Esta lista fornece uma descrição básica dos parâmetros de comandos da CLI de criptografia da AWS. Para obter uma descrição completa, consulte a [Documentação da aws-encryption-sdk-cli](#).

`--encrypt (-e)`

Criptografa os dados de entrada. Cada comando deve ter um parâmetro `--encrypt`, `--decrypt` ou `--decrypt-unsigned`.

`--decrypt (-d)`

Descriptografa os dados de entrada. Cada comando deve ter um parâmetro `--encrypt`, `--decrypt` ou `--decrypt-unsigned`.

`--decrypt-unsigned` [Introduzido nas versões 1.9.x e 2.2.x]

O parâmetro `--decrypt-unsigned` descriptografa o texto cifrado e garante que as mensagens não sejam assinadas antes de serem descriptografadas. Use esse parâmetro se você usou o parâmetro `--algorithm` e selecionou um pacote de algoritmos sem assinatura digital para criptografar dados. Se o texto cifrado for assinado, a descriptografia falhará.

Você pode usar `--decrypt` ou `--decrypt-unsigned` para fazer a descriptografia, mas não ambos.

`--wrapping-keys (-w)` [Introduzido na versão 1.8.x]

Especifica as [chaves de encapsulamento](#) (ou chaves mestras) usadas em operações de criptografia e descriptografia. Você pode usar [vários parâmetros de --wrapping-keys](#) em cada comando.

A partir da versão 2.1.x, o parâmetro `--wrapping-keys` passou a ser necessário ao criptografar e descriptografar comandos. Na versão 1.8.x, os comandos `encrypt` requerem um parâmetro `--wrapping-keys` ou `--master-keys`. Nos comandos `decrypt` da versão 1.8.x um parâmetro `--wrapping-keys` é opcional, mas recomendado.

Ao usar um provedor de chaves mestres personalizado, os comandos `encrypt` e `decrypt` exigem os atributos `key` e `provider`. Ao usar AWS KMS keys, os comandos `encrypt` exigem um atributo `key`. Os comandos `decrypt` exigem que um atributo `key` ou um atributo `discovery` sejam definidos com um valor de `true` (mas não ambos). Usar o atributo `key` ao descriptografar é uma [prática recomendada do AWS Encryption SDK](#). Ela é particularmente importante se você estiver descriptografando lotes de mensagens desconhecidas, como aquelas em um bucket do Amazon S3 ou em uma fila do Amazon SQS.

Para obter um exemplo de como usar chaves multirregionais do AWS KMS como chaves de encapsulamento, consulte [Usar AWS KMS keys multirregionais](#).

Attributes: o valor do parâmetro `--wrapping-keys` consiste nos seguintes atributos. O formato é `attribute_name=value`.

chave

Identifica a chave de encapsulamento usada na operação. O formato é um par de `key=ID`. Você pode especificar vários atributos `key` em cada valor do parâmetro `--wrapping-keys`.

- Comandos `encrypt`: todos os comandos `encrypt` exigem o atributo `key`. Quando você usa um AWS KMS key em um comando `encrypt`, o valor do atributo `key` pode ser um ID de chave, um ARN de chave, um nome de alias ou um ARN de alias. Para obter detalhes dos identificadores de chave do AWS KMS, consulte [Identificadores de chave](#) no Guia do desenvolvedor do AWS Key Management Service.
- Comandos `decrypt`: ao descriptografar com as AWS KMS keys, o parâmetro `--wrapping-keys` exige que o valor de um atributo `key` seja definido como um [ARN de chave](#) ou que o valor de um atributo `discovery` seja definido como `true` (mas não ambos). Usar o atributo `key` é uma [prática recomendada do AWS Encryption SDK](#). Ao descriptografar com um provedor de chave mestra personalizado, o atributo `key` é obrigatório.

Note

Para especificar uma chave de encapsulamento do AWS KMS em um comando `decrypt`, o valor do atributo da chave deve ser um ARN de chave. Se você usar uma ID de chave, um ARN de chave ou um ARN de alias, a CLI de criptografia da AWS não reconhecerá a chave de encapsulamento.

Você pode especificar vários atributos `key` em cada valor do parâmetro `--wrapping-keys`. No entanto, qualquer atributo `provider`, `region` e `profile` em um parâmetro `--wrapping-keys` será aplicável a todas as chaves de encapsulamento no valor desse parâmetro. Para especificar chaves de encapsulamento com diferentes valores de atributos, use vários parâmetros `--wrapping-keys` no comando.

discovery

Permite que a CLI de criptografia da AWS use qualquer AWS KMS key para descriptografar a mensagem. O valor de `discovery` pode ser `true` ou `false`. O valor padrão é `false`. O

atributo `discovery` é válido apenas em comandos `decrypt` e somente quando o provedor de chaves mestras for do AWS KMS.

Ao descriptografar com AWS KMS keys, o parâmetro `--wrapping-keys` requer um atributo `key` ou um atributo `discovery` com um valor definido como `true` (mas não ambos). Se você usar o atributo `key`, poderá usar um atributo de `discovery` com um valor definido como `false` para rejeitar explicitamente a descoberta.

- `False` (padrão): quando o atributo `discovery` não for especificado ou seu valor for `false`, a CLI de criptografia da AWS descriptografará a mensagem usando somente as AWS KMS keys especificadas pelo atributo `key` do parâmetro `--wrapping-keys`. Se você não especificar um atributo `key` quando `discovery` for `false`, o comando `decrypt` falhará. Esse valor dá suporte a uma [prática recomendada](#) da CLI de criptografia da AWS.
- `True`: quando o valor do atributo `discovery` é `true`, a CLI de criptografia da AWS obterá os AWS KMS keys dos metadados da mensagem criptografada e usará esses AWS KMS keys para descriptografar a mensagem. O atributo `discovery` com um valor definido como `true` se comporta como as versões da CLI de criptografia da AWS anteriores à versão 1.8.x, que não permitiam que você especificasse uma chave de encapsulamento ao descriptografar. No entanto, sua intenção de usar qualquer AWS KMS key é explícita. Se você especificar um atributo `key` quando `discovery` for `true`, o comando `decrypt` falhará.

O valor `true` pode fazer com que a CLI de criptografia da AWS use AWS KMS keys em diferentes Contas da AWS e regiões ou tente usar AWS KMS keys que o usuário não tenha autorização para uso.

Quando `discovery` for `true`, é uma prática recomendada usar os atributos `discovery-partition` e `discovery-account` para limitar o uso das AWS KMS keys àquelas presentes na Contas da AWS especificadas por você.

`discovery-account`

Limita as AWS KMS keys usadas para descriptografia à Conta da AWS especificadas. O único valor válido para esse atributo é um [ID de Conta da AWS](#).

Esse atributo é opcional e válido somente em comandos `decrypt` com AWS KMS keys em que o atributo `discovery` esteja definido como `true` e o atributo `discovery-partition` esteja especificado.

Cada atributo `discovery-account` usa apenas um ID de Conta da AWS, mas você pode especificar vários atributos de `discovery-account` no mesmo parâmetro de `--wrapping-keys`. Todas as contas especificadas em um determinado parâmetro `--wrapping-keys` devem estar na partição da AWS especificada.

`discovery-partition`

Especifica a partição da AWS para as contas no atributo `discovery-account`. Seu valor deve ser uma partição da AWS, como `aws`, `aws-cn`, ou `aws-gov-cloud`. Para obter mais informações, consulte [Nomes de atributo da Amazon](#) no Referência geral da AWS.

Esse atributo é obrigatório quando você usa o atributo `discovery-account`. Você pode especificar somente um atributo `discovery-partition` em cada parâmetro `--wrapping-keys`. Para especificar Contas da AWS em várias partições, use um parâmetro `--wrapping-keys` adicional.

`provider`

Identifica o [provedor de chaves mestres](#). O formato é um par de `provider=ID`. O valor padrão, `aws-kms`, representa o AWS KMS. Esse atributo é necessário somente quando o provedor de chaves mestras não é o AWS KMS.

`region`

Identifica a Região da AWS de uma AWS KMS key . Esse atributo é válido apenas para AWS KMS keys. É usado apenas quando o identificador da chave não especifica uma região; caso contrário, é ignorado. Quando é usado, ele substitui a região padrão no perfil nomeado da CLI da AWS.

`profile`

Identifica um AWS CLI perfil nomeado [da](#) . Esse atributo é válido apenas para AWS KMS keys. A região no perfil é usada apenas quando o identificador da chave não especifica uma região e não há nenhum atributo `region` no comando.

`--input (-i)`

Especifica o local dos dados a serem criptografados ou descriptografados. Esse parâmetro é obrigatório. O valor pode ser um caminho para um arquivo ou diretório ou um nome de arquivo padrão. Se você estiver redirecionando a entrada para o comando (stdin), use `-`.

Se a entrada não existir, o comando é concluído com êxito sem erro ou aviso.

--recursive (-r, -R)

Executa a operação nos arquivos no diretório de entrada e em seus subdiretórios. Esse parâmetro é necessário quando o valor de `--input` é um diretório.

--decode

Decodifica entrada codificada em Base64.

Se estiver descriptografando uma mensagem que foi criptografada e, em seguida, codificado, você deverá decodificar a mensagem antes de descriptografá-la. Esse parâmetro faz isso para você.

Por exemplo, se você tiver usado o parâmetro `--encode` em um comando `encrypt`, use o parâmetro `--decode` no comando `decrypt` correspondente. Você também pode usar esse parâmetro para decodificar a entrada codificada em Base64 antes de criptografá-la.

--output (-o)

Especifica um destino para a saída. Esse parâmetro é obrigatório. O valor pode ser um nome de arquivo, um diretório existente ou `-`, que grava a saída na linha de comando (`stdout`).

Se o diretório de saída especificado não existir, o comando falhará. Se a entrada contiver subdiretórios, a CLI de criptografia da AWS reproduzirá os subdiretórios sob o diretório de saída especificado.

Por padrão, a CLI de criptografia da AWS substitui arquivos com o mesmo nome. Para alterar esse comportamento, use os parâmetros `--interactive` ou `--no-overwrite`. Para suprimir o aviso de substituição, use o parâmetro `--quiet`.

Note

Se um comando que deve substituir um arquivo de saída falhar, o arquivo de saída será excluído.

--interactive

Solicita antes de substituir o arquivo.

--no-overwrite

Não substitui arquivos. Em vez disso, se o arquivo de saída existir, a CLI de criptografia da AWS ignorará a entrada correspondente.

--sufixo

Especifica um sufixo de nome de arquivo personalizado para arquivos criados pela a CLI de criptografia da AWS. Para indicar nenhum sufixo, use o parâmetro sem um valor (`--suffix`).

Por padrão, quando o parâmetro `--output` não especifica um nome de arquivo, o nome do arquivo de saída tem o mesmo nome que o nome do arquivo de entrada mais o sufixo. O sufixo para comandos `encrypt` é `.encrypted`. O sufixo para comandos `decrypt` é `.decrypted`.

--encode

Aplica codificação de Base64 (de binário para texto) à saída. A codificação impede que o programa shell do host interprete incorretamente caracteres não ASCII no texto de saída.

Use esse parâmetro ao gravar a saída criptografada em `stdout` (`--output -`), especialmente em um console do PowerShell, mesmo quando você estiver redirecionando a saída para outro comando ou salvando-a em uma variável.

--metadata-output

Especifica um local para metadados sobre as operações de criptografia. Insira um caminho e um nome de arquivo. Se o diretório não existir, o comando falhará. Para gravar os metadados na linha de comando (`stdout`), use `-`.

Você não pode gravar a saída do comando (`--output`) e a saída dos metadados (`--metadata-output`) em `stdout` no mesmo comando. Além disso, quando o valor de `--input` ou `--output` for um diretório (sem nomes de arquivos), você não poderá gravar a saída de metadados no mesmo diretório ou em qualquer subdiretório desse diretório.

Se você especificar um arquivo existente, por padrão, CLI de criptografia da AWS acrescentará novos registros de metadados a qualquer conteúdo no arquivo. Esse recurso permite que você crie um único arquivo que contém os metadados de todas as suas operações de criptografia. Para substituir o conteúdo em um arquivo existente, use o parâmetro `--overwrite-metadata`.

A CLI de criptografia da AWS retorna um registro de metadados formatado em JSON para cada operação de criptografia ou decriptografia executada pelo comando. Cada registro de metadados inclui os caminhos completos para os arquivos de entrada e de saída, o contexto de criptografia, o pacote de algoritmos e outras informações valiosas que você pode usar para rever a operação e verificar se ela atende a seus padrões de segurança.

--overwrite-metadata

Substitui o conteúdo no arquivo de saída de metadados. Por padrão, o parâmetro `--metadata-output` acrescenta metadados a qualquer conteúdo existente no arquivo.

--suppress-metadata (-S)

Suprime os metadados sobre a operação de criptografia ou de descryptografia.

--commitment-policy

Especifica a [política de compromisso](#) para comandos `encrypt` e `decrypt`. A política de compromisso determina se sua mensagem será criptografada e descryptografada com o atributo de segurança [confirmação de chave](#).

O parâmetro `--commitment-policy` foi introduzido na versão 1.8.x.. Ele 'é válido em comandos de criptografia e descryptografia.

Na versão 1.8.x, a CLI de criptografia da AWS usa a política de compromisso `forbid-encrypt-allow-decrypt` para todas as operações de criptografia e descryptografia. Quando você usa o parâmetro `--wrapping-keys` em um comando `encrypt` ou `decrypt`, é obrigatório que um parâmetro `--commitment-policy` seja definido com o valor `forbid-encrypt-allow-decrypt`. Se você não usar o parâmetro `--wrapping-keys`, o parâmetro `--commitment-policy` será inválido. Definir uma política de compromisso explicitamente impede que sua política de compromisso seja alterada automaticamente para `require-encrypt-require-decrypt` quando você atualizar para a versão 2.1.x

A partir da versão 2.1.x, todos os valores da política de compromisso são compatíveis. O parâmetro `--commitment-policy` é opcional e o valor padrão é `require-encrypt-require-decrypt`.

Esse parâmetro tem os valores a seguir:

- `forbid-encrypt-allow-decrypt`: não é possível criptografar com confirmação de chave. Ele pode descryptografar textos cifrados criptografados com ou sem confirmação de chave.

Na versão 1.8.x, esse é o único valor válido. A CLI de criptografia da AWS usa a política de compromisso `forbid-encrypt-allow-decrypt` para todas as operações de criptografia e descryptografia.

- `require-encrypt-allow-decrypt`: criptografa somente com confirmação de chave. Descryptografa com e sem compromisso chave. Esse valor foi introduzido na versão 2.1.x..

- `require-encrypt-require-decrypt` (padrão): criptografa e descriptografa somente com confirmação de chave. Esse valor foi introduzido na versão 2.1.x.. É o valor padrão em versões 2.1.x. e posteriores. Com esse valor, a CLI de criptografia da AWS não descriptografará nenhum texto cifrado que tenha sido criptografado com versões anteriores do AWS Encryption SDK.

Para obter informações detalhadas sobre como definir sua política de compromisso, consulte [Como migrar seu AWS Encryption SDK](#).

`--encryption-context (-c)`

Especifica um [contexto de criptografia](#) para a operação. Esse parâmetro não é obrigatório, mas é recomendado.

- Em um comando `--encrypt`, insira um ou mais pares de `name=value`. Use espaços para separar os pares.
- Em um comando `--decrypt`, insira pares de `name=value`, elementos `name` sem valores ou ambos.

Se o `name` ou o `value` em um par de `name=value` incluir espaços ou caracteres especiais, coloque o par inteiro entre aspas. Por exemplo, `--encryption-context "department=software development"`.

`--buffer (-b)` [Introduzido nas versões 1.9.x e 2.2.x]

Retorna texto simples somente após o processamento de todas as entradas, incluindo a verificação da assinatura digital, se houver uma.

`--max-encrypted-data-keys` [Introduzido nas versões 1.9.x e 2.2.x]

Especifica o número máximo de chaves de dados criptografadas em uma mensagem criptografada. Esse parâmetro é opcional.

Os valores válidos são 1–65.535. Se você omitir esse parâmetro, a CLI de criptografia da AWS não aplicará nenhum valor máximo. Uma mensagem criptografada pode conter até 65.535 ($2^{16} - 1$) chaves de dados criptografadas.

Você pode usar esse parâmetro em comandos `encrypt` para evitar a malformação de uma mensagem. Você pode usá-lo em comandos `decrypt` para detectar mensagens maliciosas e evitar descriptografar mensagens com várias chaves de dados criptografadas que você não pode descriptografar. Para obter detalhes e um exemplo, consulte [Limitar as chaves de dados criptografadas](#).

--help (-h)

Imprime o uso e a sintaxe na linha de comando.

--version

Obtém a versão da CLI de criptografia da AWS.

-v | -vv | -vvv | -vvvv

Exibe informações, avisos e mensagens de depuração detalhados. Os detalhes na saída aumentam com o número de vs no parâmetro. A configuração mais detalhada (-vvvv) retorna dados em nível de depuração da CLI de criptografia da AWS e de todos os componentes que ela usa.

--quiet (-q)

Suprime mensagens de aviso, como a mensagem que aparece quando você substitui um arquivo de saída.

--master-keys (-m) [Descontinuado]**Note**

O parâmetro `--master-keys` foi descontinuado na versão 1.8.x e foi removido na versão 2.1.x. Em vez dele, use o parâmetro [--wrapping-keys](#).

Especifica as [chaves mestres](#) usadas em operações de criptografia e descryptografia. Você pode usar vários parâmetros de chaves mestras em cada comando.

O parâmetro `--master-keys` é necessário em comandos `encrypt`. Ele é necessário em comandos `decrypt` somente quando você estiver usando um provedor de chaves mestras personalizado (que não seja do AWS KMS).

Attributes: o valor do parâmetro `--master-keys` consiste nos seguintes atributos. O formato é `attribute_name=value`.

chave

Identifica a [chave de encapsulamento](#) usada na operação. O formato é um par de `key=ID`. O atributo `key` é obrigatório em todos os comandos `encrypt`.

Quando você usa um AWS KMS key em um comando `encrypt`, o valor do atributo `key` pode ser um ID de chave, um ARN de chave, um nome de alias ou um ARN de alias. Para obter

detalhes sobre os identificadores de chave do AWS KMS, consulte [identificadores de chave](#) no Guia do desenvolvedor do AWS Key Management Service.

O atributo `key` é necessário em comandos `decrypt` quando o provedor de chaves mestras não for o AWS KMS. O atributo `key` não é permitido em comandos que descriptografam dados que foram criptografados com uma AWS KMS key.

Você pode especificar vários atributos `key` em cada valor do parâmetro `--master-keys`. No entanto, qualquer atributo `provider`, `region` e `profile` aplica-se a todas as chaves mestras no valor do parâmetro. Para especificar chaves mestras com diferentes valores de atributos, use vários parâmetros `--master-keys` no comando.

`provider`

Identifica o [provedor de chaves mestras](#). O formato é um par de `provider=ID`. O valor padrão, `aws-kms`, representa o AWS KMS. Esse atributo é necessário somente quando o provedor de chaves mestras não é o AWS KMS.

`region`

Identifica a Região da AWS de uma AWS KMS key . Esse atributo é válido apenas para AWS KMS keys. É usado apenas quando o identificador da chave não especifica uma região; caso contrário, é ignorado. Quando é usado, ele substitui a região padrão no perfil nomeado da CLI da AWS.

`profile`

Identifica um AWS CLI perfil nomeado [da](#) . Esse atributo é válido apenas para AWS KMS keys. A região no perfil é usada apenas quando o identificador da chave não especifica uma região e não há nenhum atributo `region` no comando.

Parâmetros avançados

`--algorithm`

Especifica um [pacote de algoritmos](#) alternativo. Esse parâmetro é opcional e válido apenas em comandos `encrypt`.

Se você omitir esse parâmetro, a CLI de criptografia da AWS usará um dos pacotes de algoritmos padrão para o AWS Encryption SDK apresentados na versão 1.8.x.. Ambos os algoritmos padrão usam o AES-GCM com um [HKDF](#), uma assinatura ECDSA e uma chave de criptografia de 256

bits. Um usa confirmação de chave; o outro não. A escolha do pacote de algoritmos padrão é determinada pela [política de compromisso](#) do comando.

Os pacotes de algoritmo padrão são recomendados para a maioria das operações de criptografia. Para obter uma lista de valores válidos, consulte os valores do parâmetro `algorithm` em [Leia os documentos](#).

`--frame-length`

Cria uma saída com o tamanho da moldura especificado. Esse parâmetro é opcional e válido apenas em comandos `encrypt`.

Digite um valor em bytes. Os valores válidos são 0 e $1-2^{31}-1$. Um valor igual a 0 indica dados sem moldura. O padrão é 4.096 (bytes).

Note

Sempre que possível, use dados com moldura. O AWS Encryption SDK oferece suporte dados não emoldurados somente para uso herdado. Algumas implementações de linguagem do AWS Encryption SDK ainda podem gerar texto cifrado sem moldura. Todas as implementações de linguagem compatíveis podem descriptografar texto cifrado e não emoldurado.

`--max-length`

Indica o tamanho máximo da moldura (ou o tamanho máximo do conteúdo de mensagens sem moldura) a ser lido em mensagens criptografadas. Esse parâmetro é opcional e válido apenas em comandos `decrypt`. Ele foi projetado para proteção contra a descriptografia de texto cifrado mal-intencionado extremamente grande.

Digite um valor em bytes. Se você omitir esse parâmetro, o AWS Encryption SDK não limitará o tamanho do quadro ao descriptografar.

`--caching`

Habilita o recurso de [armazenamento em cache de chaves de dados](#), que reutiliza chaves de dados, em vez de gerar uma nova chave de dados para cada arquivo de entrada. Esse parâmetro é compatível com um cenário avançado. Não deixe de ler a documentação [Armazenamento em cache de chaves de dados](#) antes de usar esse recurso.

O parâmetro `--caching` tem os seguintes atributos.

capacity (obrigatório)

Determina o número máximo de entradas no cache.

O valor mínimo é 1. Não há um valor máximo.

max_age (obrigatório)

Determina o tempo em que as entradas do cache são usadas, em segundos, a partir do momento em que são adicionadas ao cache.

Digite um valor maior que 0. Não há um valor máximo.

max_messages_encrypted (opcional)

Determina o número máximo de mensagens que uma entrada armazenada em cache pode criptografar.

Os valores válidos são $1-2^{32}$. O valor padrão é 2^{32} (mensagens).

max_bytes_encrypted (opcional)

Determina o número máximo de bytes que uma entrada armazenada em cache pode criptografar.

Os valores válidos são 0 e $1-2^{63} - 1$. O valor padrão é $2^{63} - 1$ (mensagens). Um valor de 0 permite usar armazenamento em cache de chaves de dados somente quando você está criptografando strings de mensagem vazias.

Versões da CLI de criptografia da AWS

Recomendamos que você use a versão mais recente da CLI de criptografia da AWS.

Note

Todas as versões da CLI de criptografia da AWS anteriores à 4.0.0 estão na fase de [fim do suporte](#).

Você pode atualizar com segurança a partir da versão 2.1.x e posteriores até a versão mais recente da CLI de criptografia da AWS sem realizar alterações no código ou nos dados. No entanto, os [novos atributos de segurança](#) introduzidos na versão 2.1.x não são compatíveis com versões anteriores. Para atualizar a partir da versão 1.7.x ou anteriores, primeiro será necessário atualizar para a versão 1.x mais recente da CLI de criptografia da AWS. Para obter mais detalhes, consulte [Como migrar seu AWS Encryption SDK](#).

Os novos atributos de segurança foram lançados originalmente nas versões 1.7.x e 2.0.x da CLI de criptografia da AWS. No entanto, a versão 1.8x da CLI de criptografia da AWS substituiu a versão 1.7.x e a versão 2.1.x da CLI de criptografia da AWS substituiu a 2.0.x. Para obter detalhes, consulte o [boletim de segurança](#) pertinente no repositório [aws-encryption-sdk-cli](#) no GitHub.

Para obter informações sobre versões significativas do AWS Encryption SDK, consulte [Versões do AWS Encryption SDK](#).

Qual versão devo usar?

Se você é novo na CLI de criptografia da AWS, use a versão mais recente.

Para descriptografar dados criptografados por uma versão do AWS Encryption SDK anterior à 1.7.x, migre primeiro para a versão mais recente da CLI de criptografia da AWS. Faça [todas as alterações recomendadas](#) antes de atualizar para a versão 2.1.x ou versões posteriores. Para obter mais detalhes, consulte [Como migrar seu AWS Encryption SDK](#).

Saiba mais

- Para obter informações detalhadas sobre as alterações e orientações para migrar para essas novas versões, consulte [Como migrar seu AWS Encryption SDK](#).
- Para obter descrições dos novos parâmetros e atributos da CLI de criptografia da AWS, consulte [Referência da sintaxe e de parâmetros da CLI do AWS Encryption SDK](#).

As listas a seguir descrevem a alteração na CLI de criptografia da AWS nas versões 1.8.x e 2.1.x..

Alterações da versão 1.8.x na CLI de criptografia da AWS

- Descontinua o parâmetro `--master-keys`. Em vez disso, use o parâmetro `--wrapping-keys`.
- Adiciona o parâmetro `--wrapping-keys (-w)`. Compatível com todos os atributos do parâmetro `--master-keys`. Também adiciona os seguintes atributos opcionais, que são válidos somente ao descriptografar com AWS KMS keys.
 - `discovery`
 - `discovery-partition`
 - `discovery-account`

Para provedores de chaves mestras personalizadas, os comandos `--encrypt` e `--decrypt` exigem um parâmetro `--wrapping-keys` ou um parâmetro `--master-keys` (mas não ambos). Além disso, um comando `--encrypt` com AWS KMS keys exige um parâmetro `--wrapping-keys` ou um parâmetro `--master-keys` (mas não ambos).

Em um comando `--decrypt` com AWS KMS keys, o parâmetro `--wrapping-keys` é opcional, mas recomendado, pois é obrigatório na versão 2.1.x.. Se você usá-lo, deverá especificar o `key` ou o atributo `discovery` com um valor definido como `true` (mas não ambos).

- Adiciona o parâmetro `--commitment-policy`. O único valor válido é `forbid-encrypt-allow-decrypt`. A política de compromisso `forbid-encrypt-allow-decrypt` é usada em todos os comandos `encrypt` e `decrypt`.

Na versão 1.8.x, quando você usa o parâmetro `--wrapping-keys`, é necessário definir um parâmetro `--commitment-policy` com o valor `forbid-encrypt-allow-decrypt`. Definir o valor explicitamente impede que sua [política de compromisso](#) seja alterada automaticamente para `require-encrypt-require-decrypt` quando você atualizar para a versão 2.1.x.

Alterações da versão 2.1.x na CLI de criptografia da AWS

- Remove o parâmetro `--master-keys`. Em vez disso, use o parâmetro `--wrapping-keys`.
- O parâmetro `--wrapping-keys` é obrigatório em comandos `encrypt`. Você deve especificar o atributo `key` ou o atributo `discovery` com um valor definido como `true` (mas não ambos).
- O parâmetro `--commitment-policy` oferece suporte aos seguintes valores: Para obter mais detalhes, consulte [Como definir sua política de compromisso](#).
 - `forbid-encrypt-allow-decrypt`
 - `require-encrypt-allow-decrypt`
 - `require-encrypt-require-decrypt` (padrão)
- O parâmetro `--commitment-policy` é opcional na versão 2.1.x.. O valor padrão é `require-encrypt-require-decrypt`.

Alterações das versões 1.9x e 2.2.x na CLI de criptografia da AWS

- Adiciona o parâmetro `--decrypt-unsigned`. Para obter mais detalhes, consulte [Versão 2.2x](#).
- Adiciona o parâmetro `--buffer`. Para obter mais detalhes, consulte [Versão 2.2x](#).

- Adiciona o parâmetro `--max-encrypted-data-keys`. Para obter mais detalhes, consulte [Limitar as chaves de dados criptografadas](#).

Alterações da versão 3.0.x na CLI de criptografia da AWS

- Adiciona suporte para chaves de várias regiões do AWS KMS. Para obter mais detalhes, consulte [Usar AWS KMS keys multirregionais](#).

Armazenamento em cache de chaves de dados

O armazenamento em cache de chaves de dados armazena [chaves de dados](#) e o [material criptográfico relacionado](#) em um cache. Quando você criptografa ou descriptografa dados, ele AWS Encryption SDK procura uma chave de dados correspondente no cache. Se encontrar uma correspondência, ele usará a chave de dados armazenada em cache em vez de gerar uma nova. O armazenamento em cache de chaves de dados pode melhorar o desempenho, reduzir os custos e ajudar você a manter os limites do serviço à medida que seu aplicativo é escalado.

O aplicativo poderá se beneficiar do armazenamento em cache de chaves de dados se:

- Puder reutilizar chaves de dados.
- Gerar várias chaves de dados.
- As operações de criptografia estiverem inaceitavelmente lentas, caras, limitadas ou usarem recursos de forma intensiva.

O armazenamento em cache pode reduzir o uso de serviços criptográficos, como AWS Key Management Service (AWS KMS). Se você está atingindo seu [AWS KMS requests-per-second limite](#), o armazenamento em cache pode ajudar. Seu aplicativo pode usar chaves em cache para atender a algumas de suas solicitações de chave de dados em vez de chamar AWS KMS. (Você também pode criar um caso no [AWS Support Center](#) para aumentar o limite da conta.)

AWS Encryption SDK Isso ajuda você a criar e gerenciar seu cache de chaves de dados. Ele fornece um [cache local](#) e um [gerenciador de armazenamento em cache de materiais criptográficos](#) (CMM de armazenamento em cache) que interage com o cache e impõe [limites de segurança](#) definidos por você. Juntos, esses componentes ajudam você a se beneficiar da eficiência de reutilização de chaves de dados mantendo a segurança do sistema.

O armazenamento em cache da chave de dados é um recurso opcional do AWS Encryption SDK que você deve usar com cautela. Por padrão, AWS Encryption SDK gera uma nova chave de dados para cada operação de criptografia. Essa técnica é compatível com as melhores práticas criptográficas, que desencorajam a reutilização excessiva de chaves de dados. Em geral, use o armazenamento em cache de chaves de dados somente quando ele for necessário para atender às suas metas de desempenho. Em seguida, use os [limites de segurança](#) do armazenamento em cache de chaves de dados para garantir que você use a quantidade mínima de armazenamento em cache necessário para atender a suas metas de desempenho e custo.

[O CMM de cache não é compatível com o.NET AWS Encryption SDK](#) . Versão 3. x of the suporta AWS Encryption SDK for Java apenas o CMM de armazenamento em cache com a interface antiga de provedores de chaves mestras, não a interface de chaveiro. No entanto, versão 4. x do AWS Encryption SDK para .NET e versão 3. x do AWS Encryption SDK for Java suporte ao [AWS KMS chaveiro hierárquico](#), uma solução alternativa de cache de materiais criptográficos. O conteúdo criptografado com o AWS KMS chaveiro hierárquico só pode ser descriptografado com o chaveiro hierárquico. AWS KMS

Para ver uma discussão detalhada dessas vantagens e desvantagens de segurança, consulte [AWS Encryption SDK: Como decidir se o armazenamento em cache de chaves de dados é ideal para sua aplicação](#) no Blog de segurança da AWS .

Tópicos

- [Como usar o armazenamento em cache de chaves de dados](#)
- [Definir limites de segurança do cache](#)
- [Detalhes do armazenamento em cache de chaves de dados](#)
- [Exemplo de armazenamento em cache de chaves de dados](#)

Como usar o armazenamento em cache de chaves de dados

Este tópico mostra como usar o armazenamento em cache de chaves de dados em seu aplicativo. Ele fornece uma demonstração passo a passo do processo. Em seguida, ele combina as etapas em um exemplo simples que usa o armazenamento em cache da chave de dados em uma operação para criptografar uma string.

Esses exemplos mostram como usar a versão 2.0.x e versões posteriores do AWS Encryption SDK. Para exemplos que usam versões anteriores, encontre sua versão na lista de [lançamentos](#) do GitHub repositório da sua [linguagem de programação](#).

Para obter exemplos completos e testados de como usar o armazenamento em cache de chaves de dados no AWS Encryption SDK, consulte:

- C/C++: [caching_cmm.cpp](#)
- Java: [SimpleDataKeyCachingExample.java](#)
- JavaScript Navegador: [caching_cmm.ts](#)
- JavaScript Node.js: [caching_cmm.ts](#)
- Python: [data_key_caching_basic.py](#)

O [AWS Encryption SDK para .NET](#) não oferece é compatível com o cache de chaves de dados.

Tópicos

- [Usando o cache de chaves de dados: S tep-by-step](#)
- [Armazenamento em cache de chaves de dados de exemplo: criptografar uma string](#)

Usando o cache de chaves de dados: S tep-by-step

Essas step-by-step instruções mostram como criar os componentes necessários para implementar o armazenamento em cache de chaves de dados.

- [Crie um cache de chave de dados](#). Nestes exemplos, usamos o cache local fornecido pelo AWS Encryption SDK. Limitamos o cache a 10 chaves de dados.

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

O exemplo a seguir usa a versão 2. x do AWS Encryption SDK for Java. Versão 3. x do AWS Encryption SDK for Java desaprova o CMM de armazenamento em cache da chave de dados. Com a versão 3. x, você também pode usar o [AWS KMSchaveiro hierárquico](#), uma solução alternativa de cache de materiais criptográficos.

```
// 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)
```

- Crie um [provedor de chave mestra](#) (Java e Python) ou um [chaveiro](#) (C e JavaScript). Estes exemplos usam um provedor de chaves mestras do AWS Key Management Service (AWS KMS) ou um [token de autenticação do AWS KMS](#) compatível.

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

O exemplo a seguir usa a versão 2. x do AWS Encryption SDK for Java. Versão 3. x do AWS Encryption SDK for Java desaprova o CMM de armazenamento em cache da chave de dados. Com a versão 3. x, você também pode usar o [AWS KMS chaveiro hierárquico](#), uma solução alternativa de cache de materiais criptográficos.

```
// 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

No navegador, você deve injetar suas credenciais com segurança. Este exemplo define credenciais em um webpack (`kms.webpack.config`) que resolve credenciais no runtime. Ele cria uma instância de provedor de cliente do AWS KMS de um cliente do AWS KMS e das credenciais. Ao criar o token de autenticação, ele passa o provedor cliente para o construtor juntamente com a 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])
```

- [Crie um gerenciador de armazenamento em cache de materiais criptográficos](#) (CMM de armazenamento em cache).

Associe o CMM de armazenamento em cache ao seu cache e seu provedor de chaves mestras. Em seguida, [defina os limites de segurança do cache](#) no CMM de armazenamento em cache.

C

No AWS Encryption SDK for C, você pode criar um CMM de armazenamento em cache a partir de um CMM subjacente, como o CMM padrão, ou de um token de autenticação. Este exemplo cria o CMM de armazenamento em cache de um token de autenticação.

Depois de criar o CMM de armazenamento em cache, você pode liberar suas referências para o token de autenticação e o cache. Para obter detalhes, consulte [the section called “Contagem de referências”](#).

```
// 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

O exemplo a seguir usa a versão 2. x do AWS Encryption SDK for Java. Versão 3. x of the AWS Encryption SDK for Java não oferece suporte ao cache de chaves de dados, mas suporta o [AWS KMSchaveiro hierárquico](#), uma solução alternativa de cache de materiais criptográficos.

```
/*
 * 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  
)
```

Isso é tudo o que você precisa fazer. Deixe que o AWS Encryption SDK gerencie o cache para você ou adicione sua própria lógica de gerenciamento de cache.

Quando desejar usar o armazenamento em cache de chaves de dados em uma chamada para criptografar ou descriptografar dados, especifique o CMM de armazenamento em cache em vez de especificar um provedor de chaves mestras ou outro CMM.

Note

Se estiver criptografando streamings de dados ou quaisquer dados de tamanho desconhecido, certifique-se de especificar o tamanho dos dados na solicitação. O AWS Encryption SDK não usa armazenamento em cache de chave de dados ao criptografar dados de tamanho desconhecido.

C

No AWS Encryption SDK for C, crie uma sessão com o CMM de armazenamento em cache e processe a sessão.

Por padrão, quando o tamanho da mensagem é desconhecido e não vinculado, o AWS Encryption SDK não armazena chaves de dados em cache. Para permitir o armazenamento em cache quando não se sabe o tamanho exato dos dados, use o método `aws_cryptosdk_session_set_message_bound` para definir o tamanho máximo da mensagem. Defina o vínculo maior do que o tamanho estimado da mensagem. Se o tamanho real da mensagem exceder o vínculo, ocorrerá uma falha na operação da criptografia.

```
/* 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

O exemplo a seguir usa a versão 2. x do AWS Encryption SDK for Java. Versão 3. x do AWS Encryption SDK for Java desaprova o CMM de armazenamento em cache da chave de dados. Com a versão 3. x, você também pode usar o [AWS KMSchaveiro hierárquico](#), uma solução alternativa de cache de materiais criptográficos.

```
// 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

Ao usar o CMM de armazenamento em cache no AWS Encryption SDK para JavaScript para Node.js, o método `encrypt` requer o tamanho do texto simples. Se você não fornecer, a chave de dados não será armazenada em cache. Se você fornecer um tamanho, mas os dados de texto simples fornecidos excederem esse tamanho, a operação de criptografia falhará. Se você não souber o tamanho exato do texto simples, como quando estiver fazendo streaming de dados, forneça o maior valor esperado.

```
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
)
```

Armazenamento em cache de chaves de dados de exemplo: criptografar uma string

Este código de exemplo simples usa o armazenamento em cache de chaves de dados ao criptografar uma string. Ele combina o código do [step-by-step procedimento](#) em um código de teste que você pode executar.

O exemplo cria um [cache local](#) e um [provedor de chave mestra](#) ou [token de autenticação](#) para uma AWS KMS key. Em seguida, ele usa o cache local e o provedor de chaves mestras ou o token de autenticação para criar um CMM de armazenamento em cache com os [limites de segurança](#) adequados. Em Java e em Python, a solicitação de criptografia especifica o CMM de armazenamento em cache, os dados de texto simples a serem criptografados e um [contexto de criptografia](#). Em C, o CMM de armazenamento em cache é especificado na sessão, e a sessão é fornecida para a solicitação de criptografia.

Para executar estes exemplos, você precisa fornecer o [nome do atributo da Amazon \(ARN\) de uma AWS KMS key](#). Verifique se você tem [permissão para usar a AWS KMS key](#) para gerar uma chave de dados.

Para obter exemplos reais mais detalhados de como criar e usar um cache de chave mestra, consulte [Exemplo de código de armazenamento em cache de chaves de dados](#).

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

O exemplo a seguir usa a versão 2. x do AWS Encryption SDK for Java. Versão 3. x do AWS Encryption SDK for Java desaprova o CMM de armazenamento em cache da chave de dados. Com a versão 3. x, você também pode usar o [AWS KMS Chaveiro hierárquico](#), uma solução alternativa de cache de materiais criptográficos.

```
// 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.kmsdkv2.KmsMasterKey;
import com.amazonaws.encryptionsdk.kmsdkv2.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 testCachingCMMEExample() {
  /* 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 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 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
```

Definir limites de segurança do cache

Quando você implementa o armazenamento em cache de chave de dados, precisa configurar os limites de segurança impostos pelo [CMM de armazenamento em cache](#).

Os limites de segurança ajudam a limitar duração do uso de cada chave de dados e o volume de dados protegido em cada chave de dados. O CMM de armazenamento em cache retorna as chaves de dados armazenadas em cache somente quando a entrada do cache estiver em conformidade com todos os limites de segurança. Se a entrada do cache exceder o limite, ela não será usada para a operação atual e será removida do cache assim que possível. O primeiro uso de cada chave de dados (antes do armazenamento em cache) é isento desses limites.

Como regra, use a quantidade mínima de armazenamento em cache necessária para atender a suas metas de custos e de desempenho.

O AWS Encryption SDK armazena somente chaves de dados criptografadas usando uma [função de derivação de chaves](#). Além disso, ele estabelece limites máximos para alguns dos valores de limites. Essas restrições garantem que as chaves de dados não sejam reutilizadas além dos limites criptográficos. No entanto, como as chaves de dados de texto sem criptografia são armazenadas em cache (na memória, por padrão), tente minimizar o tempo em que as chaves são salvas. Além disso, tente limitar os dados que poderão ser expostos se uma chave estiver comprometida.

Para obter exemplos de como definir limites de segurança de cache, consulte [:AWS Encryption SDK: como decidir se o cache de chaves de dados é adequado para sua aplicação](#) no Blog de segurança da AWS.

Note

O CMM do armazenamento em cache impõe todos os limites a seguir. Se você não especificar um valor opcional, o CMM de armazenamento em cache usará o valor padrão. Para desativar temporariamente o armazenamento em cache de chaves de dados, as implementações de Java e Python do AWS Encryption SDK fornecem um cache de materiais criptográficos nulo (cache nulo). O cache nulo retorna um erro para cada solicitação GET e não responde a solicitações PUT. Recomendamos usar o cache nulo em vez de definir a [capacidade do cache](#) ou os limites de segurança como 0. Para obter mais informações, consulte o cache nulo em [Java](#) e [Python](#).

Idade máxima (obrigatório)

Determina por quanto tempo uma entrada armazenada em cache pode ser usada, a partir do momento em que foi adicionada. Este valor é obrigatório. Digite um valor maior que 0. O AWS Encryption SDK não limita o valor de idade máxima.

Todas as implementações de linguagem do AWS Encryption SDK definem a idade máxima em segundos, exceto o AWS Encryption SDK para JavaScript, que usa milissegundos.

Use o intervalo mais curto que ainda permita que seu aplicativo se beneficie do cache. Você pode usar o limite máximo de idade como uma política de rotação de chaves. Use-o para limitar a reutilização de chaves de dados, minimizar a exposição de material criptográfico e remover chaves de dados cujas políticas podem ter sido alteradas enquanto estavam armazenadas em cache.

Número máximo de mensagens criptografadas (opcional)

Especifica o número máximo de mensagens que uma chave de dados armazenada em cache pode criptografar. Este valor é opcional. Digite um valor entre 1 e 2^{32} mensagens. O valor padrão é 2^{32} mensagens.

Defina o número de mensagens protegidas por cada chave armazenada em cache para que seja grande o suficiente para obter o valor da reutilização, mas pequeno o suficiente para limitar o número de mensagens que podem ser expostas se uma chave for comprometida.

Número máximo de bytes criptografados (opcional)

Especifica o número máximo de bytes que uma chave de dados armazenada em cache pode criptografar. Este valor é opcional. Digite um valor entre 0 e $2^{63} - 1$. O valor padrão é $2^{63} -$

1. Um valor de 0 permite usar armazenamento em cache de chaves de dados somente quando você está criptografando strings de mensagem vazias.

Os bytes na solicitação atual são incluídos ao avaliar esse limite. Se os bytes processados, mais os bytes atuais, excederem o limite, a chave de dados armazenada em cache será removida do cache, mesmo que ela tenha sido usada em uma solicitação menor.

Detalhes do armazenamento em cache de chaves de dados

A maioria dos aplicativos pode usar a implementação padrão do armazenamento em cache de chave de dados sem escrever código personalizado. Esta seção descreve a implementação padrão e alguns detalhes sobre as opções.

Tópicos

- [Como o armazenamento em cache de chaves de dados funciona](#)
- [Criar um cache de material de criptografia](#)
- [Criar um gerenciador de material de criptografia de armazenamento em cache](#)
- [O que é uma entrada de chave de dados em cache?](#)
- [Contexto de criptografia: como selecionar entradas do cache](#)
- [Meu aplicativo está usando chaves de dados armazenadas em cache?](#)

Como o armazenamento em cache de chaves de dados funciona

Quando você usa o armazenamento em cache de chave de dados em uma solicitação para criptografar ou descriptografar dados, o AWS Encryption SDK primeiro pesquisa uma chave de dados no cache que corresponde à solicitação. Se localizar uma correspondência válida, ele usa a chave de dados armazenada em cache para criptografar os dados. Caso contrário, ele gerará uma nova chave de dados, da mesma forma como o faria sem o cache.

O armazenamento em cache da chave de dados não é usado para dados de tamanho desconhecido, como streaming de dados. Isso permite que o CMM de armazenamento em cache imponha o [limite máximo de bytes](#) corretamente. Para evitar esse comportamento, adicione o tamanho da mensagem à solicitação de criptografia.

Além de um cache, o armazenamento em cache de chaves de dados usa [um gerenciador de armazenamento em cache de materiais criptográficos](#) (CMM de armazenamento em cache). O CMM

de armazenamento em cache é um [gerenciador de materiais criptográficos \(CMM\)](#) especializado que interage com um [cache](#) e um [CMM](#) subjacente. (Quando você especifica um [provedor de chaves mestra](#) ou um token de autenticação, o AWS Encryption SDK cria um CMM padrão para você.) O CMM de armazenamento em cache armazena em cache as chaves de dados que seu CMM subjacente retorna. Também impõe limites de segurança de cache definidos por você.

Para evitar que a chave de dados incorreta seja selecionada no cache, todos os CMMs de armazenamento em cache compatíveis exigem que as seguintes propriedades dos materiais criptográficos em cache correspondam à solicitação de material.

- [Pacote de algoritmos](#)
- [Contexto de criptografia](#) (mesmo quando vazio)
- Nome da partição (uma string que identifica o CMM de armazenamento em cache)
- (Somente descritografia) chaves de dados criptografadas

Note

O AWS Encryption SDK armazena em cache chaves de dados somente quando o [pacote de algoritmos](#) usa uma [função de derivação de chaves](#).

Os seguintes fluxos de trabalho mostram como uma solicitação para criptografar dados é processada com e sem armazenamento em cache da chave de dados. Eles mostram como o armazenamento em cache de componentes que você cria, incluindo o cache e o CMM de armazenamento em cache, são usados no processo.

Criptografar dados sem armazenamento em cache

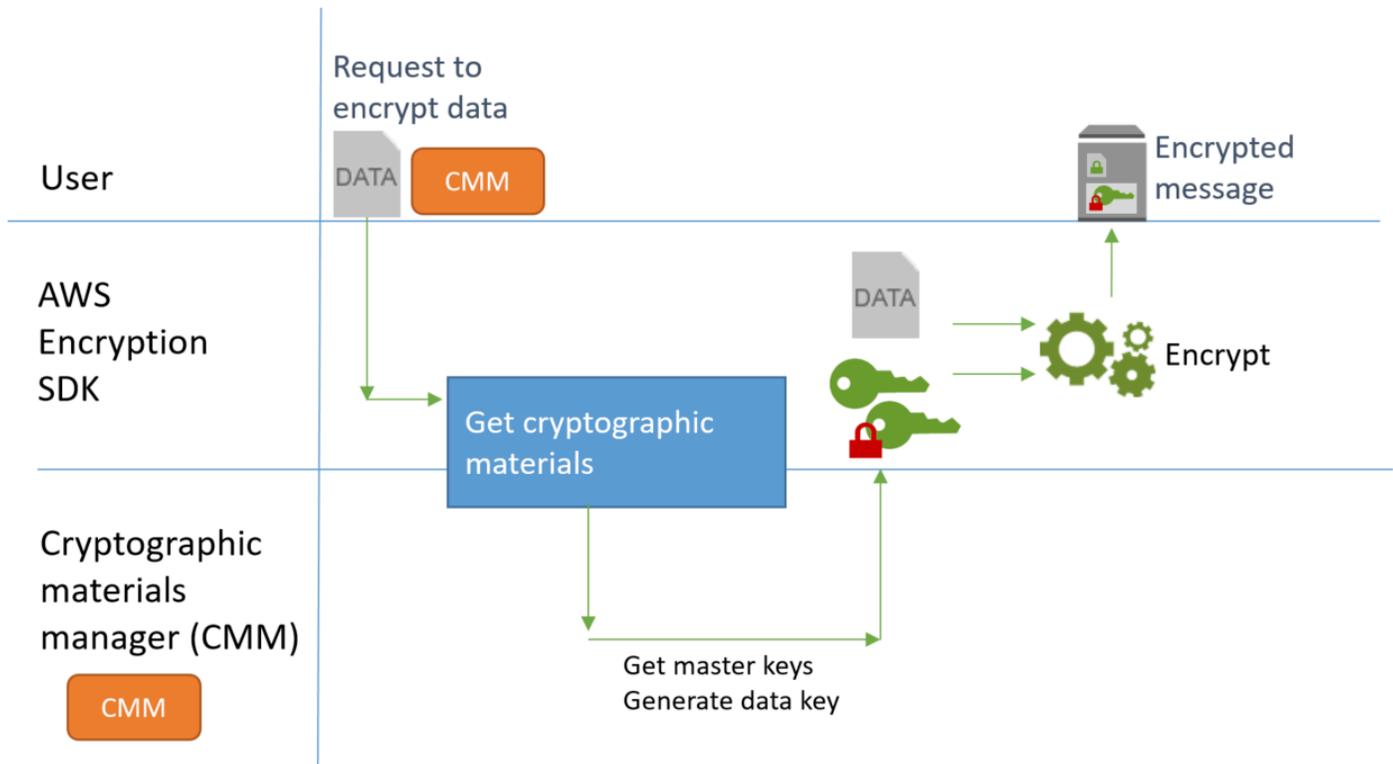
Para obter materiais de criptografia sem armazenamento em cache:

1. Um aplicativo solicita que o AWS Encryption SDK criptografe os dados.

A solicitação especifica um provedor de chaves mestres ou um token de autenticação. O AWS Encryption SDK cria um CMM padrão que interage com a chave mestra ou com o token de autenticação.

2. O AWS Encryption SDK solicitará ao CMM materiais de criptografia (obter materiais criptográficos).

- O CMM solicita seu [token de autenticação](#) (C e JavaScript) ou o [provedor de chaves mestras](#) (Java e Python) para materiais criptográficos. Isso pode envolver uma chamada a um serviço criptográfico, como o AWS Key Management Service (AWS KMS). O CMM retorna os materiais de criptografia para o AWS Encryption SDK.
- O AWS Encryption SDK usa a chave de dados em texto simples para criptografar os dados. Ele armazena os dados criptografados e as chaves de dados criptografadas em uma [mensagem criptografada](#), que ele retorna ao usuário.



Criptografar dados com armazenamento em cache

Para obter materiais de criptografia com armazenamento de chaves de dados em cache:

- Um aplicativo solicita que o AWS Encryption SDK criptografe os dados.

A solicitação especifica um [gerenciador de armazenamento em cache materiais criptográficos \(CMM de armazenamento em cache\)](#) associado a um gerenciador de materiais criptográficos (CMM) subjacente. Quando você especifica um provedor de chaves mestras ou um token de autenticação, o AWS Encryption SDK cria um CMM padrão para você.

- O SDK solicita ao CMM de armazenamento em cache especificado materiais de criptografia.

3. O CMM de armazenamento em cache solicita materiais de criptografia do cache.
 - a. Se encontrar uma correspondência, o cache atualizará a idade e usará os valores da entrada do cache correspondente, retornando os materiais de criptografia armazenados em cache ao CMM de armazenamento em cache.

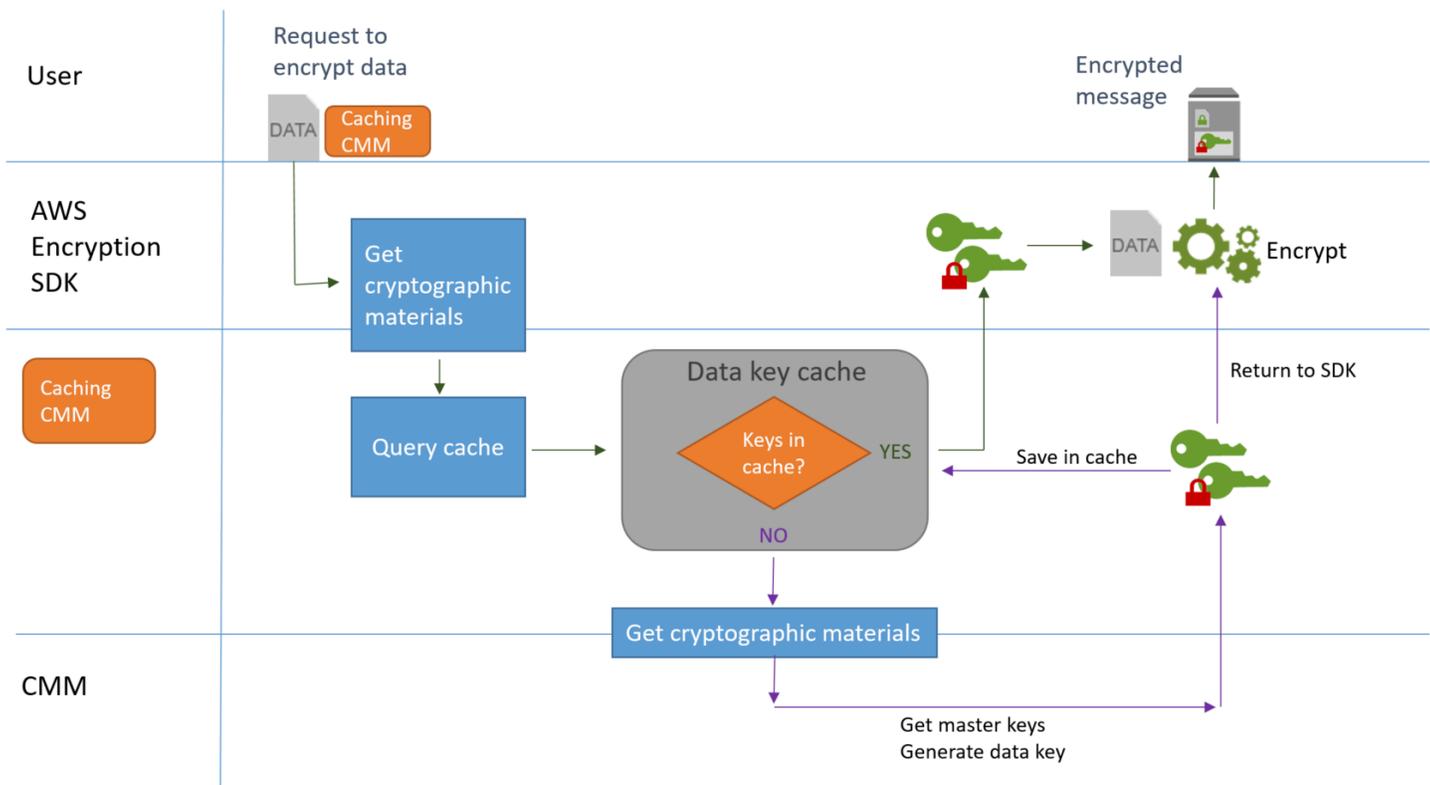
Se a entrada do cache estiver em conformidade com os [limites de segurança](#), o CMM de armazenamento em cache a retorna ao SDK. Caso contrário, ele instruirá o cache a remover a entrada e prosseguir como se não houvesse correspondência.

- b. Se o cache não puder encontrar uma correspondência válida, o CMM de armazenamento em cache solicitará que CMM subjacente gere uma nova chave de dados.

O CMM subjacente obtém os materiais criptográficos de seu token de autenticação (C e JavaScript) ou do provedor de chaves mestras (Java e Python). Isso pode envolver uma chamada a um serviço criptográfico, como o AWS Key Management Service. O CMM subjacente retorna o texto simples e cópias criptografadas da chave de dados ao CMM de armazenamento em cache.

O CMM de armazenamento em cache salva os novos materiais de criptografia no cache.

4. O CMM de armazenamento em cache retorna os materiais de criptografia para o AWS Encryption SDK.
5. O AWS Encryption SDK usa a chave de dados em texto simples para criptografar os dados. Ele armazena os dados criptografados e as chaves de dados criptografadas em uma [mensagem criptografada](#), que ele retorna ao usuário.



Criar um cache de material de criptografia

O AWS Encryption SDK define os requisitos para um cache de material de criptografia usado no armazenamento em cache de chaves de dados. Também fornece um cache local, que é um [cache least recently used \(LRU - menos usado recentemente\)](#) configurável e na memória. Para criar uma instância do cache local use o construtor do `LocalCryptoMaterialsCache` em Java e Python, a função `getLocalCryptographicMaterialsCache` em JavaScript ou o construtor `aws_cryptosdk_materials_cache_local_new` em C.

O cache local contém lógica para gerenciamento básico do cache, incluindo adição, remoção e correspondência de entradas armazenadas em cache e manutenção do cache. Você não precisa escrever nenhuma lógica de gerenciamento de cache personalizado. O cache local pode ser usado como está, ser personalizado ou substituído por cache compatível.

Quando cria um cache local, você define sua capacidade, isto é, o número máximo de entradas que o cache pode conter. Essa configuração ajuda a criar um cache eficiente com reutilização limitada de chaves de dados.

O AWS Encryption SDK for Java e o AWS Encryption SDK for Python também fornecem um cache de materiais criptográficos nulo (`NullCryptoMaterialsCache`). O `NullCryptoMaterialsCache`

retorna um erro para todas as operações GET e não responde a operações PUT. Você pode usar o `NullCryptoMaterialsCache` em testes ou desabilitar temporariamente o armazenamento em cache em uma aplicação que inclui código de armazenamento em cache.

No AWS Encryption SDK, cada cache de materiais criptográficos é associado a um [gerenciador armazenamento em cache de materiais criptográficos](#) (CMM de armazenamento em cache). O CMM de armazenamento em cache obtém chaves de dados do cache, coloca chaves de dados no cache e impõe os [limites de segurança](#) que você define. Ao criar um CMM de armazenamento em cache, você especifica o cache que ele usa e o CMM subjacente ou o provedor de chaves mestras que gera as chaves de dados que ele armazena em cache.

Criar um gerenciador de material de criptografia de armazenamento em cache

Para habilitar o armazenamento em cache da chave de dados, você cria um [cache](#) e um gerenciador de armazenamento em cache (CMM de armazenamento em cache). Em seguida, em suas solicitações para criptografar ou descriptografar dados, você especifica um CMM de armazenamento em cache em vez de um [gerenciador de materiais criptográficos \(CMM\)](#) padrão, um [provedor de chaves mestras](#) ou um [token de autenticação](#).

Há dois tipos de CMMs. Os dois obtêm chaves de dados (e o material criptográfico relacionado), mas de diferentes maneiras, da seguinte forma:

- Um CMM é associado a um token de autenticação (C ou JavaScript) ou a um provedor de chaves mestras (Java e Python). Quando o SDK solicita ao CMM materiais de criptografia ou descriptografia, o CMM obtém os materiais de seu token de autenticação ou do provedor de chaves mestras. Em Java e Python, o CMM usa as chaves mestras para gerar, criptografar ou descriptografar as chaves de dados. Em C e em JavaScript, o token de autenticação gera, criptografa e retorna os materiais criptográficos.
- Um CMM de armazenamento em cache está associado a um cache, como um [cache local](#) e a um CMM subjacente. Quando o SDK solicita materiais criptográficos ao CMM de armazenamento em cache, o CMM de armazenamento em cache tenta obtê-los do cache. Se não conseguir encontrar uma correspondência, o CMM de armazenamento em cache solicitará os materiais ao seu CMM subjacente. Depois, ele armazenará os novos materiais criptográficos antes de retorná-los ao chamador.

O CMM de armazenamento em cache também impõe [limites de segurança](#) que você define para cada entrada do cache. Como os limites de segurança são definidos e impostos pelo CMM de armazenamento em cache, você pode usar qualquer cache compatível, mesmo que o cache não esteja projetado para material confidencial.

O que é uma entrada de chave de dados em cache?

O cache de chaves de dados armazena chaves de dados e o material criptográfico relacionado em um cache. Cada entrada inclui os elementos listados a seguir. Você pode considerar essas informações úteis ao decidir se deseja usar o atributo de armazenamento em cache de chave de dados e ao configurar os limites de segurança em um gerenciador de armazenamento em cache de materiais criptográficos (CMM de armazenamento em cache).

Entradas armazenadas em cache para solicitações de criptografia

As entradas adicionadas a um cache de chave de dados como resultado de uma operação de criptografia incluem os seguintes elementos:

- Chave de dados de texto não criptografado
- Chaves de dados criptografadas (uma ou mais)
- [Contexto de criptografia](#)
- Chave de assinatura de mensagem (se uma for usada)
- [Pacote de algoritmos](#)
- Metadados, incluindo contadores de uso para impor limites de segurança

Entradas armazenadas em cache para solicitações de descriptografia

As entradas adicionadas a um cache de chave de dados como resultado de uma operação de descriptografia incluem os seguintes elementos:

- Chave de dados de texto não criptografado
- Chave de verificação de assinatura (se uma for usada)
- Metadados, incluindo contadores de uso para impor limites de segurança

Contexto de criptografia: como selecionar entradas do cache

Você pode especificar um contexto de criptografia em qualquer solicitação para criptografar dados. No entanto, o contexto de criptografia desempenha uma função especial no armazenamento em cache de chaves de dados. Ele permite criar subgrupos de chaves de dados em seu cache, mesmo quando as chaves de dados forem originárias do mesmo CMM de armazenamento em cache.

Um [contexto de criptografia](#) é um conjunto de pares de chave-valor que contêm dados arbitrários não secretos. Durante a criptografia, o contexto de criptografia é associado de maneira criptográfica aos dados criptografados de forma que o mesmo contexto de criptografia é necessário para descriptografar os dados. No AWS Encryption SDK, o contexto de criptografia é armazenado na [mensagem criptografada](#) com os dados criptografados e as chaves de dados.

Ao usar um cache de chave de dados, você também pode usar o contexto de criptografia para selecionar chaves de dados armazenadas em cache específicas para suas operações de criptografia. O contexto de criptografia é salvo na entrada do cache com a chave de dados (ele faz parte do ID de entrada do cache). As chaves de dados armazenadas em cache só são reutilizadas quando os contextos de criptografia correspondem. Se desejar reutilizar determinadas chaves de dados para uma solicitação de criptografia, especifique o mesmo contexto de criptografia. Para evitar essas chaves de dados, especifique outro contexto de criptografia.

O contexto de criptografia é sempre opcional, mas é recomendado. Se você não especificar um contexto de criptografia na solicitação, um contexto de criptografia vazio será incluído no identificador de entrada do cache e correspondido a cada solicitação.

Meu aplicativo está usando chaves de dados armazenadas em cache?

O armazenamento em cache de chaves de dados é uma estratégia de otimização muito eficaz para determinados aplicativos e cargas de trabalho. No entanto, como isso implica algum risco, é importante determinar o quão eficaz é provável que seja para a sua situação e decidir se os benefícios superam os riscos.

Como o armazenamento em cache de chaves de dados reutiliza chaves de dados, o efeito mais óbvio é a redução do número de chamadas para gerar novas chaves de dados. Quando o armazenamento em cache da chave de dados é implementado, o AWS Encryption SDK chama a operação `GenerateDataKey` do AWS KMS apenas para criar a chave de dados inicial e quando o cache falha. Mas, o armazenamento em cache melhora o desempenho de forma perceptível somente em aplicativos que geram várias chaves de dados com as mesmas características, incluindo o mesmo contexto de criptografia e pacote de algoritmos.

Para determinar se a implementação do AWS Encryption SDK está realmente usando chaves de dados do cache, tente as seguintes técnicas.

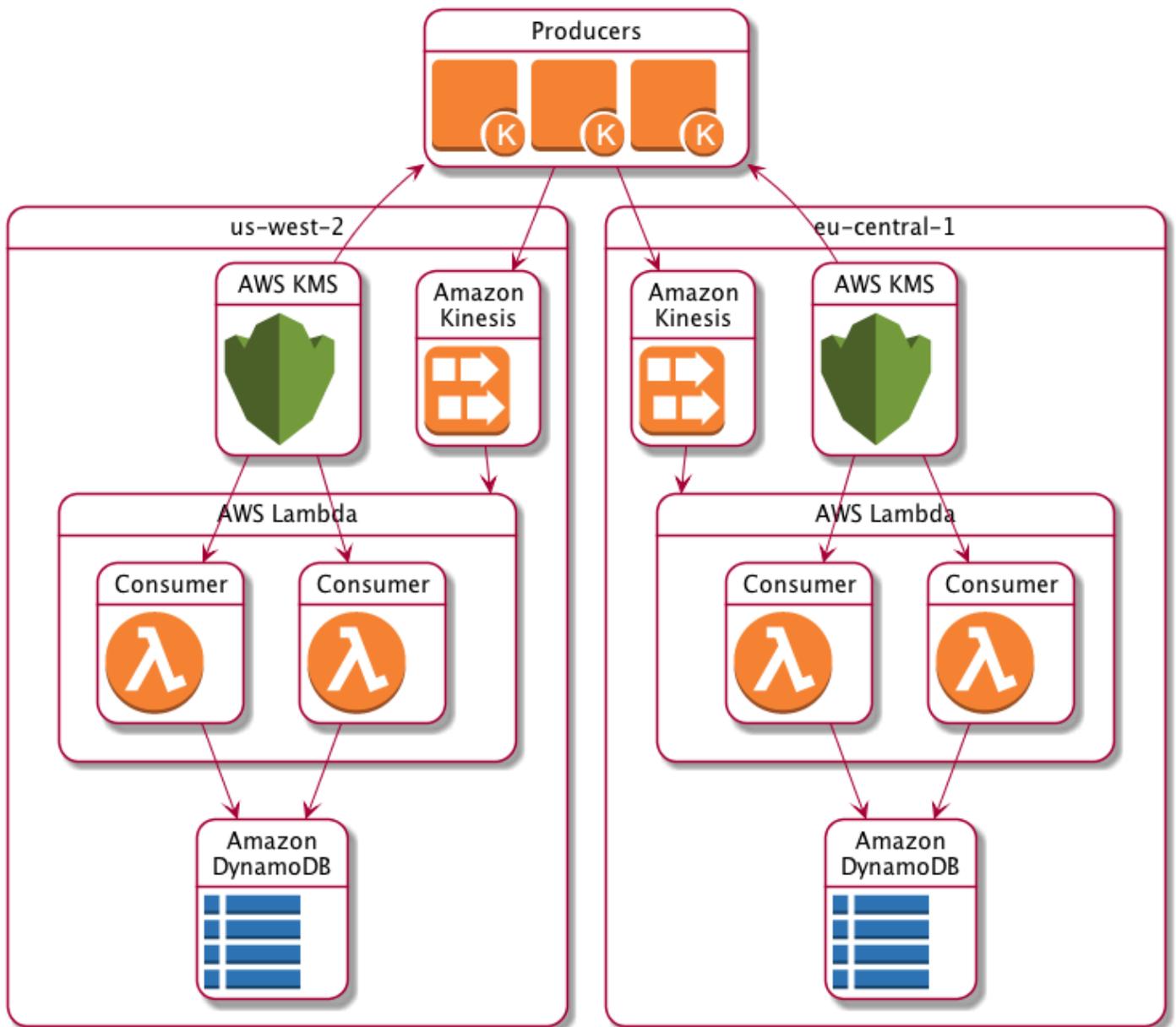
- Nos logs da infraestrutura de sua chave mestra, verifique a frequência de chamadas para criar novas chaves de dados. Quando o armazenamento em cache de chaves de dados está efetivo, o número de chamadas para criar novas chaves deve cair de forma perceptível. Por exemplo, se você estiver usando um provedor de chaves mestras do AWS KMS ou um token de autenticação, pesquise chamadas [GenerateDataKey](#) nos logs do CloudTrail.
- Compare as [mensagens criptografadas](#) que o AWS Encryption SDK retorna em resposta a diferentes solicitações de criptografia. Por exemplo, se você estiver usando o AWS Encryption SDK for Java, compare o objeto [ParsedCiphertext](#) de diferentes chamadas de criptografia. No AWS Encryption SDK para JavaScript, compare o conteúdo da propriedade `encryptedDataKeys` do [MessageHeader](#). Quando as chaves de dados são reutilizadas, as chaves de dados criptografadas na mensagem criptografada são idênticas.

Exemplo de armazenamento em cache de chaves de dados

Este exemplo usa [armazenamento em cache de chaves de dados](#) com um [cache local](#) para acelerar uma aplicação em que os dados gerados por vários dispositivos são criptografados e armazenados em diferentes regiões.

Nesse cenário, vários produtores de dados geram, criptografam e gravam dados em um [stream do Kinesis](#) em cada região. As funções do [AWS Lambda](#) (consumidoras) descriptografam os streams e gravam dados de texto simples em uma tabela do DynamoDB na região. Os produtores e os consumidores de dados usam o AWS Encryption SDK e um [AWS KMSprovedor de chaves mestras do](#) . Para reduzir as chamadas ao KMS, cada produtor e consumidor tem seu próprio armazenamento em cache local.

Você pode encontrar o código-fonte desses exemplos em [Java e Python](#). O exemplo também inclui um modelo do AWS CloudFormation que define os recursos para os exemplos.



Resultados do cache local

A tabela a seguir mostra que um armazenamento em cache local reduz o total de chamadas ao KMS (por segundo por região) neste exemplo em 1% de seu valor original.

Solicitações de produtores

Solicitações por segundo por cliente	Cientes por região	Média de solicitações
--------------------------------------	--------------------	-----------------------

	Gerar chaves de dados (us-west-2)	Criptografar chave de dados (eu-central-1)	Total (por região)		por segundo por região
Sem cache	1	1	1	500	500
Cache local	1 rps/100 usos	1 rps/100 usos	1 rps/100 usos	500	5

Solicitações de consumidor

	Solicitações por segundo por cliente			Cliente por região	Média de solicitações por segundo por região
	Descritor grafar chave de dados	Produtores	Total		
Sem cache	1 rps por produtor	500	500	2	1.000
Cache local	1 rps por produtor/100 usos	500	5	2	10

Exemplo de código de armazenamento em cache de chaves de dados

Este exemplo de código cria uma implementação básica do armazenamento em cache de chaves de dados com um [cache local](#) em Java e Python. O código cria duas instâncias de um armazenamento em cache local: uma para [produtores de dados](#) que estão criptografando dados e outra para [consumidores de dados](#) (funções do AWS Lambda) que estão descriptografando dados. Para obter detalhes sobre a implementação do armazenamento em cache de chaves de dados em cada linguagem, consulte a documentação de [Javadoc](#) e [Python](#) para o AWS Encryption SDK.

O armazenamento em cache de chaves de dados está disponível para todas as [linguagens de programação](#) compatíveis com AWS Encryption SDK.

Para obter exemplos completos e testados de como usar o armazenamento em cache de chaves de dados no AWS Encryption SDK, consulte:

- C/C++: [caching_cmm.cpp](#)
- Java: [SimpleDataKeyCachingExample.java](#)
- JavaScript Navegador: [caching_cmm.ts](#)
- JavaScript Node.js: [caching_cmm.ts](#)
- Python: [data_key_caching_basic.py](#)

Produtor

O produtor obtém um mapa, converte-o em JSON, usa o AWS Encryption SDK para criptografá-lo e envia o registro do texto cifrado a um [stream do Kinesis](#) em cada Região da AWS.

O código define um [gerenciador de materiais criptográficos de armazenamento em cache](#) (CMM de armazenamento em cache) e o associa a um [cache local](#) e a um [provedor de chave mestrado AWS KMS](#) subjacente. O CMM de armazenamento em cache armazena em cache as chaves de dados (e o [material criptográfico relacionado](#)) do provedor de chaves mestras. Ele também interage com o cache em nome do SDK e impõe os limites de segurança que você define.

Como a chamada para o método de criptografia especifica um CMM de armazenamento em cache, em vez de um [gerenciador de materiais criptográficos \(CMM\)](#) ou provedor de chave mestra comum, a criptografia usará o cache de chave de dados.

Java

O exemplo a seguir usa a versão 2. x do AWS Encryption SDK for Java. Versão 3. x do AWS Encryption SDK for Java desaprova o CMM de armazenamento em cache da chave de dados. Com a versão 3. x, você também pode usar o [AWS KMSchaveiro hierárquico](#), uma solução alternativa de cache de materiais criptográficos.

```
/*
 * 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.kmsdkv2.KmsMasterKey;
import com.amazonaws.encryptionsdk.kmsdkv2.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
        )
```

Consumidor

O consumidor de dados é uma função do [AWS Lambda](#) acionada por eventos do [Kinesis](#). Ele descriptografa e desserializa cada registro e grava o registro de texto simples em uma tabela do [Amazon DynamoDB](#) na mesma região.

Como o código do produtor, o código do consumidor habilita o armazenamento em cache da chave de dados usando um gerenciador de materiais criptográficos de cache (caching CMM) em chamadas para o método de descriptografia.

O código Java cria um provedor de chave mestra no modo estrito com um AWS KMS key especificado. O modo estrito não é necessário ao descriptografar, mas é uma [prática recomendada](#). O código Python usa o modo de descoberta, que permite ao AWS Encryption SDK usar qualquer chave de encapsulamento que criptografe uma chave de dados para descriptografá-la.

Java

O exemplo a seguir usa a versão 2. x do AWS Encryption SDK for Java. Versão 3. x do AWS Encryption SDK for Java desaprova o CMM de armazenamento em cache da chave de dados. Com a versão 3. x, você também pode usar o [AWS KMSchaveiro hierárquico](#), uma solução alternativa de cache de materiais criptográficos.

Esse código cria um provedor de chave mestra para descriptografia no modo estrito. O AWS Encryption SDK pode usar somente o AWS KMS keys especificado por você para descriptografar sua mensagem.

```
/*
 * 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.kmsdkv2.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 UnsupportedOperationException {
    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

Esse código Python é descrito grafado com um provedor de chave mestra no modo de descoberta. Ele permite ao AWS Encryption SDK usar qualquer chave de encapsulamento que criptografe uma chave de dados para descrito grafá-la. O modo estrito, no qual você especifica as chaves de encapsulamento que podem ser usadas para decodificação, é uma [prática recomendada](#).

```
"""
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)
```

Exemplo de armazenamento em cache de chave de dados: modelo AWS CloudFormation

Este modelo do AWS CloudFormation configura todos os atributos da AWS necessários para reproduzir o [exemplo de armazenamento em cache da chave de dados](#).

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": {

```



```

        },
        "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": "*"
    }
  ]

```



```
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
```

Versões do AWS Encryption SDK

As implementações da AWS Encryption SDK linguagem usam [versionamento semântico](#) para facilitar a identificação da magnitude das mudanças em cada versão. Uma alteração no número da versão principal, como de 1.x. x para 2.x.x, indica uma alteração significativa que provavelmente exigirá alterações no código e uma implantação planejada. Alterações significativas em uma nova versão podem não afetar todos os casos de uso. Consulte as notas de lançamento para ver se você foi afetado. Uma alteração em uma versão secundária, como de x.1.x para x.2.x, é sempre compatível com versões anteriores, mas pode incluir elementos descontinuados.

Sempre que possível, use a versão mais recente do AWS Encryption SDK na linguagem de programação escolhida. A [política de manutenção e suporte](#) para cada versão é diferente para cada implementação de linguagem de programação. Para obter detalhes sobre as versões suportadas em sua linguagem de programação preferida, consulte o `SUPPORT_POLICY.rst` arquivo em seu [GitHub repositório](#).

Quando as atualizações incluem novos atributos que exigem configuração especial para evitar erros de criptografia ou descryptografia, fornecemos uma versão intermediária e instruções detalhadas para usá-la. Por exemplo, as versões 1.7.x e 1.8.x foram projetadas para serem versões transitórias que ajudam você a atualizar de versões anteriores à 1.7.x para as versões 2.0.x e posteriores. Para obter detalhes, consulte [Como migrar seu AWS Encryption SDK](#).

Note

O x em um número de versão representa qualquer patch da versão principal e secundária. Por exemplo, a versão 1.7.x representa todas as versões que começam com 1.7, incluindo 1.7.1 e 1.7.9.

Novos recursos de segurança foram lançados originalmente nas versões 1.7 do AWS Encryption CLI. x e 2.0. x. No entanto, a versão AWS 1.8 do Encryption CLI. x substitui a versão 1.7. x e CLI de AWS criptografia 2.1. x substitui 2.0. x. Para obter detalhes, consulte a [consultoria de segurança](#) relevante no [aws-encryption-sdk-cli](#) repositório em GitHub.

As tabelas a seguir fornecem uma visão geral das principais diferenças entre as versões suportadas do AWS Encryption SDK para cada linguagem de programação.

C

Para obter uma descrição detalhada de todas as alterações, consulte o [CHANGELOG.md no repositório](#) em. [aws-encryption-sdk-c](#) GitHub

Versão principal	Detalhes	Fase do ciclo de vida da versão principal do SDK
1.x	1,0	Versão inicial.
	1,7	Atualizações do AWS Encryption SDK que ajudam os usuários de versões anteriores a atualizar para as versões 2.0. x e mais tarde. Para obter mais informações, consulte a versão 1.7. x .
2.x	2,0	Atualizações do AWS Encryption SDK. Para obter mais informações, consulte a versão 2.0. x .
	2,2	Melhorias no processo de decodificação de mensagens.
	2,3	Adiciona suporte para chaves AWS KMS multirregionais.

[Fase de fim do suporte](#)

[Disponibilidade geral \(GA\)](#)

C# /.NET

Para obter uma descrição detalhada de todas as alterações, consulte o [CHANGELOG.md no repositório](#) em. [aws-encryption-sdk-net](#) GitHub

Versão principal	Detalhes		Fase do ciclo de vida da versão principal do SDK
3.x	3.0	Versão inicial.	Disponibilidade geral (GA) A versão 3.x do AWS Encryption SDK para.NET entrará no modo de manutenção o em 13 de maio de 2024.
4.x	4,0	Adiciona suporte ao AWS KMS chaveiro hierárquico, ao contexto de criptografia necessário (CMM) e aos chaveiros RSA assimétricos. AWS KMS	Disponibilidade geral (GA)

Interface de linha de comando (CLI)

Para obter uma descrição detalhada de todas as alterações, consulte [Versões da CLI de criptografia da AWS](#) e o [Changelog.rst](#) no repositório em. [aws-encryption-sdk-cli](#) GitHub

Versão principal	Detalhes		Fase do ciclo de vida da versão principal do SDK

1.x	1,0	Versão inicial.	Fase de fim do suporte
	1,7	Atualizações do AWS Encryption SDK que ajudam os usuários de versões anteriores a atualizar para as versões 2.0. x e mais tarde. Para obter mais informações, consulte a versão 1.7. x .	
2.x	2,0	Atualizações do AWS Encryption SDK. Para obter mais informações, consulte a versão 2.0. x .	Fase de fim do suporte
	2.1	Remove o <code>--discovery</code> parâmetro e o substitui pelo <code>discovery</code> atributo do <code>--wrapping-keys</code> parâmetro. A versão 2.1.0 da CLI de AWS criptografia é equivalente à versão 2.0 em outras linguagens de programação.	
	2.2	Melhorias no processo de decodificação de mensagens.	

3.x	3.0	Adiciona suporte para chaves AWS KMS multirregionais.	Fase de fim do suporte
4.x	4.0	A CLI de AWS criptografia não é mais compatível com Python 2 ou Python 3.4. A partir da versão principal 4. x da CLI de AWS criptografia, somente o Python 3.5 ou posterior é suportado.	Disponibilidade geral (GA)
	4.1	A CLI AWS de criptografia não oferece mais suporte ao Python 3.5. A partir da versão 4.1. x da CLI de AWS criptografia, somente o Python 3.6 ou posterior é suportado.	
	4.2	A CLI AWS de criptografia não oferece mais suporte ao Python 3.6. A partir da versão 4.2. x da CLI de AWS criptografia, somente o Python 3.7 ou posterior é suportado.	

Java

Para obter uma descrição detalhada de todas as alterações, consulte o [Changelog.rst](#) no repositório em. [aws-encryption-sdk-java](#) GitHub

Versão principal	Detalhes	Fase do ciclo de vida da versão principal do SDK	
1.x	1,0	Versão inicial.	
	1.3	Adiciona suporte para gerenciador de materiais criptográficos e armazenamento em cache de chaves de dados. Transferido para a geração IV determinística.	Fase de fim do suporte
	1.6.1	Deprecia <code>AwsCrypto.encryptString()</code> e <code>AwsCrypto.decryptString()</code> e os substitui por <code>AwsCrypto.encryptData()</code> e <code>AwsCrypto.decryptData()</code>	
	1,7	Atualizações do AWS Encryption SDK que ajudam os usuários de versões anteriores a atualizar para as	

		versões 2.0. x e mais tarde. Para obter mais informações, consulte a versão 1.7. x .	
2.x	2,0	Atualizações do AWS Encryption SDK. Para obter mais informações, consulte a versão 2.0. x .	Disponibilidade geral (GA) A versão 2.x do AWS Encryption SDK for Java entrará no modo de manutenção em 2024.
	2.2	Melhorias no processo de decodificação de mensagens.	
	2.3	Adiciona suporte para chaves AWS KMS multirregionais.	
	2.4	Adiciona suporte para AWS SDK for Java 2.x.	

3.x	3.0	<p>Integra-o AWS Encryption SDK for Java com a biblioteca do fornecedor de materiais.</p> <p>Adiciona suporte para chaveiros RSA simétricos e assimétricos, AWS KMS chaveiros AWS KMS hierárquicos, chaveiros AES brutos, chaveiros RSA brutos, chaveiros múltiplos e o contexto de criptografia necessário CMM.</p>	Disponibilidade geral (GA)
-----	-----	--	--

JavaScript

Para obter uma descrição detalhada de todas as alterações, consulte o [CHANGELOG.md no repositório](#) em. [aws-encryption-sdk-javascript](#) GitHub

Versão principal	Detalhes	Fase do ciclo de vida da versão principal do SDK
1.x	<p>1,0</p> <p>1,7</p>	<p>Versão inicial.</p> <p>Atualizações do AWS Encryption SDK que ajudam os usuários de versões anteriores a atualizar para as versões 2.0. x e mais</p>

[Fase de fim do suporte](#)

		tarde. Para obter mais informações, consulte a versão 1.7. x .	
2.x	2,0	Atualizações do AWS Encryption SDK. Para obter mais informações, consulte a versão 2.0. x .	Fase de fim do suporte
	2.2	Melhorias no processo de decodificação de mensagens.	
	2.3	Adiciona suporte para chaves AWS KMS multirregionais.	
3.x	3.0	Remove a cobertura de CI para o Node 10. Atualiza as dependências para que não sejam mais compatíveis com o Node 8 e o Node 10.	Manutenção O suporte para a versão 3.x do AWS Encryption SDK para JavaScript terminará em 17 de janeiro de 2024.
4.x	4,0	Requer a versão 3 AWS Encryption SDK para JavaScript do <code>kms-client</code> para usar o AWS KMS chaveiro.	Disponibilidade geral (GA)

Python

Para obter uma descrição detalhada de todas as alterações, consulte o [Changelog.rst](#) no repositório em. [aws-encryption-sdk-python](#) GitHub

Versão principal	Detalhes	Fase do ciclo de vida da versão principal do SDK
1.x	1,0	Versão inicial.
	1.3	Adiciona suporte para gerenciador de materiais criptográficos e armazenamento em cache de chaves de dados. Transferido para a geração IV determinística.
	1,7	Atualizações do AWS Encryption SDK que ajudam os usuários de versões anteriores a atualizar para as versões 2.0. x e mais tarde. Para obter mais informações, consulte a versão 1.7. x .
2.x	2,0	Atualizações do AWS Encryption SDK. Para obter mais informações, consulte a versão 2.0. x .
	2.2	Melhorias no processo de decodificação de mensagens.

	2.3	Adiciona suporte para chaves AWS KMS multirregionais.	
3.x	3.0	O AWS Encryption SDK for Python não oferece mais suporte ao Python 2 ou ao Python 3.4. A partir da versão principal 3.x do AWS Encryption SDK for Python, somente o Python 3.5 ou posterior é suportado.	Disponibilidade geral (GA)

Detalhes da versão

A lista a seguir descreve as principais diferenças entre as versões suportadas do AWS Encryption SDK.

Tópicos

- [Versões anteriores à 1.7.x](#)
- [Versão 1.7.x](#)
- [Versão 2.0x](#)
- [Versão 2.2x](#)
- [Versão 2.3x](#)

Versões anteriores à 1.7.x

Note

Todos os 1. x. As versões x do AWS Encryption SDK estão em [end-of-supportfase](#). Atualize para a versão mais recente disponível do AWS Encryption SDK para sua linguagem de programação assim que possível. Para atualizar de uma AWS Encryption SDK versão

anterior à 1.7. x, você deve primeiro atualizar para 1.7. x. Para obter detalhes, consulte [Como migrar seu AWS Encryption SDK](#).

Versões AWS Encryption SDK anteriores à 1.7. x fornecem recursos de segurança importantes, incluindo criptografia com o algoritmo Advanced Encryption Standard no Galois/Counter Mode (AES-GCM), uma função de derivação de chave (HKDF) baseada em HMAC, assinatura e uma extract-and-expand chave de criptografia de 256 bits. No entanto, elas não são compatíveis com as [práticas recomendadas](#) por nós, incluindo [confirmação de chave](#).

Versão 1.7.x

Note

Todos os 1. x. As versões x do AWS Encryption SDK estão em [end-of-supportfase](#).

Versão 1.7. x foi projetado para ajudar os usuários de versões anteriores do a atualizar AWS Encryption SDK para as versões 2.0. x e mais tarde. Se você é novo no AWS Encryption SDK, pode pular essa versão e começar com a versão mais recente disponível em sua linguagem de programação.

A versão 1.7.x é totalmente compatível com versões anteriores; ela não introduz nenhuma alteração significativa nem altera o comportamento do AWS Encryption SDK. Também é compatível com versões posteriores. Permite que você atualize seu código para que ele seja compatível com a versão 2.0.x.. Ela inclui novos atributos, mas não os habilita completamente. E requer valores de configuração que evitem que você adote imediatamente todos os novos atributos até que esteja pronto para fazer isso.

A versão 1.7.x inclui as seguintes alterações:

AWS KMS atualizações do provedor de chave mestra (obrigatório)

Versão 1.7. x introduz novos construtores no AWS Encryption SDK for Java e AWS Encryption SDK for Python que criam explicitamente provedores de chaves AWS KMS mestras no modo estrito ou no modo de descoberta. Esta versão adiciona alterações semelhantes à interface de AWS Encryption SDK linha de comando (CLI). Para obter detalhes, consulte [Como atualizar provedores de chaves mestras do AWS KMS](#).

- No modo estrito, os provedores de chaves mestras do AWS KMS exigem uma lista de chaves de encapsulamento e criptografam e descriptografam somente com as chaves de encapsulamento que você especificar. Essa é uma prática recomendada do AWS Encryption SDK que garante que você use as chaves de encapsulamento que pretende usar.
- No modo de descoberta, os provedores de chaves mestras do AWS KMS não aceitam nenhuma chave de encapsulamento. Você não pode usá-los para criptografar. Ao descriptografar, eles podem usar qualquer chave de encapsulamento para descriptografar uma chave de dados criptografada. No entanto, você pode limitar as chaves de encapsulamento usadas para descriptografia àquelas presentes em Contas da AWS específicas. Esse filtro de descoberta é opcional, mas é uma [prática recomendada](#) que incentivamos.

Os construtores que criam versões anteriores dos provedores de chaves AWS KMS mestras estão obsoletos na versão 1.7. x e removido na versão 2.0. x. Esses construtores instanciam provedores de chave mestra que criptografam usando as chaves de encapsulamento que você especificar. No entanto, eles descriptografam chaves de dados criptografadas usando a chave de encapsulamento que as criptografou, independentemente das chaves de encapsulamento especificadas. Os usuários podem decifrar mensagens sem querer com chaves de agrupamento que não pretendem usar, inclusive em outras regiões. AWS KMS keys Contas da AWS

Não há alterações nos construtores das chaves AWS KMS mestras. Ao criptografar e descriptografar, as chaves AWS KMS mestras usam somente o AWS KMS key que você especificar.

AWS KMS atualizações de chaveiros (opcional)

Versão 1.7. x adiciona um novo filtro às AWS Encryption SDK para JavaScript implementações AWS Encryption SDK for C e que limita os [chaveiros de AWS KMS descoberta a determinados](#). Contas da AWS Esse novo filtro de conta é opcional, mas é uma [prática recomendada](#) que apoiamos. Para obter detalhes, consulte [Atualizar tokens de autenticação do AWS KMS](#).

Não há alterações nos construtores dos AWS KMS chaveiros. AWS KMS Os chaveiros padrão se comportam como fornecedores de chaves mestras no modo estrito. AWS KMS os chaveiros de descoberta são criados explicitamente no modo de descoberta.

Passando um ID de chave para AWS KMS Decrypt

A partir da versão 1.7. x, [ao descriptografar chaves de dados criptografadas, o AWS Encryption SDK sempre especifica an AWS KMS key em suas chamadas para a operação Decrypt.](#) [AWS KMS](#) O AWS Encryption SDK obtém o valor do ID da chave a AWS KMS key partir dos

metadados em cada chave de dados criptografada. Esse atributo não requer alterações no código.

[Não AWS KMS key é necessário especificar o ID da chave do para descriptografar o texto cifrado que foi criptografado com uma chave KMS de criptografia simétrica, mas é uma prática recomendada.](#)[AWS KMS](#) Assim como especificar chaves de agrupamento em seu provedor de chaves, essa prática garante que AWS KMS apenas descriptografe usando a chave de agrupamento que você pretende usar.

Descriptografar texto cifrado com confirmação de chave

A versão 1.7x pode descriptografar texto cifrado criptografado com ou sem [confirmação de chave](#). No entanto, ela não pode criptografar texto cifrado com confirmação de chave. Essa propriedade permite que você implante totalmente aplicações que podem descriptografar texto cifrado criptografado com confirmação de chave antes mesmo de encontrem esse tipo de texto cifrado. Como essa versão descriptografa mensagens que são criptografadas sem confirmação de chave, você não precisa recriptografar nenhum texto cifrado.

Para implementar esse comportamento, versão 1.7. x inclui uma nova configuração de [política de compromisso](#) que determina se eles AWS Encryption SDK podem criptografar ou descriptografar com compromisso de chave. Na versão 1.7. x, o único valor válido para a política de compromisso, `ForbidEncryptAllowDecrypt`, é usado em todas as operações de criptografia e descriptografia. Esse valor impede que o AWS Encryption SDK criptografe com qualquer um dos novos pacotes de algoritmos que incluem confirmação de chave. Ele permite AWS Encryption SDK decifrar texto cifrado com e sem compromisso de chave.

Embora haja apenas um valor de política de compromisso válido na versão 1.7.x, exigimos que você defina esse valor explicitamente ao usar as novas APIs introduzidas nesta versão. Definir o valor explicitamente impede que sua política de compromisso seja alterada automaticamente para `require-encrypt-require-decrypt` quando você atualizar para a versão 2.1.x. Em vez disso, você pode [migrar sua política de compromisso](#) em etapas.

Pacotes de algoritmos com confirmação de chave fundamental

Versão 1.7. x inclui dois novos [pacotes de algoritmos](#) compatíveis com o confirmação de chaves. Um inclui assinatura; o outro, não. Como os pacotes de algoritmos suportados anteriormente, esses dois novos pacotes de algoritmos incluem criptografia com AES-GCM, uma chave de criptografia de 256 bits e uma função de derivação de chave baseada em HMAC (`extract-and-expand HKDF`).

No entanto, o conjunto de algoritmos padrão usado para criptografia não muda. Esses pacotes de algoritmos foram adicionados à versão 1.7.x para preparar a aplicação para usá-los nas versões 2.0.x e posteriores.

Alterações na implementação do CMM

A versão 1.7.x introduz mudanças na interface Default do gerenciador de materiais criptográficos (CMM) para dar suporte ao o comprometimento chave. Essa alteração lhe afeta somente se você tiver escrito um CMM personalizado. Para obter detalhes, consulte a documentação da API ou o GitHub repositório da sua [linguagem de programação](#).

Versão 2.0x

Versão 2.0. x oferece suporte aos novos recursos de segurança oferecidos no AWS Encryption SDK, incluindo chaves de empacotamento especificadas e comprometimento de chaves. A versão 2.0.x inclui alterações significativas em relação a todas as versões anteriores do AWS Encryption SDK. Você pode se preparar para essas mudanças implantando a versão 1.7.x.. A versão 2.0.x inclui todos os novos atributos introduzidos na versão 1.7.x com as seguintes adições e alterações.

Note

Versão 2. x. x [do AWS Encryption SDK for Python, AWS Encryption SDK para JavaScript, e a CLI de AWS criptografia](#) estão em fase. end-of-support

Para obter informações sobre [suporte e manutenção](#) dessa AWS Encryption SDK versão em sua linguagem de programação preferida, consulte o SUPPORT_POLICY.rst arquivo em seu [GitHubrepositório](#).

AWS KMS fornecedores de chaves mestras

Os construtores originais do provedor de chave AWS KMS mestra que foram descontinuados na versão 1.7. x são removidos na versão 2.0. x. Você deve criar explicitamente provedores de chaves mestras do AWS KMS no [modo estrito ou no modo de descoberta](#).

Criptografe e descriptografe texto cifrado com confirmação de chave

A versão 2.0.x pode descriptografar texto cifrado criptografado com ou sem [confirmação de chave](#). Seu comportamento é determinado pela definição da política de compromisso. Por padrão, ela sempre criptografa com confirmação de chave e só descriptografa texto cifrado

criptografado com confirmação de chave. A menos que você altere a política de compromisso, o AWS Encryption SDK não descriptografa textos cifrados criptografados por nenhuma versão anterior do AWS Encryption SDK, incluindo a versão 1.7.x..

Important

Por padrão, a versão 2.0.x não descriptografa nenhum texto cifrado que tenha sido criptografado sem a confirmação de chave. Se a aplicação encontrar um texto cifrado criptografado sem confirmação de chave, defina um valor de política de compromisso como `AllowDecrypt`.

Na versão 2.0.x, a configuração da política de compromisso tem três valores válidos:

- `ForbidEncryptAllowDecrypt`: o AWS Encryption SDK não pode criptografar com confirmação de chave. Ele pode descriptografar textos cifrados criptografados com ou sem confirmação de chave.
- `RequireEncryptAllowDecrypt`: o AWS Encryption SDK deve criptografar com confirmação de chave. Ele pode descriptografar textos cifrados criptografados com ou sem confirmação de chave.
- `RequireEncryptRequireDecrypt`(padrão) — AWS Encryption SDK É necessário criptografar com compromisso de chave. Ele só descriptografa textos cifrados com confirmação de chave.

Se você estiver migrando de uma versão anterior do AWS Encryption SDK para a versão 2.0. x, defina a política de compromisso com um valor que garanta que você possa descriptografar todos os textos cifrados existentes que seu aplicativo possa encontrar. É provável que você ajuste essa configuração com o tempo.

Versão 2.2x

Adiciona suporte para assinaturas digitais e limita as chaves de dados criptografadas.

Note

Versão 2. x. x [do AWS Encryption SDK for Python, AWS Encryption SDK para JavaScript, e a CLI de AWS criptografia estão em fase. end-of-support](#)

Para obter informações sobre [suporte e manutenção](#) dessa AWS Encryption SDK versão em sua linguagem de programação preferida, consulte o `SUPPORT_POLICY.txt` arquivo em seu [GitHub repositório](#).

Assinaturas digitais

Para melhorar o manuseio de [assinaturas digitais](#) durante a decodificação, isso AWS Encryption SDK inclui os seguintes recursos:

- Modo sem streaming: retorna texto simples somente após o processamento de todas as entradas, incluindo a verificação da assinatura digital, se houver uma. Esse atributo impede que você use texto simples antes de verificar a assinatura digital. Use-o sempre que descriptografar dados criptografados com assinaturas digitais (o pacote de algoritmos padrão). Por exemplo, como a CLI de AWS criptografia sempre processa dados no modo de streaming, use o `--buffer` parâmetro ao descriptografar texto cifrado com assinaturas digitais.
- Modo de descriptografia somente não assinada: esse atributo só descriptografa texto cifrado não assinado. Se a descriptografia encontrar uma assinatura digital no texto cifrado, a operação falhará. Use esse atributo para evitar o processamento não intencional de texto simples de mensagens assinadas antes de verificar a assinatura.

Limitar as chaves de dados criptografadas

Você pode [limitar o número de chaves de dados criptografadas](#) em uma mensagem criptografada. Esse atributo pode ajudar você a detectar um provedor de chave mestra ou um token de autenticação mal configurado ao criptografar ou a identificar um texto cifrado malicioso ao descriptografar.

Você deve limitar as chaves de dados criptografadas ao descriptografar mensagens de uma fonte não confiável. Isso evita chamadas desnecessárias, caras e potencialmente exaustivas para sua infraestrutura principal.

Versão 2.3x

Adiciona suporte para chaves AWS KMS multirregionais. Para obter detalhes, consulte [Usar AWS KMS keys multirregionais](#).

Note

A CLI AWS de criptografia oferece suporte a chaves multirregionais a partir da versão 3.0. x.

Versão 2. x. x [do AWS Encryption SDK for Python, AWS Encryption SDK para JavaScript, e a CLI de AWS criptografia estão em fase. end-of-support](#)

Para obter informações sobre [suporte e manutenção](#) dessa AWS Encryption SDK versão em sua linguagem de programação preferida, consulte o `SUPPORT_POLICY.rst` arquivo em seu [GitHub repositório](#).

Como migrar seu AWS Encryption SDK

O AWS Encryption SDK é compatível com várias [implementações de linguagem de programação](#) interoperáveis, cada uma delas desenvolvida em um repositório de código aberto no GitHub. Como [prática recomendada](#), use a versão mais recente do AWS Encryption SDK para cada linguagem.

Você pode atualizar com segurança a partir da versão 2.0.x ou posterior até a versão mais recente do AWS Encryption SDK. No entanto, a versão 2.0x do AWS Encryption SDK introduz novos atributos de segurança significativos, alguns dos quais são mudanças significativas. Para atualizar de versões anteriores à 1.7.x para a versão 2.0.x e posteriores, primeiro será necessário atualizar para a versão 1.x mais recente. Os tópicos desta seção foram elaborados para ajudar você a entender as alterações, selecionar a versão correta para a aplicação e migrar com segurança e sucesso para as versões mais recentes do AWS Encryption SDK.

Para obter informações sobre versões significativas do AWS Encryption SDK, consulte [Versões do AWS Encryption SDK](#).

Important

Não atualize diretamente de uma versão anterior à 1.7x para a versão 2.0.x ou posterior sem primeiro atualizar para a mais recente versão 1x.. Se você atualizar diretamente para a versão 2.0.x ou posterior e ativar todos os novos atributos imediatamente, o AWS Encryption SDK não conseguirá descriptografar texto cifrado criptografado em versões mais antigas do AWS Encryption SDK.

Note

A versão mais antiga do AWS Encryption SDK para .NET é a versão 3.0.x. Todas as versões do AWS Encryption SDK para .NET são compatíveis com as práticas recomendadas de segurança introduzidas na versão 2.0. x do AWS Encryption SDK. É possível atualizar com segurança para a versão mais recente sem fazer alterações no código ou nos dados.

CLI de criptografia da AWS: ao ler este guia de migração, use as instruções de migração 1.7.x para a CLI de criptografia da AWS 1.8.x e use as instruções de migração 2.0.x para a CLI de criptografia da AWS 2.1.x.. Para obter mais detalhes, consulte [Versões da CLI de criptografia da AWS](#).

Os novos atributos de segurança foram lançados originalmente nas versões 1.7.x e 2.0.x da CLI de criptografia da AWS. No entanto, a versão 1.8x da CLI de criptografia da AWS substituiu a versão 1.7.x e a versão 2.1.x da CLI de criptografia da AWS substituiu a 2.0.x. Para obter detalhes, consulte o [boletim de segurança](#) pertinente no repositório [aws-encryption-sdk-cli](#) no GitHub.

Novos usuários

Se você é novo no AWS Encryption SDK, instale a versão mais recente do AWS Encryption SDK para sua linguagem de programação. Os valores padrão habilitam todos os atributos de segurança do AWS Encryption SDK, incluindo criptografia com assinatura, derivação de chave e [confirmação de chave](#) do AWS Encryption SDK

Usuários atuais

Recomendamos atualizar da versão atual para a versão mais recente disponível assim que possível. Todas as versões 1.x do AWS Encryption SDK estão na [fase de fim do suporte](#), assim como versões posteriores em algumas linguagens de programação. Para obter detalhes sobre o status de suporte e manutenção do AWS Encryption SDK em sua linguagem de programação, consulte [Suporte e manutenção](#).

Versões 2.0.x e posteriores do AWS Encryption SDK fornecem novos atributos de segurança para ajudar a proteger seus dados. No entanto, a versão 2.0.x do AWS Encryption SDK inclui alterações significativas que não são compatíveis com versões anteriores. Para garantir uma transição segura, comece migrando da sua versão atual para a mais recente 1.x na sua linguagem de programação. Quando a versão 1.x estiver totalmente implantada e operando com sucesso, você poderá migrar com segurança para as versões 2.0.x e posteriores. Esse [processo de duas etapas](#) é essencial, especialmente para aplicações distribuídas.

Para obter mais informações sobre os atributos de segurança do AWS Encryption SDK subjacentes a essas mudanças, consulte [Criptografia do lado do cliente aprimorada: IDs de chave explícitos e confirmação de chave](#) no Blog de segurança da AWS.

Procurando ajuda para usar o AWS Encryption SDK for Java com o AWS SDK for Java 2.x? Consulte [Pré-requisitos](#).

Tópicos

- [Como migrar e implantar o AWS Encryption SDK](#)

- [Como atualizar provedores de chaves mestras do AWS KMS](#)
- [Atualizar tokens de autenticação do AWS KMS](#)
- [Como definir sua política de compromisso](#)
- [Solução de problemas de migração para as versões mais recentes](#)

Como migrar e implantar o AWS Encryption SDK

Ao migrar de uma versão do AWS Encryption SDK anterior à 1.7.x para a versão 2.0.x ou posterior, você deve fazer uma transição segura para a criptografia com [confirmação de chave](#). Caso contrário, a aplicação encontrará textos cifrados que não poderá descriptografar. Se você estiver usando provedores de chave mestra do AWS KMS, deverá atualizar para novos construtores que criam provedores de chave mestra no modo estrito ou no modo de descoberta.

Note

Este tópico foi desenvolvido para usuários que estão migrando de versões anteriores do AWS Encryption SDK para a versão 2.0. x posterior. Se você é novo no AWS Encryption SDK, pode começar a usar a versão mais recente disponível com as configurações padrão imediatamente.

Para evitar uma situação crítica na qual você não possa descriptografar o texto cifrado que precisa ler, recomendamos que você migre e implante em várias etapas distintas. Verifique se cada etapa está completa e totalmente implantada antes de iniciar a próxima etapa. Isso é particularmente importante para aplicações distribuídas com vários hosts.

Etapa 1: atualize a aplicação para a versão 1.x mais recente

Atualize para a versão 1.x mais recente para sua linguagem de programação. Teste com cuidado, implante suas alterações e confirme se a atualização foi propagada para todos os hosts de destino antes de iniciar a etapa 2.

Important

Verifique se a sua versão 1.x mais recente é a versão 1.7.x ou versão posterior do AWS Encryption SDK.

As versões 1.x mais recentes do AWS Encryption SDK são compatíveis com versões anteriores do AWS Encryption SDK e versões anteriores e compatíveis com as versões 2.0.x e posteriores. Elas incluem os novos atributos presentes na versão 2.0.x, mas inclui padrões seguros projetados para essa migração. Esses padrões permitem que você atualize seus provedores de chave mestra do AWS KMS se necessário, e implante totalmente pacotes de algoritmos que podem descriptografar texto cifrado com confirmação de chave.

- Substitua elementos descontinuados, incluindo construtores para provedores de chaves mestras do AWS KMS herdados. Em Python, ative os [avisos de descontinuidade](#). Elementos de código que foram descontinuados na mais recente versão 1.x foram removidos das versões 2.0.x e posteriores.
- Defina explicitamente sua política de compromisso como `ForbidEncryptAllowDecrypt`. Embora esse seja o único valor válido nas versões 1.x mais recentes, essa configuração é necessária quando você usa as APIs introduzidas nesta versão. Isso impede que a aplicação rejeite texto cifrado criptografado sem confirmação de chave quando você migra para a versão 2.0.x e versões posteriores. Para obter mais detalhes, consulte [the section called “Como definir sua política de compromisso”](#).
- Se você usa provedores de chave mestra do AWS KMS, deve atualizar seus provedores de chave mestra herdados para provedores de chave mestra que ofereçam suporte ao modo estrito e ao modo de descoberta. Essa atualização é necessária para o AWS Encryption SDK for Java, o AWS Encryption SDK for Python, e a CLI de criptografia da AWS. Se você usa provedores de chave mestra no modo de descoberta, recomendamos que implemente o filtro de descoberta que limita as chaves de encapsulamento usadas àquelas presentes em Contas da AWS. Essa atualização é opcional, mas é uma [prática recomendada](#) que incentivamos. Para obter mais detalhes, consulte [Como atualizar provedores de chaves mestras do AWS KMS](#).
- Se você usa [token de autenticação de descoberta do AWS KMS](#), recomendamos que inclua um filtro de descoberta que limite as chaves de encapsulamento usadas na descriptografia a aquelas em particular. Contas da AWS Essa atualização é opcional, mas é uma [prática recomendada](#) que incentivamos. Para obter mais detalhes, consulte [Atualizar tokens de autenticação do AWS KMS](#).

Etapa 2: atualize a aplicação para a versão mais recente

Depois que a implantação da mais recente versão 1.x for bem-sucedida em todos os hosts, você pode atualizar para as versões 2.0.x e posteriores. A versão 2.0.x inclui alterações significativas em relação a todas as versões anteriores do AWS Encryption SDK. No entanto, se você fizer as

alterações de código recomendadas na etapa 1, poderá evitar erros ao migrar para a versão mais recente.

Antes de atualizar para a versão mais recente, verifique se sua política de compromisso está consistentemente definida como `ForbidEncryptAllowDecrypt`. Em seguida, dependendo da configuração de dados, você pode migrar no seu próprio ritmo para `RequireEncryptAllowDecrypt` e depois para a configuração padrão, `RequireEncryptRequireDecrypt`. Recomendamos uma série de etapas de transição, como o padrão a seguir.

1. Comece com sua [política de compromisso](#) definida como `ForbidEncryptAllowDecrypt`. O AWS Encryption SDK pode descriptografar mensagens com confirmação de chave, mas ainda não descriptografa com confirmação de chave.
2. Quando estiver pronto, atualize a política de compromisso para `RequireEncryptAllowDecrypt`. O AWS Encryption SDK começará a criptografar seus dados com uma [confirmação de chave](#). Ele poderá descriptografar textos cifrados criptografados com ou sem confirmação de chave.

Antes de atualizar sua política de compromisso para `RequireEncryptAllowDecrypt`, verifique se sua versão 1x mais recente foi implantada em todos os hosts, incluindo os hosts de qualquer aplicação que decodifique o texto cifrado que você produz. Versões do AWS Encryption SDK anteriores à versão 1.7.x não podem descriptografar texto cifrado criptografado com confirmação de chave.

Esse também é um bom momento para adicionar métricas à sua aplicação para medir se você ainda está processando texto cifrado sem confirmação de chave. Isso ajudará você a determinar quando é seguro atualizar sua configuração de política de compromisso para `RequireEncryptRequireDecrypt`. Para algumas aplicações, como aquelas que criptografam mensagens em uma fila do Amazon SQS, isso pode significar esperar tempo suficiente para que todo o texto cifrado criptografado nas versões antigas seja recriptografado ou excluído. Para outras aplicações, como objetos criptografados do S3, talvez seja necessário baixar, recriptografar e recarregar todos os objetos.

3. Quando tiver certeza de que não tem nenhuma mensagem criptografada sem confirmação de chave, você pode atualizar sua política de compromisso para `RequireEncryptRequireDecrypt`. Esse valor garante que seus dados sejam sempre criptografados e descriptografados com o confirmação de chave. Essa configuração é a padrão, então você não precisa defini-la explicitamente, mas recomendamos que faça isso. Uma

configuração explícita [ajudará na depuração](#) e em quaisquer possíveis reversões que possam ser necessárias se a aplicação encontrar texto cifrado criptografado sem confirmação de chave.

Como atualizar provedores de chaves mestras do AWS KMS

Para migrar para a versão 1.x mais recente do AWS Encryption SDK, e, em seguida, para a versão 2.0.x ou posteriores, você deve substituir os provedores de chave mestra herdados do AWS KMS por provedores de chave mestra criados explicitamente no [modo estrito ou no modo de descoberta](#). Os provedores de chave mestra herdados foram descontinuados na versão 1.7.x e foram removidos na versão 2.0. x. Essa alteração é necessária para aplicações e scripts que usam o [AWS Encryption SDK for Java](#), o [AWS Encryption SDK for Python](#) e a [CLI de criptografia da AWS](#). Os exemplos nesta seção mostrarão como atualizar seu código.

Note

Em Python, [ative os avisos de obsolescência](#). Isso ajudará você a identificar as partes do código que precisa atualizar.

Se estiver usando uma chave mestra do AWS KMS (que não seja um provedor de chave mestra), você pode ignorar esta etapa. As chaves mestras do AWS KMS não estão obsoletas nem foram removidas. Elas criptografam e descriptografam somente com as chaves de encapsulamento que você especificar.

Os exemplos nesta seção se concentram nos elementos do seu código precisam ser alterados. [Para ver um exemplo completo do código atualizado, consulte a seção Exemplos do repositório do GitHub para sua linguagem de programação](#). Além disso, esses exemplos normalmente usam ARNs de chave para representar AWS KMS keys. Ao criar um provedor de chave mestra para criptografia, você pode usar qualquer [identificador de chave](#) do AWS KMS válido para representar uma AWS KMS key. Ao criar um provedor de chave mestra para descriptografia, você deve usar um ARN de chave.

Saiba mais sobre migração

Todos os usuários do AWS Encryption SDK podem saber como definir sua política de compromisso em [the section called “Como definir sua política de compromisso”](#).

Usuários do AWS Encryption SDK for C e do AWS Encryption SDK para JavaScript podem saber mais sobre uma atualização opcional dos token de autenticação em [Atualizar tokens de autenticação do AWS KMS](#).

Tópicos

- [Migração para o modo estrito](#)
- [Migrar para o modo de descoberta](#)

Migração para o modo estrito

Depois de atualizar para a versão 1.x do AWS Encryption SDK, substitua seus provedores de chave mestra herdados por provedores de chave mestra no modo estrito. No modo estrito, você deve especificar as chaves de encapsulamento a serem usadas ao criptografar e descriptografar. O AWS Encryption SDK usa somente as chaves de encapsulamento que você especificar. Provedores de chaves mestras obsoletas podem descriptografar dados usando qualquer AWS KMS key que criptografe uma chave de dados, inclusive AWS KMS keys em diferentes Contas da AWS e regiões.

Os provedores de chaves mestras no modo estrito foram introduzidos na versão 1.7.x do AWS Encryption SDK. Eles substituem os provedores de chaves mestras herdados, que foram suspensos na versão 1.7.x e removidos na versão 2.0.x.. Usar provedores de chaves mestras no modo estrito é uma [prática recomendada do AWS Encryption SDK](#).

O código a seguir cria um provedor de chave mestra no modo estrito que você pode usar para criptografar e descriptografar.

Java

Este exemplo representa o código em uma aplicação que usa a versão 1.6.2 ou anterior do AWS Encryption SDK for Java.

Esse código usa o método `KmsMasterKeyProvider.builder()` para instanciar um provedor de chave mestra do AWS KMS que usa uma AWS KMS key como chave de encapsulamento.

```
// 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();
```

Este exemplo representa o código em uma aplicação que usa a versão 1.7.x ou versões posteriores do AWS Encryption SDK for Java. Para ver um exemplo completo, consulte [BasicEncryptionExample.java](#).

Os métodos `Builder.build()` e `Builder.withKeysForEncryption()` usados no exemplo anterior foram suspensos na versão 1.7.x e removidos da versão 2.0.x..

Para atualizar para um provedor de chave mestra de modo estrito, esse código substitui as chamadas para métodos suspensos por uma chamada para o novo método `Builder.buildStrict()`. Este exemplo especifica uma AWS KMS key como a chave de encapsulamento, mas o método `Builder.buildStrict()` pode usar uma lista de várias AWS KMS keys.

```
// Create a master key provider in strict mode  
// Replace the example key ARN with a valid one from your Conta da AWS.  
String awsKmsKey = "arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";  
  
KmsMasterKeyProvider masterKeyProvider = KmsMasterKeyProvider.builder()  
    .buildStrict(awsKmsKey);
```

Python

Este exemplo representa o código em uma aplicação que usa a versão 1.4.1 do AWS Encryption SDK for Python. Esse código usa `KMSMasterKeyProvider`, que foi suspenso na versão 1.7. x e removido da versão 2.0.x.. Ao descriptografar, ele usa qualquer AWS KMS key que criptografe uma chave de dados sem levar em conta a AWS KMS keys que você especificar.

Observe que `KMSMasterKey` não foi suspenso nem removido. Ao criptografar e descriptografar, ele usa somente a AWS KMS key que você especificar.

```
# 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]
)
```

Este exemplo representa o código em uma aplicação que usa a versão 1.7.x do AWS Encryption SDK for Python. Para ver um exemplo completo, consulte [basic_encryption.py](#).

Para atualizar para um provedor de chave mestra de modo estrito, esse código substitui a chamada para `KMSMasterKeyProvider()` com uma chamada para `StrictAwsKmsMasterKeyProvider()`.

```
# Create a master key provider in strict mode
# Replace the example key ARNs with valid values from your Conta da 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

Este exemplo mostra como criptografar e descriptografar usando a versão 1.1.7 ou versões anteriores da CLI de criptografia AWS.

Na versão 1.1.7 e anteriores, ao criptografar, você especifica uma ou mais chaves mestras (ou chaves de encapsulamento), como uma AWS KMS key. Ao descriptografar, você não pode especificar nenhuma chave de encapsulamento, a menos que esteja usando um provedor de chave mestra personalizado. A CLI de criptografia da AWS pode usar qualquer chave de encapsulamento que criptografe uma chave de dados.

```
\\ 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 .

```

Este exemplo mostra como criptografar e descriptografar usando a versão 1.7.x ou versões posteriores da CLI de criptografia da AWS. Para obter exemplos completos, consulte [Exemplos da CLI de criptografia da AWS](#).

O parâmetro `--master-keys` foi suspenso na versão 1.7.x e removido na versão 2.0.x.. Ele foi substituído pelo parâmetro `--wrapping-keys`, que é exigido nos comandos de `encrypt` e `decrypt`. Esse parâmetro é compatível com o modo estrito e o modo de descoberta. O modo estrito é uma prática recomendada do AWS Encryption SDK que garante que você use a chave de encapsulamento que pretende utilizar.

Para atualizar para o modo estrito, use o atributo `key` do parâmetro `--wrapping-keys` para especificar uma chave de encapsulamento ao criptografar e descriptografar.

```

\\ 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 .

```

Migrar para o modo de descoberta

A partir da versão 1.7.x, é uma [prática recomendada do AWS Encryption SDK](#) usar o modo estrito para provedores de chaves mestras do AWS KMS, ou seja, especificar chaves de encapsulamento ao criptografar e descriptografar. Você deve sempre especificar as chaves de encapsulamento ao criptografar. Mas há situações em que não é viável especificar os ARNs de chave das AWS KMS keys para descriptografar. Por exemplo, se você estiver usando aliases para identificar as AWS KMS keys ao criptografar, perderá o benefício dos aliases se precisar listar os ARNs de chave ao descriptografar. Além disso, como os provedores de chave mestra no modo de descoberta se comportam como os provedores de chave mestra originais, você pode usá-los temporariamente como parte de sua estratégia de migração e, posteriormente, atualizar para provedores de chave mestra no modo estrito.

Em casos como esse, você pode usar provedores de chaves mestras no modo de descoberta. Esses provedores de chaves mestras não permitem que você especifique chaves de encapsulamento, portanto, você não pode usá-los para criptografar. Ao descriptografar, eles podem usar qualquer chave de encapsulamento que criptografe uma chave de dados. Mas, diferentemente dos provedores de chaves mestras herdados, que se comportam da mesma maneira, você cria esses provedores explicitamente no modo de descoberta. Ao usar provedores de chave mestra no modo de descoberta, você pode limitar as chaves de encapsulamento que podem ser usadas para aquelas que estão presentes em Contas da AWS específicas. Esse filtro de descoberta é opcional, mas é uma prática recomendada que incentivamos. Para obter informações sobre partições e contas da AWS, consulte [Nomes do atributo da Amazon](#) no Referência geral da AWS.

Os exemplos a seguir criam um provedor de chave mestra do AWS KMS no modo estrito para criptografar e um provedor de chave mestra do AWS KMS no modo de descoberta para descriptografar. O provedor da chave mestra no modo de descoberta usa um filtro de descoberta para limitar as chaves de encapsulamento usadas para descriptografar à partição aws e ao exemplo específico de Contas da AWS. Embora o filtro de conta não seja necessário neste exemplo bastante simples, é uma prática recomendada muito benéfica quando uma aplicação criptografa os dados e outra diferente os descriptografa.

Java

Este exemplo representa o código em uma aplicação que usa a versão 1.7.x ou versões posteriores do AWS Encryption SDK for Java. Para obter um exemplo completo, consulte:

[DiscoveryDecryptionExample.java](#).

Para instanciar um provedor de chave mestra no modo estrito para criptografar, este exemplo usa o método `Builder.buildStrict()`. Para instanciar um provedor de chave mestra no modo de descoberta para descriptografar ele usa o método `Builder.buildDiscovery()`. O método `Builder.buildDiscovery()` usa um `DiscoveryFilter` que limita o AWS Encryption SDK às AWS KMS keys presentes na partição nas contas da AWS especificadas.

```
// Create a master key provider in strict mode for encrypting
// Replace the example alias ARN with a valid one from your Conta da 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

Este exemplo representa o código em uma aplicação que usa a versão 1.7.x ou versões posteriores do AWS Encryption SDK for Python. Para obter um exemplo completo, consulte: [discovery_kms_provider.py](#).

Para criar um provedor de chave mestra no modo estrito para criptografar, este exemplo usa o método `StrictAwsKmsMasterKeyProvider`. Para criar um provedor de chave mestra no modo de descoberta para descriptografar, ele usa `DiscoveryAwsKmsMasterKeyProvider` com um `DiscoveryFilter` que limita o AWS Encryption SDK às AWS KMS keys presentes na partição e nas contas da AWS especificadas.

```
# Create a master key provider in strict mode
# Replace the example key ARN and alias ARNs with valid values from your Conta da
  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

Este exemplo mostra como criptografar e descriptografar usando a versão 1.7.x ou versões posteriores da CLI de criptografia da AWS. A partir da versão 1.7.x, o parâmetro `--wrapping-keys` passou a ser necessário ao criptografar e descriptografar. O parâmetro `--wrapping-keys` é compatível com o modo estrito e o modo de descoberta. Para obter exemplos completos, consulte [the section called “Exemplos”](#).

Ao criptografar, este exemplo especifica uma chave de encapsulamento, que é obrigatória. Ao descriptografar, ele escolhe explicitamente o modo de descoberta usando o atributo `discovery` do parâmetro `--wrapping-keys` com um valor definido como `true`.

Para limitar as chaves de encapsulamento que podem ser usadas pelo AWS Encryption SDK no modo de descoberta àquelas presentes em Contas da AWS específicas, este exemplo usa os atributos `discovery-partition` e `discovery-account` do parâmetro `--wrapping-keys`. Esses atributos opcionais são válidos somente quando o atributo `discovery` for definido como `true`. Você deve usar os atributos `discovery-partition` e `discovery-account` juntos. Nenhum deles é válido sozinho.

```

\\ 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 .
```

Atualizar tokens de autenticação do AWS KMS

Os tokens de autenticação do AWS KMS no [AWS Encryption SDK for C](#), [AWS Encryption SDK para .NET](#) e [AWS Encryption SDK para JavaScript](#) oferecem suporte às [práticas recomendadas](#), permitindo que você especifique chaves de encapsulamento ao criptografar e descriptografar. Se você criar um [token de autenticação de descoberta do AWS KMS](#), você o fará de maneira explícita.

Note

A versão mais antiga do AWS Encryption SDK para .NET é a versão 3.0.x. Todas as versões do AWS Encryption SDK para .NET são compatíveis com as práticas recomendadas de segurança introduzidas na versão 2.0. x do AWS Encryption SDK. É possível atualizar com segurança para a versão mais recente sem fazer alterações no código ou nos dados.

Ao atualizar para a versão 1.x mais recente do AWS Encryption SDK, poderá usar um [filtro de descoberta](#) para limitar as chaves de encapsulamento usadas por um [token de autenticação de descoberta do AWS KMS](#) ou um [token de autenticação de descoberta regional do AWS KMS](#) ao descriptografar para aquelas em Contas da AWS específicas. Filtrar um token de autenticação de descoberta é uma [prática recomendada](#) do AWS Encryption SDK.

Os exemplos nesta seção mostram como adicionar o filtro de descoberta a um token de autenticação de descoberta regional do AWS KMS.

Saiba mais sobre migração

Todos os usuários do AWS Encryption SDK podem saber como definir sua política de compromisso em [the section called “Como definir sua política de compromisso”](#).

Para AWS Encryption SDK for Java, AWS Encryption SDK for Python, e a CLI de criptografia da AWS, conheça a atualização obrigatória nos provedores de chaves mestras em [the section called “Como atualizar provedores de chaves mestras do AWS KMS”](#).

Você pode ter um código como o seguinte na aplicação. Este exemplo cria um token de autenticação de descoberta regional do AWS KMS que só pode usar chaves de encapsulamento na região Oeste dos EUA (Oregon) (us-west-2). Este exemplo representa o código em versões do AWS Encryption SDK anteriores à 1.7.x.. No entanto, ele ainda é válido nas versões 1.7x e posteriores.

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 })
```

A partir da versão 1.7. x, é possível adicionar um filtro de descoberta a qualquer token de autenticação de descoberta do AWS KMS. Esse filtro de descoberta limita as AWS KMS keys que o AWS Encryption SDK pode usar para descriptografia àquelas presentes na partição e nas contas especificadas. Antes de usar esse código, altere a partição, se necessário, e substitua as IDs de conta de exemplo por outras válidas.

C

Para obter um exemplo completo, consulte: [kms_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

Para obter um exemplo completo, consulte: [kms_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' }
})
```

Como definir sua política de compromisso

O [confirmação de chave](#) assegura que seus dados criptografados sempre sejam descriptografados para o mesmo texto simples. Para fornecer essa propriedade de segurança, a partir da versão 1.7.x, o AWS Encryption SDK passou a usar novos [pacotes de algoritmos](#) com confirmação de chave. Para determinar se seus dados são criptografados e descriptografados com confirmação de chave, use a definição de configuração da [política de compromisso](#). Criptografar e descriptografar dados com [confirmação de chave](#) é uma prática recomendada do AWS Encryption SDK.

Definir uma política de compromisso é parte importante da segunda etapa do processo de migração: migrar da últimas versões 1.x do AWS Encryption SDK para as versões 2.0.x e posteriores. Após definir e alterar sua política de compromisso, certifique-se de testar a aplicação minuciosamente antes de implantá-la em produção. Para obter orientação sobre migração, consulte [Como migrar e implantar o AWS Encryption SDK](#).

A configuração da política de compromisso tem três valores válidos nas versões 2.0. x posteriores. Nas versões 1.x mais recentes (a partir da versão 1.7.x), somente `ForbidEncryptAllowDecrypt` é válido.

- `ForbidEncryptAllowDecrypt`: o AWS Encryption SDK não pode criptografar com confirmação de chave. Ele pode descriptografar textos cifrados criptografados com ou sem confirmação de chave.

Na mais recente versão 1.x, esse é o único valor válido. Isso garante que você não criptografe com confirmação de chave até que esteja totalmente preparado para descriptografar com confirmação de chave. Definir o valor explicitamente impede que sua política de compromisso seja alterada automaticamente para `require-encrypt-require-decrypt` quando você atualizar para as versões 2.0.x ou posteriores. Em vez disso, você pode [migrar sua política de compromisso](#) em etapas.

- `RequireEncryptAllowDecrypt`: o AWS Encryption SDK sempre criptografa com confirmação de chave. Ele pode descriptografar textos cifrados criptografados com ou sem confirmação de chave. Esse valor foi adicionado na versão 2.0.x..
- `RequireEncryptRequireDecrypt`: o AWS Encryption SDK sempre criptografa e descriptografa com confirmação de chave. Esse valor foi adicionado na versão 2.0.x.. É o valor padrão em versões 2.0.x. e posteriores.

Na versão 1.x mais recente, o único valor de política de compromisso válido é `ForbidEncryptAllowDecrypt`. Depois de migrar para a versão 2.0.x ou posterior, você pode [alterar sua política de compromisso em etapas](#) conforme estiver pronto. Não atualize sua política de compromisso `RequireEncryptRequireDecrypt` até ter certeza de que não tem nenhuma mensagem criptografada sem o confirmação de chave.

Esses exemplos mostram como definir sua política de compromisso na última versão 1.x e nas versões 2.0.x posteriores. A técnica depende da sua linguagem de programação.

Saiba mais sobre migração

Para AWS Encryption SDK for Java, AWS Encryption SDK for Python, e a CLI de criptografia da AWS, saiba mais sobre as mudanças necessárias nos provedores de chaves mestras em [the section called “Como atualizar provedores de chaves mestras do AWS KMS”](#).

Para AWS Encryption SDK for C e AWS Encryption SDK para JavaScript, saiba mais sobre uma atualização opcional dos tokens de autenticação em [Atualizar tokens de autenticação do AWS KMS](#).

Como definir sua política de compromisso

A técnica que você usa para definir sua política de compromisso difere um pouco em cada implementação de linguagem. Esses exemplos .mostram a você como fazer isso. Antes de alterar sua política de compromisso, revise a abordagem de vários estágios em [Como migrar e implantar](#).

C

A partir da versão 1.7.x do AWS Encryption SDK for C, a função `aws_cryptosdk_session_set_commitment_policy` passou a ser usada para definir a política de compromisso em nas suas sessões de criptografia e descriptografia. A política de compromisso que você define se aplica a todas as operações de criptografia e descriptografia chamadas na sua sessão.

As funções `aws_cryptosdk_session_new_from_keyring` e `aws_cryptosdk_session_new_from_cmm` foram descontinuadas na versão 1.7.x e foram removidas na versão 2.0.x.. Essas funções foram substituídas pelas funções `aws_cryptosdk_session_new_from_keyring_2` e `aws_cryptosdk_session_new_from_cmm_2` que retornam uma sessão.

Ao usar `aws_cryptosdk_session_new_from_keyring_2` e `aws_cryptosdk_session_new_from_cmm_2` na versão 1.x mais recentes, você deve

chamar a função `aws_cryptosdk_session_set_commitment_policy` com o valor da política de compromisso `COMMITMENT_POLICY_FORBID_ENCRYPT_ALLOW_DECRYPT`. Nas versões 2.0.x e posteriores, chamar essa função é opcional, e ela aceita todos os valores válidos. A política de compromisso padrão para as versões 2.0. x e posteriores é `COMMITMENT_POLICY_REQUIRE_ENCRYPT_REQUIRE_DECRYPT`.

Para obter um exemplo completo, consulte [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

O valor `require-encrypt-require-decrypt` é a política de compromisso padrão em todas as versões do AWS Encryption SDK para .NET. Você pode definir isso explicitamente como uma prática recomendada, mas isso não é necessário. No entanto, se estiver usando o AWS Encryption SDK para .NET para descriptografar texto cifrado criptografado por outra implementação de linguagem do AWS Encryption SDK sem confirmação de chave, você precisará alterar o valor da política de compromisso para `REQUIRE_ENCRYPT_ALLOW_DECRYPT` ou `FORBID_ENCRYPT_ALLOW_DECRYPT`. Caso contrário, ocorrerá uma falha na tentativa de descriptografar o texto cifrado.

No AWS Encryption SDK para .NET, a política de compromisso deve ser definida em uma instância do AWS Encryption SDK. Instancie um objeto `AwsEncryptionSdkConfig` com um parâmetro `CommitmentPolicy` e use o objeto de configuração para criar a instância do AWS Encryption SDK. Em seguida, chame os métodos `Encrypt()` e `Decrypt()` da instância do AWS Encryption SDK configurada.

Este exemplo define a política de compromisso como `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

Para definir uma política de compromisso na CLI de criptografia da AWS, use o parâmetro `--commitment-policy`. Esse parâmetro foi apresentado na versão 1.8.x..

No versão 1.x mais recente, quando você usa o parâmetro `--wrapping-keys` em um comando `--encrypt` ou `--decrypt`, é necessário um parâmetro `--commitment-policy` com o valor `forbid-encrypt-allow-decrypt`. Caso contrário, o parâmetro `--commitment-policy` será inválido.

Nas versões 2.1.x e posteriores, o parâmetro `--commitment-policy` é opcional e usa como padrão o valor `require-encrypt-require-decrypt`, que não criptografará nem descriptografará nenhum texto cifrado criptografado sem confirmação de chave. No entanto, recomendamos definir a política de compromisso de forma explícita em todas as chamadas de criptografia e descriptografia, para ajudar na manutenção e na solução de problemas.

Este exemplo define a política de compromisso como `.`. Ele também usa o parâmetro `--wrapping-keys`, que substituiu o parâmetro `--master-keys` a partir da versão 1.8.x.. Para obter mais detalhes, consulte [the section called “Como atualizar provedores de chaves mestras do AWS KMS”](#). Para obter exemplos completos, consulte [Exemplos da CLI de criptografia da AWS](#).

```
\\ 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

A partir da versão 1.7. x do AWS Encryption SDK for Java, a política de compromisso passou a ser definida na sua instância do objeto `AwsCrypto` que representa o cliente de AWS Encryption SDK. Essa definição política de compromisso se aplica a todas as operações de criptografia e descriptografia chamadas nesse cliente.

O construtor `AwsCrypto()` foi descontinuado nas versões 1.x do AWS Encryption SDK for Java mais recentes e foi removido na versão 2.0. x.. Ele foi substituído por uma nova classe `Builder`, um método `Builder.withCommitmentPolicy()` e pelo tipo enumerado `CommitmentPolicy`.

Nas versões 1.x mais recentes, a classe `Builder` requer o método `Builder.withCommitmentPolicy()` e o argumento `CommitmentPolicy.ForbidEncryptAllowDecrypt`. A partir da versão 2.0.x, o método `Builder.withCommitmentPolicy()` é opcional. O valor padrão é `CommitmentPolicy.RequireEncryptRequireDecrypt`.

Para obter um exemplo completo, consulte [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

A partir da versão 1.7.x do AWS Encryption SDK para JavaScript, é possível definir a política de compromisso ao chamar a nova função `buildClient` que instancia um cliente AWS Encryption SDK. A função `buildClient` assume um valor enumerado que representa sua política de compromisso. Ela retorna as funções `encrypt` e `decrypt` atualizadas, que reforçam sua política de compromisso quando você criptografa e descriptografa.

Nas versões 1.x mais recentes, a função `buildClient` requer o argumento `CommitmentPolicy.FORBID_ENCRYPT_ALLOW_DECRYPT`. A partir da versão 2.0.x, o argumento da política de compromisso é opcional, e o valor padrão é `CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT`.

O código para Node.js e o navegador são idênticos para essa finalidade, exceto que o navegador precisa de uma instrução para definir as credenciais.

O exemplo a seguir criptografa os dados com um token de autenticação do AWS KMS. A nova função `buildClient` define a política de compromisso como `FORBID_ENCRYPT_ALLOW_DECRYPT`, o valor padrão nas versões 1.x. mais recentes. A funções `encrypt` e `decrypt` atualizadas retornadas por `buildClient` reforçam a política de compromisso que você definiu.

```
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

A partir da versão 1.7. x do AWS Encryption SDK for Python, a política de compromisso passou a ser definida na sua instância de um novo objeto `EncryptionSDKClient` que representa o cliente AWS Encryption SDK. A política de compromisso que você define se aplica a todas as chamadas `encrypt` e `decrypt` que usam essa instância do cliente.

Nas versões 1.x mais recentes, o construtor `EncryptionSDKClient` requer o valor enumerado `CommitmentPolicy.FORBID_ENCRYPT_ALLOW_DECRYPT`. A partir da versão 2.0.x, o argumento da política de compromisso é opcional, e o valor padrão é `CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT`.

Este exemplo usa o novo construtor `EncryptionSDKClient` e define a política de compromisso como o valor padrão de 1.7.x. O construtor instancia um cliente que representa o AWS Encryption SDK. Quando você chama os métodos `encrypt`, `decrypt` ou `stream` desse cliente, eles aplicam a política de compromisso que você definiu. Este exemplo também usa o novo construtor da classe `StrictAwsKmsMasterKeyProvider`, que especifica AWS KMS keys ao criptografar e descriptografar.

Para obter um exemplo completo, consulte [set_commitment.py](#).

```
# Instantiate the client
client =
    aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.FORBID_ENCRYPT_ALLOW_DECRYPT)

// 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  
  )
```

Solução de problemas de migração para as versões mais recentes

Antes de atualizar a aplicação para a versão 2.0.x ou versões posteriores do AWS Encryption SDK, atualize para a versão 1.x mais recente do AWS Encryption SDK e implante-a completamente. Isso ajudará você a evitar a maioria dos erros que pode encontrar ao atualizar para as versões 2.0.x e posteriores. Para obter orientações detalhadas, incluindo exemplos, consulte [Como migrar seu AWS Encryption SDK](#).

Important

Verifique se a sua versão 1.x mais recente é a versão 1.7.x ou versão posterior do AWS Encryption SDK.

Note

CLI de criptografia da AWS: as referências neste guia à versão 1.7.x do AWS Encryption SDK se aplicam à versão 1.8.x da CLI de criptografia da AWS. As referências neste guia para a versão 2.0.x do AWS Encryption SDK se aplicam à versão 2.1.x da CLI de criptografia da AWS.

Os novos atributos de segurança foram lançados originalmente nas versões 1.7.x e 2.0.x da CLI de criptografia da AWS. No entanto, a versão 1.8x da CLI de criptografia da AWS substituiu a versão 1.7.x e a versão 2.1.x da CLI de criptografia da AWS substituiu a 2.0.x. Para obter detalhes, consulte o [boletim de segurança](#) pertinente no repositório [aws-encryption-sdk-cli](#) no GitHub.

Este tópico foi criado para ajudar você a reconhecer e solucionar os erros mais comuns que pode encontrar.

Tópicos

- [Objetos descontinuados ou removidos](#)
- [Conflito de configuração: política de compromisso e pacote de algoritmos](#)

- [Conflito de configuração: política de compromisso e texto cifrado](#)
- [Falha na validação do confirmação de chave](#)
- [Outras falhas de criptografia](#)
- [Outras falhas de descriptografia](#)
- [Considerações sobre reversão](#)

Objetos descontinuados ou removidos

A versão 2.0.x inclui várias alterações importantes, incluindo a remoção de construtores, métodos, funções e classes herdadas que foram descontinuados na versão 1.7.x.. Para evitar erros do compilador, erros de importação, erros de sintaxe e erros de símbolo não encontrado (dependendo da sua linguagem de programação), atualize primeiro para versão 1.x mais recente do AWS Encryption SDK para sua linguagem de programação. (Deve ser a versão 1.7.x ou posterior) Ao usar a versão 1.x mais recente, você pode começar a usar os elementos de substituição antes que os símbolos originais sejam removidos.

Se precisar atualizar para a versão 2.0. x ou posterior imediatamente, [consulte o changelog](#) da sua linguagem de programação e substitua os símbolos herdados pelos símbolos recomendados pelo changelog.

Conflito de configuração: política de compromisso e pacote de algoritmos

Se você especificar um pacote de algoritmos que entre em conflito com sua [política de compromisso](#), a chamada para criptografar falhará com um erro de conflito de configuração.

Para evitar esse tipo de erro, não especifique um pacote de algoritmos. Por padrão, o AWS Encryption SDK escolhe o algoritmo mais seguro que seja compatível com sua política de compromisso. No entanto, se você precisar especificar um pacote de algoritmos, como um sem assinatura, certifique-se de escolher um pacote de algoritmos que seja compatível com sua política de compromisso.

Política de compromisso	Pacotes de algoritmos compatíveis
ForbidEncryptAllowDecrypt	Qualquer pacote de algoritmos sem confirmação de chave, como:

Política de compromisso	Pacotes de algoritmos compatíveis
	<p>AES_256_GCM_IV12_TAG16_HKDF_SHA384_ECDSA_P384 (03 78) (com assinatura)</p> <p>AES_256_GCM_IV12_TAG16_HKDF_SHA256 (01 78) (sem assinatura)</p>
<p>RequireEncryptAllowDecrypt</p> <p>RequireEncryptRequireDecrypt</p>	<p>Qualquer pacote de algoritmos com confirmação de chave, como:</p> <p>AES_256_GCM_HKDF_SHA512_COM_MIT_KEY_ECDSA_P384 (05 78) (com assinatura)</p> <p>AES_256_GCM_HKDF_SHA512_COM_MIT_KEY (04 78) (sem assinatura)</p>

Se você encontrar esse erro sem ter especificado um pacote de algoritmos, o pacote de algoritmos conflitante pode ter sido escolhido pelo seu [gerenciador de materiais criptográficos](#) (CMM). O CMM padrão não selecionará um pacote de algoritmos conflitantes, mas um CMM personalizado pode fazer isso. Para obter ajuda, consulte a documentação do seu CMM personalizado.

Conflito de configuração: política de compromisso e texto cifrado

A [política de compromisso RequireEncryptRequireDecrypt](#) não permite que o AWS Encryption SDK descriptografe uma mensagem que foi criptografada sem [confirmação de chave](#). Se você solicitar que o AWS Encryption SDK descriptografe uma mensagem sem confirmação de chave, ele retornará um erro de Conflito de configuração.

Para evitar esse erro, antes de definir a política de compromisso `RequireEncryptRequireDecrypt`, certifique-se de que todos os textos cifrados criptografados sem confirmação de chave sejam descriptografados e recriptografados com confirmação de chave ou sejam manipulados por uma aplicação diferente. Se você encontrar esse erro, poderá retornar um erro para o texto cifrado conflitante ou alterar temporariamente sua política de compromisso para `RequireEncryptAllowDecrypt`.

Se estiver encontrando esse erro porque você atualizou para a versão 2.0.x ou posterior a partir uma versão anterior à 1.7.x sem ter atualizado primeiro para a versão 1.x mais recente (versão 1.7. x ou posterior), considere [reverter](#) para a versão 1.x mais recente e implantar essa versão em todos os hosts antes de atualizar para a versão 2.0.x ou posterior. Para obter ajuda, consulte [Como migrar e implantar o AWS Encryption SDK](#).

Falha na validação do confirmação de chave

Ao descriptografar mensagens criptografadas com confirmação de chave, você pode receber uma mensagem de erro de Falha na validação da confirmação de chave. Isso indica que a chamada de descriptografia falhou porque a chave de dados em uma [mensagem criptografada](#) não é idêntica à chave de dados exclusiva da mensagem. Ao validar a chave de dados durante a descriptografia, a [confirmação de chave](#) protege você de descriptografar uma mensagem que pode resultar em mais de um texto simples.

Esse erro indica que a mensagem criptografada que você estava tentando descriptografar não foi retornada pelo AWS Encryption SDK. Pode ser uma mensagem criada manualmente ou o resultado de dados corrompidos. Se você encontrar esse erro, a aplicação poderá rejeitar a mensagem e prosseguir ou interromper o processamento de novas mensagens.

Outras falhas de criptografia

A criptografia pode falhar por vários motivos. Você não pode usar um [token de autenticação de descoberta do AWS KMS](#) ou um [provedor de chave mestra em modo de descoberta](#) para criptografar uma mensagem.

Certifique-se de especificar um provedor de token de autenticação ou de chave mestra com chaves de encapsulamento que deem [permissão para usar](#) para criptografia. Para obter ajuda com as permissões das AWS KMS keys, consulte [Visualizar uma política de chave](#) e [Determinar o acesso a uma AWS KMS key](#) no Guia do desenvolvedor do AWS Key Management Service.

Outras falhas de descriptografia

Se sua tentativa de descriptografar uma mensagem criptografada falhar, isso significa que o AWS Encryption SDK não conseguiu (ou não pôde) descriptografar alguma das chaves de dados criptografadas na mensagem.

Se você usar um de token de autenticação ou um provedor de chave mestra que especifica chaves de encapsulamento, o AWS Encryption SDK usará somente as chaves de encapsulamento que você

especificar. Verifique se está usando as chaves de encapsulamento pretendidas e se tem permissão `kms:Decrypt` em pelo menos uma das chaves de encapsulamento. Se estiver usando AWS KMS keys, como alternativa, tente descriptografar a mensagem com um [token de autenticação de descoberta do AWS KMS](#) ou um [provedor de chave mestra em modo de descoberta](#). Se a operação for bem-sucedida, antes de retornar o texto simples, verifique se a chave usada para descriptografar a mensagem é confiável.

Considerações sobre reversão

Se a aplicação não conseguir criptografar ou descriptografar dados, geralmente, é possível resolver o problema atualizando os símbolos de código, os tokens de autenticação, os provedores de chaves mestras ou a [política de compromisso](#). No entanto, em alguns casos, você pode decidir que é melhor reverter a aplicação para uma versão anterior do AWS Encryption SDK.

Se você precisar reverter, faça isso com cuidado. Versões do AWS Encryption SDK anteriores à 1.7.x não podem descriptografar texto cifrado criptografado com [confirmação de chave](#).

- Geralmente, é seguro reverter da versão 1.x mais recente para uma versão anterior do AWS Encryption SDK. Talvez seja necessário desfazer as alterações feitas no código para usar símbolos e objetos que não são compatíveis com as versões anteriores.
- Depois de começar a criptografar com confirmação de chave (definindo sua política de compromisso como `RequireEncryptAllowDecrypt`) na versão 2.0.x ou posterior, você poderá reverter para a versão 1.7.x, mas não para alguma versão anterior a ela. Versões do AWS Encryption SDK anteriores à 1.7.x não podem descriptografar texto cifrado criptografado com [confirmação de chave](#).

Se você acidentalmente ativar a criptografia com confirmação de chave antes de que todos os hosts possam descriptografar com confirmação de chave, talvez seja melhor continuar com a implantação em vez de revertê-la. Se as mensagens forem transitórias ou puderem ser descartadas com segurança, considere fazer uma reversão com perda de mensagens. Se for necessária uma reversão, considere criar uma ferramenta que descriptografe e recriptografe todas as mensagens.

Perguntas frequentes

- [Como o AWS Encryption SDK é diferente dos SDKs da AWS?](#)
- [Qual a diferença entre o AWS Encryption SDK e o cliente de criptografia do Amazon S3?](#)
- [Quais algoritmos de criptografia são compatíveis com o AWS Encryption SDK, e qual é o padrão?](#)
- [Como o vetor de inicialização \(IV\) é gerado e onde é armazenado?](#)
- [Como cada chave de dados é gerada, criptografada e descriptografada?](#)
- [Como faço para controlar as chaves de dados que foram usadas para criptografar meus dados?](#)
- [Como o AWS Encryption SDK armazena chaves de dados criptografadas com seus dados criptografados?](#)
- [Quanto de sobrecarga o formato da mensagem do AWS Encryption SDK adiciona a meus dados criptografados?](#)
- [Posso usar meu próprio provedor de chaves mestras?](#)
- [Posso criptografar dados com mais de uma chave de encapsulamento?](#)
- [Quais tipos de dados posso criptografar com o AWS Encryption SDK?](#)
- [Como o AWS Encryption SDK criptografa e descriptografa streams de entrada/saída \(E/S\)?](#)

Como o AWS Encryption SDK é diferente dos SDKs da AWS?

Os [AWSSDKs](#) fornecem bibliotecas para interagir com a Amazon Web Services (AWS), incluindo AWS Key Management Service (AWS KMS). Algumas das implementações de linguagem do AWS Encryption SDK, como [AWS Encryption SDK para .NET](#), sempre exigem o AWS SDK na mesma linguagem de programação. Outras implementações de linguagem somente exigem o SDK da AWS correspondente quando você usa chaves AWS KMS em seus tokens de autenticação ou provedores de chaves mestras. Para obter detalhes, consulte o tópico sobre sua linguagem de programação em [Linguagens de programação do AWS Encryption SDK](#).

Você pode usar os AWS SDKs para interagir com o AWS KMS, incluindo criptografar e descriptografar pequenas quantidades de dados (até 4.096 bytes com uma chave de criptografia simétrica) e gerar chaves de dados para criptografia do lado do cliente. No entanto, ao gerar uma chave de dados, você deve gerenciar todo o processo de criptografia e descriptografia, o que inclui criptografar dados com a chave de dados fora do AWS KMS, descartar com segurança a chave de dados de texto simples, armazenar a chave de dados criptografada e, em seguida,

descriptografar a chave de dados e descriptografar seus dados. O AWS Encryption SDK gerencia esse processo para você.

O AWS Encryption SDK fornece uma biblioteca que criptografa e descriptografa dados usando os padrões e as práticas recomendadas do setor. Ele gera a chave de dados, criptografa-a com as chaves de encapsulamento especificadas e retorna uma mensagem criptografada, um objeto de dados portátil que inclui os dados criptografados e as chaves de dados criptografadas necessárias para descriptografá-los. Na hora de descriptografar, você passa a mensagem criptografada e pelo menos uma das chaves de encapsulamento (opcional), e o AWS Encryption SDK retorna seus dados em texto sem formatação.

Você pode usar AWS KMS keys como chaves de encapsulamento no AWS Encryption SDK, mas isso não é obrigatório. Você pode usar as chaves de criptografia geradas por você e as do seu gerenciador de chaves ou módulo de segurança de hardware on-premises. Você pode usar o AWS Encryption SDK mesmo se não tem uma conta da AWS.

Qual a diferença entre o AWS Encryption SDK e o cliente de criptografia do Amazon S3?

O [cliente de criptografia do Amazon S3](#) nos AWS SDKs fornece criptografia e descriptografia dos dados que você armazena no Amazon Simple Storage Service (Amazon S3). Esses clientes são totalmente acoplados ao Amazon S3 e são destinados para uso apenas com os dados armazenados ali.

O AWS Encryption SDK fornece criptografia e descriptografia de dados que você pode armazenar em qualquer lugar. Os clientes de criptografia do AWS Encryption SDK e do Amazon S3 não são compatíveis, pois eles produzem textos cifrados com diferentes formatos de dados.

Quais algoritmos de criptografia são compatíveis com o AWS Encryption SDK, e qual é o padrão?

O AWS Encryption SDK usa o algoritmo Advanced Encryption Standard (AES – Padrão avançado de criptografia) no Galois/Counter Mode (GCM), conhecido como AES-GCM, para criptografar dados brutos. Ele permite que você escolha entre vários algoritmos simétricos e assimétricos para criptografar as chaves de dados que criptografam seus dados.

Para o AES-GCM, o pacote de algoritmos padrão é o AES-GCM com chave de 256 bits, derivação de chave (HKDF), [assinaturas digitais](#) e [confirmação de chave](#). O AWS Encryption SDK também oferece suporte a chaves de criptografia de 192 e 128 bits e algoritmos de criptografia sem assinaturas digitais e confirmação de chaves.

Em todos os casos, o tamanho do vetor de inicialização (IV) é de 12 bytes, e o tamanho da tag de autenticação é de 16 bytes. Por padrão, o SDK usa a chave de dados como entrada para a HKDF

(HMAC-based extract-and-expand key derivation function - Função de derivação de chaves de extração e expansão com base em HMAC) para derivar a chave de criptografia do AES-GCM, e também adiciona uma assinatura Elliptic Curve Digital Signature Algorithm (ECDSA - Algoritmo de assinatura digital de curva elíptica).

Para obter informações sobre como escolher o algoritmo a ser usado, consulte [Pacotes de algoritmos compatíveis](#).

Para obter detalhes sobre a implementação de algoritmos compatíveis, consulte [Referência de algoritmos](#).

Como o vetor de inicialização (IV) é gerado e onde é armazenado?

O AWS Encryption SDK usa um método determinístico para construir um IV valor diferente para cada moldura. Esse procedimento garante que os IVs nunca sejam repetidos em uma mensagem. (Antes da versão 1.3.0 do AWS Encryption SDK for Java e do AWS Encryption SDK for Python, o AWS Encryption SDK gerava aleatoriamente um valor IV exclusivo para cada moldura.)

O SDK armazena o IV na mensagem criptografada que o AWS Encryption SDK retorna. Para obter mais informações, consulte [AWS Encryption SDK referência de formato de mensagem](#).

Como cada chave de dados é gerada, criptografada e descriptografada?

O método depende do token de autenticação ou do provedor de chave mestra que você usa.

Os tokens de autenticação e provedores de chaves mestras do AWS KMS no AWS Encryption SDK usam a operação da API [GenerateDataKey](#) do AWS KMS para gerar cada chave de dados e criptografá-las sob sua chave de encapsulamento. Para criptografar cópias da chave de dados em chaves do KMS adicionais, eles usam a operação [Criptografar](#) do AWS KMS. Para descriptografar uma chave de dados, use a operação [Descriptografar](#) do AWS KMS. Para obter detalhes, consulte [token de autenticação do AWS KMS](#) na Especificação do AWS Encryption SDK no GitHub.

Outros tokens de autenticação geram a chave de dados, criptografam e descriptografam usando os métodos das práticas recomendadas para cada linguagem de programação. Para obter detalhes, consulte a especificação do fornecedor do token de autenticação ou da chave mestra na seção [Estrutura](#) da Especificação AWS Encryption SDK no GitHub.

Como faço para controlar as chaves de dados que foram usadas para criptografar meus dados?

O AWS Encryption SDK faz isso para você. Ao criptografar dados, o SDK criptografa a chave de dados e armazena a chave criptografada junto com os dados criptografados na [mensagem](#)

[criptografada](#) que retorna. Ao descriptografar dados, o AWS Encryption SDK extrai a chave de dados criptografada da mensagem criptografada, descriptografa-a e usa-a para descriptografar os dados.

Como o AWS Encryption SDK armazena chaves de dados criptografadas com seus dados criptografados?

As operações de criptografia no AWS Encryption SDK retornam uma [mensagem criptografada](#), uma única estrutura de dados que contém os dados criptografados e suas chaves de dados criptografadas. O formato da mensagem consiste em pelo menos duas partes: um cabeçalho e um corpo. O cabeçalho da mensagem contém as chaves de dados criptografadas e informações sobre como o corpo da mensagem é formado. O corpo da mensagem contém os dados criptografados. Se o pacote de algoritmos incluir uma [assinatura digital](#), o formato da mensagem incluirá um rodapé que contém a assinatura. Para obter mais informações, consulte [AWS Encryption SDK referência de formato de mensagem](#).

Quanto de sobrecarga o formato da mensagem do AWS Encryption SDK adiciona a meus dados criptografados?

A quantidade de sobrecarga adicionada pelo AWS Encryption SDK depende de vários fatores, incluindo o seguinte:

- O tamanho dos dados de texto descriptografado
- Qual dos algoritmos compatíveis é usado
- Se dados autenticados adicionais (AAD) são fornecidos e o tamanho desse AAD
- O número e o tipo de chave de encapsulamento ou chave mestra
- O tamanho da moldura (quando [dados com moldura](#) são usados)

Quando você usa o AWS Encryption SDK com a configuração padrão (um AWS KMS key como chave de encapsulamento (ou chave mestra), sem AAD, dados sem moldura e com um algoritmo de criptografia com assinatura), a sobrecarga é de aproximadamente 600 bytes. Em geral, você pode pressupor de forma razoável que o AWS Encryption SDK adiciona uma sobrecarga de 1 KB ou menos, sem incluir o AAD fornecido. Para obter mais informações, consulte [AWS Encryption SDK referência de formato de mensagem](#).

Posso usar meu próprio provedor de chaves mestras?

Sim. Os detalhes da implementação variam dependendo das [linguagens de programação compatíveis](#) que você usa. No entanto, todas as linguagens compatíveis permitem que você defina [gerenciadores de material de criptografia \(CMMs\)](#), provedores de chaves mestras e chaves mestras personalizados.

Posso criptografar dados com mais de uma chave de encapsulamento?

Sim. Você pode criptografar a chave de dados com chaves de encapsulamento (ou chaves mestras) adicionais para adicionar redundância, no caso de uma estar em uma região diferente ou não estar disponível para a descriptografia.

Para criptografar dados com várias chaves de encapsulamento, crie um provedor de tokens de chave ou de chaves mestras com várias chaves de encapsulamento. Ao trabalhar com tokens de autenticação, você pode criar um [único token de autenticação com várias chaves de empacotamento](#) ou um [multitoken de autenticação](#).

Quando você criptografa dados com várias chaves de encapsulamento, o AWS Encryption SDK usa uma chave de encapsulamento para gerar uma chave de dados em texto simples. A chave de dados é exclusiva e matematicamente não está relacionada à chave de encapsulamento. A operação retorna a chave de dados em texto simples e uma cópia da chave de dados criptografada pela chave de encapsulamento. Em seguida, o método de criptografia criptografa a chave de dados com as outras chaves de encapsulamento. A [mensagem criptografada](#) resultante inclui os dados criptografados e uma chave de dados criptografada para cada chave de encapsulamento.

A mensagem resultante pode ser descriptografada usando qualquer uma das chaves de encapsulamento usadas na operação de criptografia. O AWS Encryption SDK usa a chave de encapsulamento para descriptografar a chave de dados criptografada. Em seguida, usa a chave de dados de texto simples para descriptografar os dados.

Quais tipos de dados posso criptografar com o AWS Encryption SDK?

A maioria das implementações de linguagem de programação do AWS Encryption SDK pode criptografar bytes brutos (matrizes de bytes), streams de E/S (streams de bytes) e strings. O AWS Encryption SDK para .NET não é compatível com fluxos de E/S. Fornecemos um código de exemplo para cada uma das [linguagens de programação compatíveis](#).

Como o AWS Encryption SDK criptografa e descriptografa streams de entrada/saída (E/S)?

O AWS Encryption SDK cria um stream de criptografia ou de descriptografia que empacota um stream de E/S subjacente. O fluxo de criptografia ou descriptografia executa uma operação de criptografia em uma chamada de leitura ou de gravação. Por exemplo, ele pode ler dados de texto não criptografado no fluxo subjacente e criptografá-los antes de retornar o resultado. Ou pode ler texto cifrado de um fluxo subjacente e descriptografá-lo antes de retornar o resultado. Fornecemos um código de exemplo para criptografar e descriptografar fluxos para cada uma das [linguagens de programação compatíveis](#) que oferecem suporte a streaming.

O AWS Encryption SDK para .NET não é compatível com fluxos de E/S.

AWS Encryption SDK referência

As informações nesta página são uma referência para criar sua própria biblioteca de criptografia compatível com o AWS Encryption SDK. Se você não estiver criando sua própria biblioteca de criptografia compatível, provavelmente não precisará dessas informações.

Para usar o AWS Encryption SDK em uma das linguagens de programação suportadas, consulte [Linguagens de programação](#).

Para a especificação que define os elementos de uma AWS Encryption SDK implementação adequada, consulte a [AWS Encryption SDK Especificação](#) em GitHub.

O AWS Encryption SDK usa os [algoritmos compatíveis](#) para retornar uma única estrutura de dados ou mensagem que contém dados criptografados e as chaves de dados criptografadas correspondentes. Os tópicos a seguir explicam os algoritmos e a estrutura de dados. Use essas informações para criar bibliotecas que podem ler e gravar textos cifrados compatíveis com este SDK.

Tópicos

- [AWS Encryption SDK referência de formato de mensagem](#)
- [AWS Encryption SDK exemplos de formato de mensagem](#)
- [Referência de corpo de dados autenticados adicionais \(AAD\) para o AWS Encryption SDK](#)
- [AWS Encryption SDK referência de algoritmos](#)
- [AWS Encryption SDK referência vetorial de inicialização](#)
- [AWS KMS Detalhes técnicos do chaveiro hierárquico](#)

AWS Encryption SDK referência de formato de mensagem

As informações nesta página são uma referência para criar sua própria biblioteca de criptografia compatível com o AWS Encryption SDK. Se você não estiver criando sua própria biblioteca de criptografia compatível, provavelmente não precisará dessas informações.

Para usar o AWS Encryption SDK em uma das linguagens de programação suportadas, consulte [Linguagens de programação](#).

Para a especificação que define os elementos de uma AWS Encryption SDK implementação adequada, consulte a [AWS Encryption SDK Especificação](#) em GitHub.

As operações de criptografia AWS Encryption SDK retornam uma única estrutura de dados ou [mensagem criptografada](#) que contém os dados criptografados (texto cifrado) e todas as chaves de dados criptografadas. Para compreender essa estrutura de dados ou para criar bibliotecas que a leem ou gravam nela, você precisa compreender o formato da mensagem.

O formato da mensagem consiste em pelo menos duas partes: um cabeçalho e um corpo. Em alguns casos, o formato da mensagem consiste em uma terceira parte, um rodapé. O formato da mensagem define uma sequência ordenada de bytes em ordem de bytes de rede, também chamado de formato big-endian. O formato da mensagem começa com o cabeçalho, seguido pelo corpo, seguido pelo rodapé (se houver).

Os [pacotes de algoritmos](#) suportados pelo AWS Encryption SDK usam uma das duas versões de formato de mensagem. Os pacotes de algoritmos sem [confirmação de chave](#) usam formato de mensagem versão 1. Os pacotes de algoritmos com confirmação de chave usam formato de mensagem versão 2.

Tópicos

- [Estrutura do cabeçalho](#)
- [Estrutura do corpo](#)
- [Estrutura do rodapé](#)

Estrutura do cabeçalho

O cabeçalho da mensagem contém a chave de dados criptografada e informações sobre como o corpo da mensagem é formado. A tabela a seguir descreve os campos que formam o cabeçalho nas versões 1 e 2 do formato de mensagem. Os bytes são anexados na ordem mostrada.

O valor Não presente indica que o campo não existe nessa versão do formato de mensagem. O texto em negrito indica valores que são diferentes em cada versão.

Note

Talvez seja necessário rolar horizontalmente ou verticalmente para ver todos os dados nessa tabela.

Estrutura do cabeçalho

Campo	Formato da mensagem versão 1	Formato da mensagem versão 2
	Tamanho (bytes)	Tamanho (bytes)
Version (Versão)	1	1
Tipo	1	Não está presente
ID do algoritmo	2	2
ID de mensagem	16	32
Tamanho do AAD	2	2
	Quando o contexto de criptografia está vazio, o valor do tamanho do AAD de 2 bytes é 0.	Quando o contexto de criptografia está vazio, o valor do tamanho do AAD de 2 bytes é 0.
AAD	Variável. O tamanho desse campo aparece nos 2 bytes anteriores (campo Tamanho do AAD). Quando o contexto de criptografia está vazio, não há um campo AAD no cabeçalho.	Variável. O tamanho desse campo aparece nos 2 bytes anteriores (campo Tamanho do AAD). Quando o contexto de criptografia está vazio, não há um campo AAD no cabeçalho.
Contagem de chaves de dados criptografadas	2	2
Chaves de dados criptografadas	Variável. Determinado pelo número de chaves de dados criptografadas e pelo tamanho de cada uma delas.	Variável. Determinado pelo número de chaves de dados criptografadas e pelo tamanho de cada uma delas.
Tipo de conteúdo	1	1

Campo	Formato da mensagem versão 1	Formato da mensagem versão 2
	Tamanho (bytes)	Tamanho (bytes)
<u>Reservado</u>	4	Não está presente
<u>Tamanho do IV</u>	1	Não está presente
<u>Tamanho da moldura</u>	4	4
<u>Dados do pacote de algoritmo</u> <u>s</u>	Não está presente	Variável. Determinado pelo algoritmo que gerou a mensagem.
<u>Autenticação do cabeçalho</u>	Variável. Determinado pelo algoritmo que gerou a mensagem.	Variável. Determinado pelo algoritmo que gerou a mensagem.

Version (Versão)

A versão do formato desta mensagem. A versão é 1 ou 2 codificado como o byte 01 ou 02 em notação hexadecimal

Tipo

O tipo deste formato de mensagem. O tipo indica o tipo da estrutura. O único tipo suportado é descrito como dados criptografados e autenticados pelo cliente. Seu valor de tipo é 128 bytes, codificado como byte 80 em notação hexadecimal.

Esse campo não está presente na versão 2 do formato de mensagem.

ID do algoritmo

Um identificador para o algoritmo usado. É um valor de 2 bytes interpretado como um inteiro não assinado de 16 bits. Para obter mais informações sobre os algoritmos, consulte [AWS Encryption SDK referência de algoritmos](#).

ID de mensagem

Um valor gerado aleatoriamente que identifica a mensagem. O ID da mensagem:

- Identifica exclusivamente a mensagem criptografada.
- Associa levemente o cabeçalho da mensagem ao corpo da mensagem.
- Fornece um mecanismo para reutilizar uma chave de dados com segurança com várias mensagens criptografadas.
- Protege contra a reutilização acidental de uma chave de dados ou contra o desgaste de chaves no AWS Encryption SDK.

Esse valor é de 128 bits no formato de mensagem versão 1 e 256 bits na versão 2.

Tamanho do AAD

O tamanho dos dados autenticados adicionais (AAD). É um valor de 2 bytes interpretado como um número inteiro não assinado de 16 bits que especifica o número de bytes que contém o AAD.

Quando o [contexto de criptografia](#) está vazio, o valor do campo de tamanho do AAD é 0.

AAD

Os dados autenticados adicionais. O AAD é uma codificação do [contexto de criptografia](#), uma matriz de pares de chave-valor onde cada chave e o valor é uma string de caracteres codificados em UTF-8. O contexto de criptografia é convertido em uma sequência de bytes e usado para o valor do AAD. Quando o contexto de criptografia está vazio, não há um campo AAD no cabeçalho.

Quando os [algoritmos com assinatura](#) são usados, o contexto de criptografia deve conter o par de chave-valor {'aws-crypto-public-key', Qtxt}. Qtxt representa o ponto Q compactado da curva elíptica de acordo com o [SEC 1 versão 2.0](#) e, em seguida, codificado em base64. O contexto de criptografia pode conter valores adicionais, mas o tamanho máximo do construído AAD é $2^{16} - 1$ bytes.

A tabela a seguir descreve os campos que formam o AAD. Os pares de chave-valor são classificados, por chave, em ordem crescente de acordo com o código de caracteres UTF-8. Os bytes são anexados na ordem mostrada.

Estrutura do AAD

Campo	Tamanho (bytes)
Contagem de pares de chave-valor	2

Campo	Tamanho (bytes)
Comprimento da chave	2
Chave	Variável. Igual ao valor especificado nos 2 bytes anteriores (tamanho da chave).
Tamanho do valor	2
Valor	Variável. Igual ao valor especificado nos 2 bytes anteriores (tamanho do valor).

Contagem de pares de chave-valor

O número de pares chave-valor no AAD. É um valor de 2 bytes interpretado como um número inteiro não assinado de 16 bits que especifica o número de pares de chave-valor no AAD. O número máximo de pares chave-valor no AAD é $2^{16} - 1$.

Quando não houver um contexto de criptografia ou o contexto de criptografia estiver vazio, esse campo não estará presente na estrutura do AAD.

Comprimento da chave

O tamanho da chave do par de chave-valor. É um valor de 2 bytes interpretado como um número inteiro não assinado de 16 bits que especifica o número de bytes que contém a chave.

Chave

A chave do par de chave-valor. É uma sequência de bytes codificados em UTF-8.

Tamanho do valor

O tamanho do valor do par de chave-valor. É um valor de 2 bytes interpretado como um número inteiro não assinado de 16 bits que especifica o número de bytes que contém o valor.

Valor

O valor do par de chave-valor. É uma sequência de bytes codificados em UTF-8.

Contagem de chaves de dados criptografadas

O número de chaves de dados criptografadas. É um valor de 2 bytes interpretado como um número inteiro não assinado de 16 bits que especifica o número de chaves de dados

criptografadas. O número máximo de chaves de dados criptografadas em cada registro é 65.535 ($2^{16} - 1$).

Chaves de dados criptografadas

Uma sequência de chaves de dados criptografadas. O tamanho da sequência é determinado pelo número de chaves de dados criptografadas e pelo tamanho de cada uma delas. A sequência contém pelo menos uma chave de dados criptografada.

A tabela a seguir descreve os campos que formam cada chave de dados criptografada. Os bytes são anexados na ordem mostrada.

Estrutura da chave de dados criptografada

Campo	Tamanho (bytes)
Tamanho do ID do provedor de chave	2
ID do provedor de chave	Variável. Igual ao valor especificado nos 2 bytes anteriores (tamanho do ID do provedor de chave).
Tamanho das informações do provedor de chave	2
Informações do provedor de chave	Variável. Igual ao valor especificado nos 2 bytes anteriores (tamanho das informações do provedor de chave).
Tamanho da chave de dados criptografada	2
Chave de dados criptografada	Variável. Igual ao valor especificado nos 2 bytes anteriores (tamanho da chave de dados criptografada).

Tamanho do ID do provedor de chave

O tamanho do identificador do provedor de chave. É um valor de 2 bytes interpretado como um número inteiro não assinado de 16 bits que especifica o número de bytes que contém o ID do provedor de chave.

ID do provedor de chave

O identificador do provedor de chave. É usado para indicar o provedor da chave de dados criptografada e deve ser extensível.

Tamanho das informações do provedor de chave

O tamanho das informações do provedor de chave. É um valor de 2 bytes interpretado como um número inteiro não assinado de 16 bits que especifica o número de bytes que contém as informações do provedor de chave.

Informações do provedor de chave

As informações do provedor de chave. São determinadas pelo provedor de chaves.

Quando AWS KMS é o provedor da chave mestra ou você está usando um AWS KMS chaveiro, esse valor contém o Amazon Resource Name (ARN) do AWS KMS key

Tamanho da chave de dados criptografada

O tamanho da chave de dados criptografada. É um valor de 2 bytes interpretado como um número inteiro não assinado de 16 bits que especifica o número de bytes que contém a chave de dados criptografada.

Chave de dados criptografada

A chave de dados criptografada. É a chave de criptografia dos dados criptografada pelo provedor de chaves.

Tipo de conteúdo

O tipo de conteúdo criptografado, com moldura ou sem moldura.

Note

Sempre que possível, use dados com moldura. O AWS Encryption SDK suporta dados não emoldurados somente para uso antigo. Algumas implementações de linguagem do ainda AWS Encryption SDK podem gerar texto cifrado sem moldura. Todas as implementações de linguagem compatíveis podem descriptografar texto cifrado e não emoldurado.

Os dados com moldura são divididos em partes de tamanho igual; cada parte é criptografada separadamente. O conteúdo com moldura é do tipo 2, codificado como o byte 02 em notação hexadecimal.

Os dados não emoldurados não são divididos; são um único blob criptografado. O conteúdo sem moldura é do tipo 1, codificado como o byte 01 em notação hexadecimal.

Reservado

Uma sequência reservada de 4 bytes. Esse valor deve ser 0. Ele é codificado como os bytes 00 00 00 00 em notação hexadecimal (ou seja, uma sequência de 4 bytes de um valor inteiro de 32 bits igual a 0).

Esse campo não está presente na versão 2 do formato de mensagem.

Tamanho do IV

O tamanho do IV (initialization vector - vetor de inicialização). É um valor de 1 byte interpretado como um número inteiro não assinado de 8 bits que especifica o número de bytes que contém o IV. Esse valor é determinado pelo valor de bytes do IV do [algoritmo](#) que gerou a mensagem.

Este campo não está presente no formato de mensagem versão 2, que somente é compatível com pacotes de algoritmos que usam valores IV determinísticos no cabeçalho da mensagem.

Tamanho da moldura

O tamanho de cada moldura do dado com moldura. É um valor de 4 bytes interpretado como um número inteiro não assinado de 32 bits que especifica o número de bytes que forma cada estrutura. Quando o dado não for com moldura, isto é, quando o valor do campo do Content Type campo for 1, esse valor deve ser 0.

Note

Sempre que possível, use dados com moldura. O AWS Encryption SDK suporta dados não emoldurados somente para uso antigo. Algumas implementações de linguagem do ainda AWS Encryption SDK podem gerar texto cifrado sem moldura. Todas as implementações de linguagem compatíveis podem descriptografar texto cifrado e não emoldurado.

Dados do pacote de algoritmos

Dados suplementares necessários pelo [algoritmo](#) que gerou a mensagem. O tamanho e o conteúdo são determinados pelo algoritmo. Seu tamanho pode ser 0.

Este campo não está presente na versão 1 do formato de mensagem.

Autenticação do cabeçalho

A autenticação do cabeçalho é determinada pelo [algoritmo](#) que gerou a mensagem. A autenticação do cabeçalho é calculada sobre o cabeçalho inteiro. Consiste em um IV e uma tag de autenticação. Os bytes são anexados na ordem mostrada.

Estrutura da autenticação do cabeçalho

Campo	Tamanho na versão 1.0 (bytes)	Tamanho na versão 2.0 (bytes)
IV	Variável. Determinada pelo valor de bytes do IV do algoritmo que gerou a mensagem.	N/D
Tag de autenticação	Variável. Determinada pelo valor dos bytes da tag de autenticação do algoritmo que gerou a mensagem.	Variável. Determinada pelo valor dos bytes da tag de autenticação do algoritmo que gerou a mensagem.

IV

O vetor de inicialização (IV) usado para calcular a tag de autenticação do cabeçalho.

Este campo não está presente na versão 2 do formato de mensagem. Este campo não está presente na versão 2 do formato de mensagem, que somente é compatível com pacotes de algoritmos que usam valores IV determinísticos no cabeçalho da mensagem.

Tag de autenticação

O valor da autenticação do cabeçalho. É usado para autenticar todo o conteúdo do cabeçalho.

Estrutura do corpo

O corpo da mensagem contém os dados criptografados, chamados de texto cifrado. A estrutura do corpo depende do tipo de conteúdo (sem moldura ou com moldura). As seções a seguir descrevem o formato do corpo da mensagem para cada tipo de conteúdo. A estrutura do corpo da mensagem é a mesma nas versões 1 e 2 do formato de mensagem.

Tópicos

- [Dados sem moldura](#)
- [Dados com moldura](#)

Dados sem moldura

Os dados sem moldura são criptografados em um único blob com um IV exclusivo e [AAD do corpo](#).

Note

Sempre que possível, use dados com moldura. O AWS Encryption SDK suporta dados não emoldurados somente para uso antigo. Algumas implementações de linguagem do ainda AWS Encryption SDK podem gerar texto cifrado sem moldura. Todas as implementações de linguagem compatíveis podem descriptografar texto cifrado e não emoldurado.

A tabela a seguir descreve os campos que formam os dados sem moldura. Os bytes são anexados na ordem mostrada.

Estrutura de corpo sem moldura

Campo	Tamanho, em bytes
IV	Variável. Igual ao valor especificado no byte do Tamanho do IV do cabeçalho.
Tamanho do conteúdo criptografado	8
Conteúdo criptografado	Variável. Igual ao valor especificado nos 8 bytes anteriores (tamanho do conteúdo criptografado).

Campo	Tamanho, em bytes
Tag de autenticação	Variável. Determinado pela implementação do algoritmo usado.

IV

O vetor de inicialização (IV) para uso com o [algoritmo de criptografia](#).

Tamanho do conteúdo criptografado

O tamanho do conteúdo criptografado ou do texto cifrado. É um valor de 8 bytes interpretado como um número inteiro não assinado de 64 bits que especifica o número de bytes que contém o conteúdo criptografado.

Tecnicamente, o valor máximo permitido é $2^{63} - 1$ ou 8 exbibytes (8 EiB). No entanto, na prática, o valor máximo é $2^{36} - 32$ ou 64 gibibytes (64 GiB), devido às restrições impostas pelos [algoritmos implementados](#).

Note

A implementação Java deste SDK restringe ainda mais esse valor para $2^{31} - 1$ ou 2 gibibytes (2 GiB), devido às restrições da linguagem.

Conteúdo criptografado

O conteúdo criptografado (texto cifrado) como retornado pelo [algoritmo de criptografia](#).

Tag de autenticação

O valor da autenticação do corpo. É usado para autenticar o corpo da mensagem.

Dados com moldura

Em dados com moldura, os dados de texto simples são divididos em partes iguais chamadas molduras. O AWS Encryption SDK criptografa cada quadro separadamente com um IV e um [corpo AAD](#) exclusivos.

Note

Sempre que possível, use dados com moldura. O AWS Encryption SDK suporta dados não emoldurados somente para uso antigo. Algumas implementações de linguagem do ainda AWS Encryption SDK podem gerar texto cifrado sem moldura. Todas as implementações de linguagem compatíveis podem descriptografar texto cifrado e não emoldurado.

O [tamanho da moldura](#), que é o tamanho do [conteúdo criptografado](#) na moldura, pode ser diferente para cada mensagem. O número máximo de bytes em uma moldura é $2^{32} - 1$. O número máximo de molduras em uma mensagem é $2^{32} - 1$.

Há dois tipos de moldura: normal e final. Cada mensagem deve consistir ou incluir uma moldura final.

Todas as molduras normais em uma mensagem têm o mesmo tamanho de moldura. A moldura final pode ter um tamanho de moldura diferente.

A composição das molduras em dados com molduras varia de acordo com o tamanho do conteúdo criptografado.

- Igual ao tamanho do quadro: quando o tamanho do conteúdo criptografado é igual ao tamanho do quadro das molduras regulares, a mensagem pode consistir em um quadro normal que contém os dados, seguido por um quadro final de tamanho zero (0). Ou, a mensagem pode consistir apenas em um a moldura que contém os dados. Nesse caso, a moldura final tem o mesmo tamanho de moldura que as molduras normais.
- Múltiplo do tamanho do quadro: quando o tamanho do conteúdo criptografado é um múltiplo exato do tamanho do quadro das molduras regulares, a mensagem pode terminar em um quadro regular que contém os dados, seguido por um quadro final de tamanho zero (0). Ou, a mensagem pode terminar em uma moldura final que contém os dados. Nesse caso, a moldura final tem o mesmo tamanho de moldura que as molduras normais.
- Não é múltiplo do tamanho do quadro: quando o tamanho do conteúdo criptografado não é um múltiplo exato do comprimento do quadro das molduras regulares, o quadro final contém os dados restantes. O tamanho da moldura final é menor que o tamanho das molduras normais.
- Menor que o tamanho do quadro: quando o tamanho do conteúdo criptografado é menor que o tamanho do quadro das molduras regulares, a mensagem consiste em um quadro final que contém todos os dados. O tamanho da moldura final é menor que o tamanho das molduras normais.

As tabelas a seguir descrevem os campos que formam as molduras. Os bytes são anexados na ordem mostrada.

Estrutura de corpo com moldura, moldura normal

Campo	Tamanho, em bytes
Número de sequência	4
IV	Variável. Igual ao valor especificado no byte do Tamanho do IV do cabeçalho.
Conteúdo criptografado	Variável. Igual ao valor especificado no Tamanho da moldura do cabeçalho.
Tag de autenticação	Variável. Determinada pelo algoritmo usado, conforme especificado no ID do algoritmo do cabeçalho.

Número de sequência

O número sequencial da moldura. É um número do contador incremental da moldura. É um valor de 4 bytes interpretado como um inteiro não assinado de 32 bits.

Os dados com moldura devem começar no número sequencial 1. As molduras subsequentes devem estar em ordem e devem conter um incremento de 1 da moldura anterior. Caso contrário, o processo de descryptografia será interrompido e relatará um erro.

IV

O vetor de inicialização (IV) da moldura. O SDK usa um método determinístico para construir um IV diferente para cada moldura na mensagem. O tamanho é especificado pelo [pacote de algoritmos](#) usado.

Conteúdo criptografado

O conteúdo criptografado (texto cifrado) da moldura, conforme retornado pelo [algoritmo de criptografia](#).

Tag de autenticação

O valor da autenticação da moldura. É usado para autenticar a moldura inteira.

Estrutura de corpo com moldura, moldura final

Campo	Tamanho, em bytes
Fim dos números sequenciais	4
Número de sequência	4
IV	Variável. Igual ao valor especificado no byte do Tamanho do IV do cabeçalho.
Tamanho do conteúdo criptografado	4
Conteúdo criptografado	Variável. Igual ao valor especificado nos 4 bytes anteriores (tamanho do conteúdo criptografado).
Tag de autenticação	Variável. Determinada pelo algoritmo usado, conforme especificado no ID do algoritmo do cabeçalho.

Fim dos números sequenciais

Um indicador para a moldura final. O valor é codificado como 4 bytes FF FF FF FF em notação hexadecimal.

Número de sequência

O número sequencial da moldura. É um número do contador incremental da moldura. É um valor de 4 bytes interpretado como um inteiro não assinado de 32 bits.

Os dados com moldura devem começar no número sequencial 1. As molduras subsequentes devem estar em ordem e devem conter um incremento de 1 da moldura anterior. Caso contrário, o processo de descryptografia será interrompido e relatará um erro.

IV

O vetor de inicialização (IV) da moldura. O SDK usa um método determinístico para construir um IV diferente para cada moldura na mensagem. O tamanho do IV é especificado pelo [pacote de algoritmos](#).

Tamanho do conteúdo criptografado

O tamanho do conteúdo criptografado. É um valor de 4 bytes interpretado como um número inteiro não assinado de 32 bits que especifica o número de bytes que contém o conteúdo criptografado da moldura.

Conteúdo criptografado

O conteúdo criptografado (texto cifrado) da moldura, conforme retornado pelo [algoritmo de criptografia](#).

Tag de autenticação

O valor da autenticação da moldura. É usado para autenticar a moldura inteira.

Estrutura do rodapé

Quando os [algoritmos com assinatura](#) são usados, o formato da mensagem contém um rodapé. O rodapé da mensagem contém uma [assinatura digital](#) calculada sobre o cabeçalho e o corpo da mensagem. A tabela a seguir descreve os campos que formam o rodapé. Os bytes são anexados na ordem mostrada. A estrutura do rodapé da mensagem é a mesma nas versões 1 e 2 do formato de mensagem.

Estrutura do rodapé

Campo	Tamanho, em bytes
Tamanho da assinatura	2
Assinatura	Variável. Igual ao valor especificado nos 2 bytes anteriores (tamanho da assinatura).

Tamanho da assinatura

O tamanho da assinatura. É um valor de 2 bytes interpretado como um número inteiro não assinado de 16 bits que especifica o número de bytes que contém a assinatura.

Assinatura

A assinatura.

AWS Encryption SDK exemplos de formato de mensagem

As informações nesta página são uma referência para criar sua própria biblioteca de criptografia compatível com o AWS Encryption SDK. Se você não estiver criando sua própria biblioteca de criptografia compatível, provavelmente não precisará dessas informações.

Para usar o AWS Encryption SDK em uma das linguagens de programação suportadas, consulte [Linguagens de programação](#).

Para a especificação que define os elementos de uma AWS Encryption SDK implementação adequada, consulte a [AWS Encryption SDK Especificação](#) em GitHub.

Os tópicos a seguir mostram exemplos do formato da AWS Encryption SDK mensagem. Cada exemplo mostra os bytes brutos, em notação hexadecimal, seguidos por uma descrição do que os bytes representam.

Tópicos

- [Dados emoldurados \(formato de mensagem versão 1\)](#)
- [Dados emoldurados \(formato de mensagem versão 2\)](#)
- [Dados não emoldurados \(formato de mensagem versão 1\)](#)

Dados emoldurados (formato de mensagem versão 1)

O exemplo a seguir mostra o formato da mensagem para dados com moldura na [versão 1 do formato de mensagem](#).

```
+-----+
| Header |
+-----+
01          Version (1.0)
80          Type (128, customer authenticated encrypted
  data)
0378       Algorithm ID (see Referência de algoritmos)
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+00kp0Z1cc8Tg2qd7rs5aLTg7lvmfUEW/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 ADBC213 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
00000002
F1140984 FF25F943 959BE514
216C7C6A 2234F395 F0D2D9B9 304670BF
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
FFFFFFFF
00000003
35F74F11 25410F01 DD9E04BF
00000008E
F7A53D37 2F467237 6FBD0B57 D1DFE830
B965AD1F A910AA5F 5EFFFFFF4 BC7D431C
BA9FA7C4 B25AF82E 64A04E3A A0915526
88859500 7096FABB 3ACAD32A 75CFED0C
4A4E52A3 8E41484D 270B7A0F ED61810C
3A043180 DF25E5C5 3676E449 0986557F

```

Frame 1, Authentication Tag

Frame 2, Sequence Number (2)

Frame 2, IV

Frame 2, Encrypted Content

Frame 2, Authentication Tag

Final Frame, Sequence Number End

Final Frame, Sequence Number (3)

Final Frame, IV

Final Frame, Encrypted Content Length (142)

Final Frame, Encrypted Content

```

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

```

Dados emoldurados (formato de mensagem versão 2)

O exemplo a seguir mostra o formato da mensagem para dados com moldura na [versão 2 do formato de mensagem](#).

```

+-----+
| 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
 ("QXRnM3JwOEVBnFFhNnBmaTk3MULtNTk3NHp0MnlZWE5vSmtwRHFPc0dIYkVaVDRqME50MlFkRStmbTFVY01WdThnPT0=
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
+-----+
| 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

```

Dados não emoldurados (formato de mensagem versão 1)

O exemplo a seguir mostra o formato da mensagem para dados sem moldura.

Note

Sempre que possível, use dados com moldura. O AWS Encryption SDK suporta dados não emoldurados somente para uso antigo. Algumas implementações de linguagem do ainda AWS Encryption SDK podem gerar texto cifrado sem moldura. Todas as implementações de linguagem compatíveis podem descriptografar texto cifrado e não emoldurado.

```

+-----+
| Header |
+-----+
01                                   Version (1.0)
80                                   Type (128, customer authenticated encrypted
  data)
0378                                 Algorithm ID (see Referência de algoritmos)
B8929B01 753D4A45 C0217F39 404F70FF Message ID (random 128-bit value)
008E                                 AAD Length (142)
0004                                 AAD Key-Value Pair Count (4)

```

0005	AAAD Key-Value Pair 1, Key Length (5)
30746869 73	AAAD Key-Value Pair 1, Key ("0This")
0002	AAAD Key-Value Pair 1, Value Length (2)
6973	AAAD Key-Value Pair 1, Value ("is")
0003	AAAD Key-Value Pair 2, Key Length (3)
31616E	AAAD Key-Value Pair 2, Key ("lan")
000A	AAAD Key-Value Pair 2, Value Length (10)
656E6372 79774690 6F6E	AAAD Key-Value Pair 2, Value ("encryption")
0008	AAAD Key-Value Pair 3, Key Length (8)
32636F6E 74657874	AAAD Key-Value Pair 3, Key ("2context")
0007	AAAD Key-Value Pair 3, Value Length (7)
6578616D 706C65	AAAD Key-Value Pair 3, Value ("example")
0015	AAAD Key-Value Pair 4, Key Length (21)
6177732D 63727970 746F2D70 75626C69 public-key")	AAAD Key-Value Pair 4, Key ("aws-crypto- public-key")
632D6B65 79	
0044	AAAD Key-Value Pair 4, Value Length (68)
41734738 67473949 6E4C5075 3136594B	AAAD Key-Value Pair 4, Value
("AsG8gG9InLPu16YK1qXTOD+nykG8YqHAhqcj8aXfD2e5B4gtVE73dZkyClA+rAM0Q==")	
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 D8AEFB E9 2F48E25A
978A019C FE49EE0A 0E96BF0D D6074DDB
66DFF333 0E10226F 0A1B219C BE54E4C2
2C15100C 6A2AA3F1 88251874 FDC94F6B
9247EF61 3E7B7E0D 29F3AD89 FA14A29C
76E08E9B 9ADCF8C 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	

Referência de corpo de dados autenticados adicionais (AAD) para o AWS Encryption SDK

As informações nesta página são uma referência para criar sua própria biblioteca de criptografia compatível com o AWS Encryption SDK. Se você não estiver criando sua própria biblioteca de criptografia compatível, provavelmente não precisará dessas informações.

Para usar o AWS Encryption SDK em uma das linguagens de programação suportadas, consulte [Linguagens de programação](#).

Para a especificação que define os elementos de uma AWS Encryption SDK implementação adequada, consulte a [AWS Encryption SDK Especificação](#) em GitHub.

Você deve fornecer dados autenticados adicionais (AAD) para o [algoritmo AES-GCM](#) para cada operação de criptografia. Isso é verdadeiro para [dados de corpo](#) com e sem moldura. Para obter mais informações sobre o AAD e como ele é usado no Galois/Counter Mode (GCM), consulte [Recomendações para os modos de cifra de bloco de operações: Galois/Counter Mode \(GCM\) e GMAC](#).

A tabela a seguir descreve os campos que formam o AAD do corpo. Os bytes são anexados na ordem mostrada.

Estrutura do AAD do corpo

Campo	Tamanho, em bytes
ID de mensagem	16

Campo	Tamanho, em bytes
Conteúdo do AAD do corpo	Variável. Consulte Conteúdo do AAD do corpo na lista a seguir.
Número de sequência	4
Tamanho do conteúdo	8

ID de mensagem

O mesmo valor de [ID de mensagem](#) definido no cabeçalho da mensagem.

Conteúdo do AAD do corpo

Um valor codificado em UTF-8 determinado pelo tipo de dados de corpo usado.

Para [dados sem moldura](#), use o valor `AWSKMSEncryptionClient Single Block`.

Para molduras normais em [dados com moldura](#), use o valor `AWSKMSEncryptionClient Frame`.

Para a moldura final nos [dados com moldura](#), use o valor `AWSKMSEncryptionClient Final Frame`.

Número de sequência

Um valor de 4 bytes interpretado como um inteiro não assinado de 32 bits.

Para [dados com moldura](#), esse é o número sequencial da moldura.

Para [dados sem moldura](#), use o valor 1, codificado como os 4 bytes `00 00 00 01` em notação hexadecimal.

Tamanho do conteúdo

O tamanho, em bytes, do texto não criptografado fornecido ao algoritmo para criptografia. É um valor de 8 bytes interpretado como um inteiro não assinado de 64 bits.

AWS Encryption SDK referência de algoritmos

As informações nesta página são uma referência para criar sua própria biblioteca de criptografia compatível com o AWS Encryption SDK. Se você não estiver criando sua própria biblioteca de criptografia compatível, provavelmente não precisará dessas informações.

Para usar o AWS Encryption SDK em uma das linguagens de programação suportadas, consulte [Linguagens de programação](#).

Para a especificação que define os elementos de uma AWS Encryption SDK implementação adequada, consulte a [AWS Encryption SDK Especificação](#) em GitHub.

Se você estiver criando sua própria biblioteca que pode ler e escrever textos cifrados compatíveis com o AWS Encryption SDK, você precisará entender como ela AWS Encryption SDK implementa os conjuntos de algoritmos compatíveis para criptografar dados brutos.

O AWS Encryption SDK suporta os seguintes conjuntos de algoritmos. Todos os pacotes de algoritmos AES-GCM têm um [vetor de inicialização](#) de 12 bytes e uma tag de autenticação AES-GCM de 16 bytes. O conjunto de algoritmos padrão varia de acordo com a AWS Encryption SDK versão e a política de comprometimento de chave selecionada. Para obter detalhes, consulte [Política de compromisso e pacote de algoritmos](#).

AWS Encryption SDK Suítes de algoritmos

ID do algoritmo	Versão do formato de mensagem	Algoritmo de criptografia	Tamanho da chave de dados (bits)	Algoritmo de derivação de chave	Algoritmo de assinatura	Algoritmo de confirmação de chave	Tamanho dos dados do pacote de algoritmos (bytes)
05 78	0x02	AES-GCM	256	HKDF com SHA-512	ECDSA com P-384 e SHA-384	HKDF com SHA-512	32 (confirmação de chave)

ID do algoritmo	Versão do formato de mensagem	Algoritmo de criptografia	Tamanho da chave de dados (bits)	Algoritmo de derivação de chave	Algoritmo de assinatura	Algoritmo de confirmação de chave	Tamanho dos dados do pacote de algoritmos (bytes)
04_78	0x02	AES-GCM	256	HKDF com SHA-512	Nenhum	HKDF com SHA-512	32 (confirmação de chave)
03_78	0x01	AES-GCM	256	HKDF com SHA-384	ECDSA com P-384 e SHA-384	Nenhum	N/D
03_46	0x01	AES-GCM	192	HKDF com SHA-384	ECDSA com P-384 e SHA-384	Nenhum	N/D
02_14	0x01	AES-GCM	128	HKDF com SHA-256	ECDSA com P-256 e SHA-256	Nenhum	N/D
01_78	0x01	AES-GCM	256	HKDF com SHA-256	Nenhum	Nenhum	N/D
01_46	0x01	AES-GCM	192	HKDF com SHA-256	Nenhum	Nenhum	N/D

ID do algoritmo	Versão do formato de mensagem	Algoritmo de criptografia	Tamanho da chave de dados (bits)	Algoritmo de derivação de chave	Algoritmo de assinatura	Algoritmo de confirmação de chave	Tamanho dos dados do pacote de algoritmos (bytes)
01 14	0x01	AES-GCM	128	HKDF com SHA-256	Nenhum	Nenhum	N/D
00 78	0x01	AES-GCM	256	Nenhum	Nenhum	Nenhum	N/D
00 46	0x01	AES-GCM	192	Nenhum	Nenhum	Nenhum	N/D
00 14	0x01	AES-GCM	128	Nenhum	Nenhum	Nenhum	N/D

ID do algoritmo

Um valor de 2 bytes hexadecimal que identifica exclusivamente a implementação de um algoritmo. Esse valor é armazenado no [cabeçalho da mensagem](#) do texto cifrado.

Versão do formato de mensagem

A versão do formato desta mensagem. Os pacotes de algoritmos com confirmação de chave usam formato de mensagem versão 2 (0x02). Os pacotes de algoritmos sem confirmação de chave usam formato de mensagem versão 1 (0x01).

Tamanho dos dados do pacote de algoritmos

O tamanho em bytes dos dados específicos do pacote de algoritmos. Esse campo é suportado somente no formato de mensagem versão 2 (0x02). No formato de mensagem versão 2 (0x02), esses dados aparecem no campo `Algorithm suite data` do cabeçalho da mensagem. Os conjuntos de algoritmos que são compatíveis com o [confirmação de chave](#) usam 32 bytes para a cadeia de caracteres de confirmação de chave. Para obter mais informações, consulte [Algoritmo de confirmação de chaves](#) nesta lista.

Tamanho da chave de dados

O tamanho da [chave de dados](#) em bits. O AWS Encryption SDK é compatível com chaves de 256, 192 e 128 bits. A chave de dados é gerada por um [token de autenticação](#) ou chave mestra.

Em algumas implementações, essa chave de dados é usada como entrada para uma função de derivação de extract-and-expand chave baseada em HMAC (HKDF). A saída da HKDF é usada como a chave de criptografia de dados no algoritmo de criptografia. Para obter mais informações, consulte Algoritmo de derivação de chaves nessa lista.

Algoritmo de criptografia

O nome e o modo do algoritmo de criptografia utilizado. O pacote de algoritmos do AWS Encryption SDK usa o algoritmo de criptografia Advanced Encryption Standard (AES) no Galois/Counter Mode (GCM).

Algoritmo de confirmação de chave

O algoritmo usado para calcular a string de confirmação de chave. A saída é armazenada no campo `Algorithm suite data` do cabeçalho da mensagem e é usada para validar a chave de dados para o confirmação de chave.

Para obter uma explicação técnica sobre como adicionar confirmação de chave a um conjunto de algoritmos, consulte [Confirmação de chaves AEADs](#) no Cryptology ePrintArchive.

Algoritmo de derivação de chave

A função de derivação de extract-and-expand chave baseada em HMAC (HKDF) usada para derivar a chave de criptografia de dados. O AWS Encryption SDK usa o HKDF definido na [RFC 5869](#).

Pacotes de algoritmos sem confirmação de chave (ID do algoritmo `01xx – 03xx`)

- A função de hash usada é SHA-384 ou SHA-256, dependendo do pacote de algoritmos.
- Para a etapa de extração:
 - Nenhum sal é usado. De acordo com a RFC, o sal é definido como uma string de zeros. O tamanho da string é igual ao tamanho da saída da função de hash, que é 48 bytes para SHA-384 e 32 bytes para SHA-256.
 - O material de chaveamento de entrada é a chave de dados recebida do provedor de tokens de autenticação ou de chaves mestras.
- Para a etapa de expansão:

- A chave pseudoaleatória de entrada é a saída da etapa de extração.
- As informações da entrada são uma concatenação do ID do algoritmo seguido pelo ID da mensagem (nessa ordem).
- O comprimento do material de chaveamento de saída é o Tamanho da chave de dados. Essa saída é usada como a chave de criptografia de dados no algoritmo de criptografia.

Pacotes de algoritmos com confirmação de chave (ID do algoritmo 04xx e 05xx)

- A função hash usada é SHA-512.
- Para a etapa de extração:
 - O sal é um valor aleatório criptográfico de 256 bits. No [formato de mensagem versão 2](#) (0x02), esse valor é armazenado no campo MessageID.
 - O material de chaveamento inicial é a chave de dados recebida do provedor de tokens de autenticação ou de chaves mestras.
- Para a etapa de expansão:
 - A chave pseudoaleatória de entrada é a saída da etapa de extração.
 - O rótulo da chave são os bytes codificados em UTF-8 da string DERIVEKEY na ordem de bytes big endian.
 - As informações da entrada são uma concatenação do ID do algoritmo seguido pelo rótulo de chave (nessa ordem).
 - O comprimento do material de chaveamento de saída é o Tamanho da chave de dados. Essa saída é usada como a chave de criptografia de dados no algoritmo de criptografia.

Versão do formato de mensagem

A versão do formato de mensagem usado com o conjunto de algoritmos. Para obter detalhes, consulte [Referência do formato de mensagens](#).

Algoritmo de assinatura

O algoritmo de assinatura usado para gerar uma [assinatura digital](#) sobre o cabeçalho e o corpo do texto cifrado. O AWS Encryption SDK usa o Algoritmo de Assinatura Digital de Curva Elíptica (ECDSA) com as seguintes especificações:

- A curva elíptica usada é a curva P-384 ou P-256, conforme especificado pelo ID do algoritmo. Essas curvas são definidas no [Digital Signature Standard \(DSS\) \(FIPS PUB 186-4\)](#).
- A função de hash usada é SHA-384 (com a curva P-384) ou SHA-256 (com a curva P-256).

AWS Encryption SDK referência vetorial de inicialização

As informações nesta página são uma referência para criar sua própria biblioteca de criptografia compatível com o AWS Encryption SDK. Se você não estiver criando sua própria biblioteca de criptografia compatível, provavelmente não precisará dessas informações.

Para usar o AWS Encryption SDK em uma das linguagens de programação suportadas, consulte [Linguagens de programação](#).

Para a especificação que define os elementos de uma AWS Encryption SDK implementação adequada, consulte a [AWS Encryption SDK Especificação](#) em GitHub.

[Ele AWS Encryption SDK fornece os vetores de inicialização \(IVs\) que são exigidos por todos os conjuntos de algoritmos compatíveis.](#) O SDK usa números sequenciais de molduras para construir um IV, de forma que duas molduras na mesma mensagem não podem ter o mesmo IV.

Cada IV de 96 bits (12 bytes) é construído a partir de duas matrizes de bytes big-endian concatenadas na seguinte ordem:

- 64 bits: 0 (reservado para uso futuro)
- 32 bits: o número sequencial da moldura. Para a tag de autenticação de cabeçalho, esse valor é todo de zeros.

Antes da apresentação do [armazenamento em cache de chaves de dados](#), o AWS Encryption SDK sempre usava uma nova chave de dados para criptografar cada mensagem e gerava todos os IVs aleatoriamente. Os IVs gerados aleatoriamente eram seguros criptograficamente porque as chaves de dados nunca eram reutilizadas. Quando o SDK apresentou o armazenamento em cache da chave de dados, que reutiliza intencionalmente as chaves de dados, alteramos a maneira como o SDK gera IVs.

O uso de IVs determinísticos que não podem ser repetidos em uma mensagem aumenta significativamente o número de chamadas que podem ser executadas com segurança sob uma única chave de dados. Além disso, as chaves de dados que são armazenadas em cache sempre usam um pacote de algoritmos com uma [função de derivação de chaves](#). Usar um IV determinístico com uma função de derivação de chave pseudo-aleatória para derivar chaves de criptografia de uma chave de dados permite AWS Encryption SDK criptografar 2^{32} mensagens sem exceder os limites criptográficos.

AWS KMS Detalhes técnicos do chaveiro hierárquico

O [token de autenticação hierárquico do AWS KMS](#) usa uma chave de dados exclusiva para criptografar cada campo e criptografa cada chave de dados com uma chave de empacotamento exclusiva derivada de uma chave de ramificação ativa. Ele usa uma [derivação de chave](#) no modo contador com uma função pseudoaleatória com HMAC SHA-256 para derivar a chave de empacotamento de 32 bytes com as seguintes entradas.

- Um sal aleatório de 16 bytes
- A chave de ramificação ativa
- O valor [codificado em UTF-8](#) para o identificador do provedor de chaves "" aws-kms-hierarchy

O token de autenticação hierárquico usa a chave de empacotamento derivada para criptografar uma cópia da chave de dados em texto simples usando o AES-GCM-256 com uma tag de autenticação de 16 bytes e as seguintes entradas.

- A chave de empacotamento derivada é usada como a chave de cifra AES-GCM
- A chave de dados é usada como mensagem AES-GCM
- Um vetor de inicialização aleatória (IV) de 12 bytes é usado como o AES-GCM IV
- Dados autenticados adicionais (AAD) contendo os seguintes valores serializados.

Valor	Tamanho em bytes	Interpretada como
"aws-kms-hierarchy"	17	Codificada em UTF-8
O identificador de chave de ramificação	Variável	Codificada em UTF-8
A versão da chave de ramificação	16	Codificada em UTF-8
Contexto de criptografia	Variável	Pares de valores-chave com codificação UTF-8

Histórico de documentos do AWS Encryption SDK Developer Guide

Este tópico descreve atualizações importantes no Guia do desenvolvedor do AWS Encryption SDK .

Tópicos

- [Atualizações recentes](#)
- [Atualizações anteriores](#)

Atualizações recentes

A tabela a seguir descreve alterações significativas nesta documentação desde novembro de 2017. Além das principais alterações listadas aqui, também atualizamos a documentação com frequência para melhorar as descrições e os exemplos e abordar os comentários que você nos envia. Para ser notificado sobre alterações significativas, inscreva-se no feed RSS.

Alteração	Descrição	Data
Disponibilidade geral	Foi adicionada documentação para o AWS KMS chaveiro ECDH e o chaveiro ECDH bruto .	17 de junho de 2024
AWS Encryption SDK for Java versão 3.x	Integra-o AWS Encryption SDK for Java com a biblioteca do fornecedor de materiais . Adiciona suporte para chaveiros e o contexto de criptografia necessário (CMM).	6 de dezembro de 2023
AWS Encryption SDK para o.NET versão 4.x	Adiciona suporte ao AWS KMS chaveiro hierárquico, ao contexto de criptografia necessário (CMM) e aos	12 de outubro de 2023

	chaveiros RSA assimétricos. AWS KMS	
Disponibilidade geral	Apresentando o suporte AWS Encryption SDK para o.NET.	17 de maio de 2022
Alteração na documentação	Substitua o AWS Key Management Service termo chave mestra do cliente (CMK) por uma AWS KMS keychave KMS.	30 de agosto de 2021
Disponibilidade geral	Foi adicionado suporte para AWS Key Management Service. (AWS KMS) Chaves multirregionais. As chaves multirregionais são AWS KMS chaves diferentes Regiões da AWS que podem ser usadas de forma intercambiável porque têm o mesmo ID de chave e material de chave.	8 de junho de 2021
Disponibilidade geral	Adicionada e atualizada a documentação sobre o processo de decodificação de mensagens aprimorado.	11 de maio de 2021
Disponibilidade geral	Documentação adicionada e atualizada para a versão de disponibilidade geral do AWS Encryption CLI versão 1.8. x para substituir a versão 1.7 do AWS Encryption CLI. x e CLI AWS de criptografia 2.1. x para substituir o AWS Encryption CLI 2.0. x.	27 de outubro de 2020

Disponibilidade geral	Adicionada e atualizada a documentação da versão de disponibilidade geral das versões 1.7.x e 2.0.x do AWS Encryption SDK , incluindo um guia de melhores práticas , um guia de migração , conceitos atualizados , tópicos de linguagem de programação atualizados, uma atualização de referência de pacotes de algoritmos , uma atualização de referência de formato de mensagem e um novo exemplo de formato de mensagem .	24 de setembro de 2020
Disponibilidade geral	Adicionada e atualizada a documentação para a versão de disponibilidade geral do AWS Encryption SDK para JavaScript .	1 de outubro de 2019
Versão de visualização	Adicionada e atualizada a documentação da versão beta pública do AWS Encryption SDK para JavaScript .	21 de junho de 2019
Disponibilidade geral	Adicionada e atualizada a documentação para a versão de disponibilidade geral do AWS Encryption SDK for C .	16 de maio de 2019
Versão de visualização	Documentação adicionada da versão de pré-visualização do AWS Encryption SDK for C .	5 de fevereiro de 2019

[Nova versão](#)

Documentação adicionada da [interface de linha de comando](#) para o AWS Encryption SDK.

20 de novembro de 2017

Atualizações anteriores

A tabela a seguir descreve as alterações significativas feitas no Guia do desenvolvedor do AWS Encryption SDK antes de novembro de 2017.

Alteração	Descrição	Data
Nova versão	<p>Adicionado o capítulo Armazenamento em cache de chaves de dados para o novo recurso.</p> <p>Adicionado o tópico the section called “Referência do vetor de inicialização” que explica que o SDK foi alterado da geração aleatória de IVs para a construção determinística de IVs.</p> <p>Adicionado o tópico the section called “Conceitos” para explicar conceitos, incluindo o novo gerenciador de materiais criptográficos.</p>	31 de julho de 2017
Atualizar	<p>Expandida a documentação de Referência do formato de mensagens para uma nova seção AWS Encryption SDK referência.</p>	21 de março de 2017

Alteração	Descrição	Data
	Foi adicionada uma seção sobre AWS Encryption SDK Pacotes de algoritmos compatíveis o.	
Nova versão	O AWS Encryption SDK agora suporta a linguagem de Python programação, além de Java .	21 de março de 2017
Versão inicial	Versão inicial do AWS Encryption SDK e desta documentação.	22 de março de 2016

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.